GEMLCA: running legacy code applications as grid services.

Thierry Delaitre[1]
Tamas Kiss[1]
Ariel Goyeneche[1]
Gabor Terstyanszky[1]
Stephen Winter[1]
Peter Kacsuk[2]

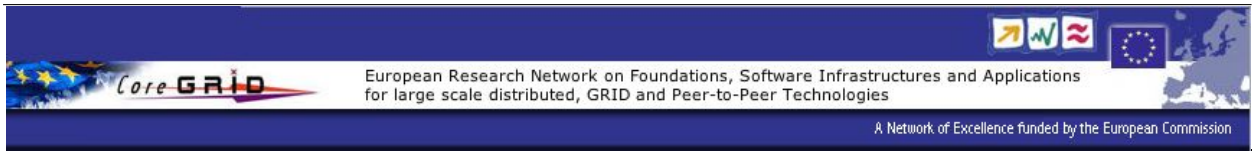
[1] School of Informatics, University of Westminster
[2] MZA SZTAKI, 1113 Kende u. 13, Budapest, Hungary

This is a reproduction of CoreGRID Technical Report Number TR-0004, April 27, 2005 and is reprinted here with permission.

The report is available on the CoreGRID website, at:

http://www.coregrid.net/mambo/images/stories/TechnicalReports/tr-0004.pdf

---



GEMLCA: Running Legacy Code Applications as Grid Services

Thierry Delaittre, Tamas Kiss*, Ariel Goyeneche*, Gabor Terstyanszky*,*
Stephen Winter and Peter Kacsuk***
**Centre of Parallel Computing, University of Westminster,*
115 New Cavendish Street,
London W1W 6UW United Kingdom
e-mail: delaitt@cpc.wmin.ac.uk
***MTA SZTAKI*
1111 Kende u. 13
Budapest, Hungary
e-mail: kacsuk@sztaki.hu

CoreGRID Technical Report
Number TR-0004
April 27, 2005

Institute on Problem Solving Environment, Tools and GRID Systems

CoreGRID - Network of Excellence
URL: http://www.coregrid.net

CoreGRID is a Network of Excellence funded by the European Commission under the Sixth Framework Programme

Project no. FP6-004265

GEMLCA: Running Legacy Code Applications as Grid Services

Thierry Delaittre*, Tamas Kiss*, Ariel Goyeneche*, Gabor Terstyanszky*,
Stephen Winter* and Peter Kacsuk**
*Centre of Parallel Computing, University of Westminster,
115 New Cavendish Street,
London W1W 6UW United Kingdom
e-mail: delaitt@cpc.wmin.ac.uk
**MTA SZTAKI
1111 Kende u. 13
Budapest, Hungary
e-mail: kacsuk@sztaki.hu

CoreGRID TR-0004

April 27, 2005

Abstract

There are many legacy code applications that cannot be run in a Grid environment without significant modification. To avoid re-engineering of legacy code, we developed the Grid Execution Management for Legacy Code Architecture (GEMLCA) that enables deployment of legacy code applications as Grid services. GEMLCA implements a general architecture for deploying legacy applications as Grid services without the need for code re-engineering, or even access to the source files. With GEMLCA, only a user-level understanding is required to run a legacy application from a standard Grid service client. The legacy code runs in its native environment using the GEMLCA resource layer to communicate with the Grid client, thus hiding the legacy nature of the application and presenting it as a Grid service. GEMLCA as a Grid service layer supports submitting jobs, getting their results and status back. The paper introduces the GEMLCA concept, its life-cycle, design and implementation. It also presents as an example a legacy simulation code that has been successfully transformed into a Grid service using GEMLCA.

1 Legacy Applications in Grid environment

Grid computing offers seamless integration of hardware and software resources, databases, special devices (like sensors or visualisation tools) and services in a geographically distributed environment. This has many potential advantages; for example, in the creation of virtual organisations for solving computationally intensive tasks or supporting collaborative work.

The Grid computing environment requires special Grid enabled applications capable of utilising the underlying Grid middleware and infrastructure. Most Grid projects so far have either developed new applications from scratch, or significantly re-engineered existing ones in order to be run on their platforms. This practice is appropriate in this context, where the applications are mainly aimed at proving the concept of the underlying architecture. However, as the Grid becomes stable and commonplace in both scientific and industrial settings, a demand will be created for porting a vast legacy of applications onto the new platform. Companies and institutions can ill afford to throw such applications away for the sake of a new technology, and there is a clear business imperative for them to be migrated onto the Grid with the least possible effort and cost.

Grid computing is now progressing to a point where reliable Grid middleware and higher level tools will be offered to support the creation of production level Grids. A high-level Grid toolkit should definitely include components

This research work is carried out under the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265).

for turning legacy applications into Grid services. The Grid Execution Management for Legacy Code Architecture (GEMLCA), presented in this paper, enables legacy code programs written in any source language (Fortran, C, Java, etc.) to be easily deployed as a Grid Service without significant user effort. GEMLCA does not require any modification of, or even access to, the original source code. A user-level understanding, describing the necessary input and output parameters and environmental values such as the number of processors or the job manager required, is all that is required to port the legacy application binary onto the Grid.

The remaining part of this paper is structured as follows: Related work and different approaches to legacy code deployment in a Grid environment are described in chapter 2. Chapter 3 outlines the general GEMLCA concept and life-cycle that can be implemented basically on top of any service-oriented Grid middleware. GEMLCA architecture, its design and implementation are discussed in chapters 4 and 5, respectively. Chapter 6 shows how security was implemented in GEMLCA. Chapter 7 illustrates how a Grid portal interface can be integrated with the solution. Chapter 8 gives an example how legacy codes were turned into Grid services and used in order to create a complex workflow application. Finally the GEMLCA vision and future works are presented in Chapter 9.

2 Related work

There are several research efforts aiming at automating the transformation of legacy code into a Grid Service. These approaches are either invasive or non-invasive. In the former, access to the source code itself is required to expose low-level functionality. In the latter, access to the code is not attempted, and exposure of internal functionality is limited to the user-level interface. Both approaches are valid in different circumstances, depending on factors such as the granularity of the code, the assumed users and application area.

In the invasive approach, it is typically assumed that an application programmer, such as a biologist or chemist with some programming background but no Grid-specific knowledge, would like to build Grid enabled applications using specific software libraries. These libraries need to be wrapped using tightly-coupled code-wrapping technology that exposes low level functionality. A network-facing container to host them is also required to make the code available for building new applications.

Most of these solutions are based on the principles outlined in [19] and use Java wrapping in order to generate stubs automatically. One prominent example is presented in [16], where the authors describe a semi-automatic conversion of programs written in C into Java using Java Native Interface (JNI). After wrapping the native C application with the Java-C Automatic Wrapper (JACAW), the MEdiation of Data and Legacy Code Interface tool (MEDLI) is used for data mapping to make the code available as part of a Grid workflow. A similar solution is described in [6] where character-based legacy systems written in Cobol are wrapped, and screen and database proxies are used to redirect input/output requests, offering a solution for interactive legacy applications.

The invasive approaches operate at low level and use wrapping technology. The common characteristics are that they are language dependent and require access to the source code. It is also necessary to extract a subset of code semantics in order to do the wrapping. These features enable application programmers to expose desired low-level code functionality.

A different approach is represented by GEMLCA, which is non-invasive. The method is relatively coarse-grained, in that the application does not allow visibility of low-level functionalities. The legacy code is provided as a black-box with specified input and output parameters and environmental requirements. Only the executable is available, and required, in this case, together with a user-level understanding of the application. This scenario is very common in both scientific and business applications when:

- the source code is not available
- the program is poorly documented and/or the necessary expertise to do any modifications has long left the organisation
- the application has to be ported onto the Grid within the shortest possible time and smallest effort and cost
- the functionalities are offered to partner organisations but the source is not.

By comparison with other solutions [5] [13] with similar aims, we see that GEMLCA offers the most comprehensive solution, since it includes portal and workflow access, security solutions incorporating authentication, authorisation and security delegation mechanisms. It also offers end-users with no programming knowledge the ability to port their applications to the Grid with relatively little effort.

3 The GEMLCA Concept and Life-cycle

GEMLCA represents a general architecture [10] for deploying legacy applications as Grid services without re-engineering the code or even requiring access to the source files. The high-level GEMLCA conceptual architecture is represented on Figure 1. As shown in the figure, there are four basic components in the architecture:

1. The *Compute Server* is a single or multiple processor computing system, including PC clusters on which several legacy codes are already implemented and available. The goal of GEMLCA is to turn these legacy codes into Grid services that can be accessed by Grid users.

2. The *Grid Host Environment* implements a service-oriented OGSA-based Grid layer, such as GT3 or GT4. This layer is a pre-requisite for connecting the Compute Server into an OGSA-built Grid. The installation of this layer is typically the task of the Compute Servers system administrator.

3. The *GEMLCA Resource layer* provides a set of Grid services which expose legacy codes as Grid services. The GEMLCA Resource layer should be installed by the Compute Servers system administrator. From now on we denote a Compute Server on which a Grid Host Environment layer and a GEMLCA Resource layer are installed as a GEMLCA Grid Resource.

4. The fourth component is the *GEMLCA Client* that can be installed on any client machine through which a user would like to access the GEMLCA resources. There are two types of GEMLCA clients. The first has a command-line interface. Both its installation and usage requires IT skills and would be difficult to use for typical end-users, like biologists, economists, etc, who dont have in-depth computing knowledge. In order to support these users, the GEMLCA client can be built into a Grid portal to provide a high-level, easy-to-use graphical interface through which any Grid user can easily access legacy codes as Grid services. In such a case no installation of the GEMLCA client is necessary. A standard web browser is enough to access the GEMLCA resources through the GEMLCA portal.

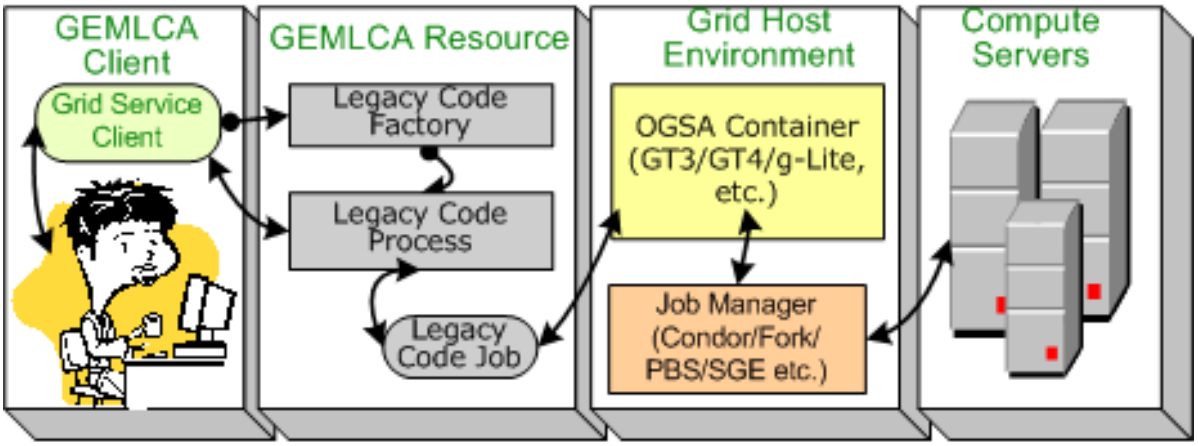


Figure 1: GEMLCA architecture

The novelty of the GEMLCA concept is that it requires minimal effort from both Compute Server administrators and end-users of the Grid. The Compute Server administrator should install the GEMLCA Resource layer on top of an available OGSA layer (GT3/GT4). It is also their task to deploy existing legacy applications on the Compute Servers as Grid services, and to make them accessible for the whole Grid community. End-users do not have to do any installation or deployment work if a GEMLCA portal is available for the Grid and they only need those legacy code services that were previously deployed by the Compute Server administrators. In such a case end-users can immediately use all these legacy code services - provided they have access to the GEMLCA Grid resources. If they would like to deploy legacy code services on GEMLCA Grid resources they can do so, but these services cannot be accessed by other Grid users (see chapter 6 on security). As a last resort, if no GEMLCA portal is available for the Grid, a user must install the GEMLCA Client on their client machine. However, since it requires some IT skills to do this, it is recommended that a GEMLCA portal is installed on every Grid where GEMLCA Grid resources are deployed.

The deployment of a GEMLCA legacy code service assumes that the legacy application runs in its native environment on a Compute Server. It is the task of the GEMLCA Resource layer to present the legacy application as a Grid service to the user, to communicate with the Grid client and to hide the legacy nature of the application. The deployment process of a GEMLCA legacy code service requires only a user-level understanding of the legacy application, ie., to know what the parameters of the legacy code are and what kind of environment is needed to run the code (eg. multiprocessor environment with n processors). The deployment defines the execution environment and the parameter set for the legacy application in an XML-based Legacy Code Interface Description (LCID) file that should be stored in a pre-defined location. This file is used by the GEMLCA Resource layer to handle the legacy application as a Grid service. The LCID file consists of three sections:

Environment. This section contains the name of the legacy code and its binary file; and defines the job manager to be used (Condor and Fork are supported by current GEMLCA implementations), the maximum number of jobs allowed to be submitted from a single legacy code process, and minimum and maximum number of processors to be used.

Description. This section describes the legacy code in a simple text format.

Parameters. This section exposes the list of parameters, defining for each of them its name, friendly name, type (input or output), order, status (compulsory or optional), file or command line, and the regular expression to be used as input validation.

Mkdir Legacy Code exposed as a Grid Service

Folder : /../.gemlca/legacycodes/mkdir

Content : i) mkdir binary or link ii) config.xml

Legacy Code Interface Description File: config.xml

```
<?xml version="1.0"?>
<!DOCTYPE GLCEnvironment "gemlcaconfig.dtd">
<GLCEnvironment
      id="mkdir" executable="LINUX/mkdir" jobManager="Fork"
      maximumJob="11" minimumProcessors="1"
      maximumProcessors="1" universe="none"
>
<Description>Unix mkdir program</Description>
 <GLCParameters>
    <Parameter  name="-p" friendlyName="Folder to be created"
            fixed="No" inputOutput="Input" order="0"
            mandatory="No" fileCommandline="Commandline">
        <initialValue> </initialValue>
     </Parameter>
    </GLCParameters>
</GLCEnvironment>
```

Figure 2: LCID file of mkdir

An example LCID file describing the standard Unix *mkdir* function as a legacy code is shown on Figure 2.

4 The GEMLCA Architecture

GEMLCA builds on the Open Grid Services Architecture [12] specification, and represents an additional layer on top of any OGSA compatible Grid middleware, such as Globus Toolkit versions 3 (GT3) and 4 (GT4) [2], or g-Lite [1]. In order to access a legacy code program, the GEMLCA Grid client creates a legacy code process, sets its parameters, and uploads its input files. Following this, the legacy code can be submitted as a job to the Compute Server through services offered by the underlying Grid infrastructure and using a particular job manager such as Condor, Fork, PBS or Sun Grid Engine. A more detailed explanation of this will follow in chapter 5.

The GEMLCA Resource layer is a set of Grid services that interacts with the underlying Grid Host Environment layer in order to:

- deploy a new legacy code application as a Grid service,
- query the GEMLCA resources and get the list of available legacy applications,
- get the list of legacy parameters with default values and allow the user to modify these,
- submit legacy jobs to a job manager,
- query the status of previously submitted legacy jobs,
- retrieve results from legacy applications,
- destroy transient Grid service instances, and free the multi-user/instance environment when required,
- offer single sign-on and utilise authentication, authorisation and delegation capabilities.

The GEMLCA Grid services, as every Grid service, are described in a WSDL file and communicate with the help of SOAP messages. This way a GEMLCA Grid service is capable to interact with other standard Grid services. However, the actual WSDL description of the service may be different in different GEMLCA implementations: for example in a GT3 version it is OGSI compatible, in GT4 it is based on WSRF. If a third party Grid service needs access to the legacy code functionalities it can contact the legacy codes through the GEMLCA Grid services.

The GEMLCA functionalities are expressed with the help of the following GEMLCA Grid services:

GLCAdmin: deploys an application as a Grid service with GEMLCA.

GLCList: retrieves the list of available legacy code applications with default parameters.

GLCProcess: submits a legacy application using GEMLCA and gets job status and results back.

In order to deploy a new legacy application as a Grid service, the client has to utilise the *GLCAdmin* GEMLCA service. After authentication, *GLCAdmin* enables the client to modify the XML-based Legacy Code Interface Description (LCID) file of an already deployed legacy code, or to create a new LCID file and upload it to the GEMLCA Resource. As it was explained in chapter 3, this is the only effort a user has to make in order to deploy a new legacy code with GEMLCA.

GLCList returns a list of already deployed legacy codes with their default input parameters using the information stored in the LCID file.

GLCProcess submits legacy code applications to the Compute Servers. The full life-cycle of this Grid service, shown in Figure 3:

1. The user signs the appropriate security certificates in order to create a Grid user proxy.
2. A Grid client creates a Grid Legacy Code Process (*GLCProcess*) instance where the initial environment is set using the GEMLCA file structure. A process environment dynamically supports the legacy code preparation in order to be submitted as a job. The Grid user credential, created in Step 1, is delegated by the *GLCProcess* from the client to the underlying Grid Host Environment for the allocation of resources. For example, in case of a Globus-based implementation the resource allocation is the task of the Master Managed Job Factory Service (MMJFS).

3. The Grid client sets and uploads the input parameters needed by the legacy code program exposed by the *GLCProcess*, deploys a job using the resource specification format of the Grid middleware (a Resource Specification Language (RSL) file in case of Globus), and creates a multi-user environment to handle input and output data.

4. If the client credential is successfully mapped, the Grid middleware contacts the appropriate job manager (Condor, Fork, PBS etc.) that allocates resources and executes the parallel or sequential legacy code on the Compute Servers.

5. As long as the client credentials have not expired and the *GLCProcess* is still alive, the client can contact GEMLCA for checking job status and retrieving partial or final results at any time.

Finally, when the Grid Service instance is destroyed, the multi-user environment is cleaned up.

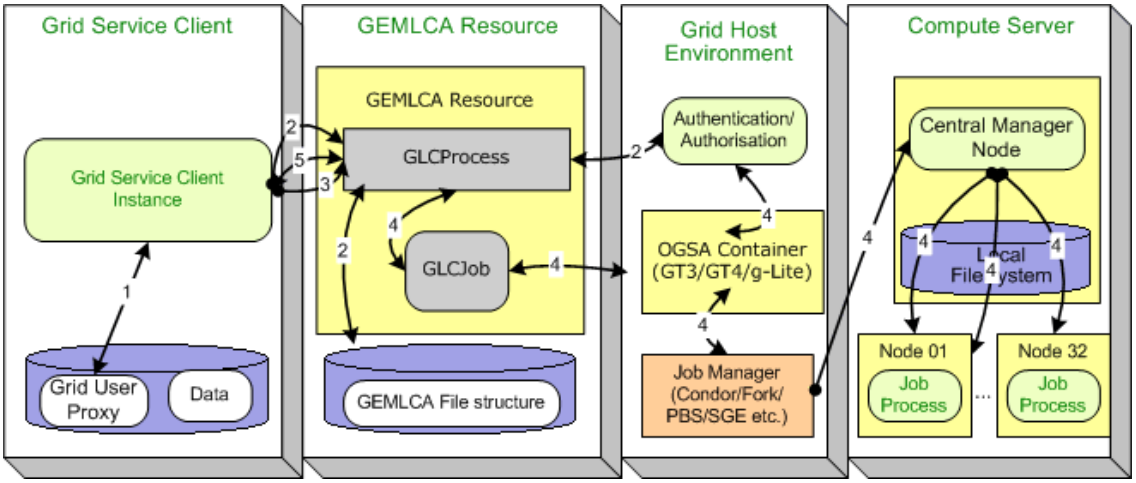


Figure 3: Lifecycle of legacy code service invocation

5 GEMLCA Design, Implementation and Deployment

The design of GEMLCA has taken into consideration a number of objectives for use, maintenance and administration. These objectives include:

- Support the easy deployment of legacy code programs exposed as Grid services. To achieve this, the interfaces provided to the Grid client have to cover the full life-cycle of legacy code deployment, execution and administration.

- Minimise the time and effort required to upgrade and migrate GEMLCA onto new platforms. Given the current dynamic evolution of Grid systems and middleware solutions this requirement is particularly crucial.

- Support Grid system administrators by providing a flexible architecture that can be easily deployed on several sites with minimum effort.

To meet these objectives, GEMLCA has been internally designed in three layers (Figure 4). Each of these layers simulates an encapsulated black-box that is committed to deliver a well-defined functionality to the layer above that, independently of the underlying Grid middleware solution.

The first, *Front End Layer*, offers a set of functionalities as Grid Services which are compatible with the middleware where the system is deployed. Any authorised Grid client can utilise the functionalities to deploy and use legacy code programs on a GEMLCA Resource. The most important functionalities of the Front End Layer are: allowing access to the list and input parameters of available and authorised legacy code programs; executing the required legacy code; retrieving its on-line status; and finally getting the result back from the executed code. A set of interfaces that allows

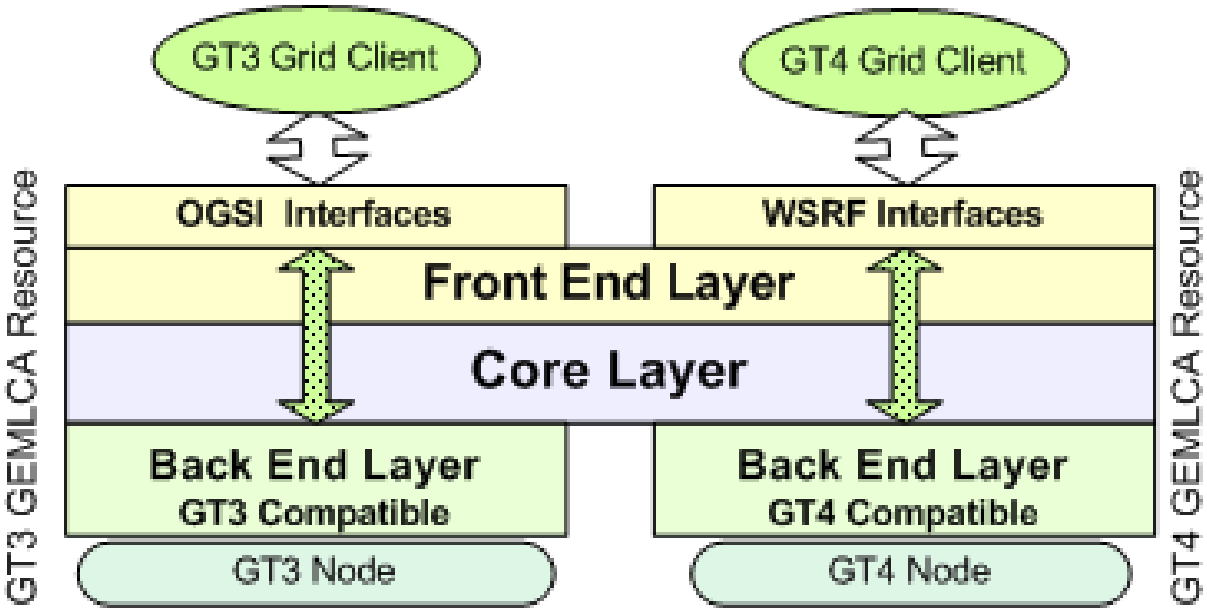


Figure 4: GEMLCA internal design

the deployment and modification of already deployed legacy code programs are also available to assist the GEMLCA administration.

In order to facilitate the submission and re-submission of legacy code programs, a distinction is made between legacy code process and legacy code job. A legacy code process is an instance of a legacy code that is about to be submitted. After a Grid client creates a legacy code process, the input parameters and files can be changed and uploaded to the process environment. When a legacy code process is submitted, a legacy code job is created with its own environment. Using this architecture, several jobs can be submitted and destroyed from a single process. Additionally, it allows multiple submissions in a multi-user environment, where user-specific information, input and output files and parameters, can be preserved separately from other instances running on the same node. The second, *Core Layer* is in charge of the above described administration for each legacy code process and job.

The final *Back End Layer*, is connected to the Grid middleware on the host where the architecture is being deployed. It can be viewed as a layer plug-in that knows the different ways to contact, submit jobs, and get status back from the correlated middleware.

GEMLCA has been implemented using one Java package per layer, each composed of several Java classes. To give a general idea of the internal composition, a stripped-down class diagram of the architecture is shown in Figure 5. In this figure, only the main classes have been represented and linked, and only the public methods of the *Frond End Layer* and *Back End Layer* classes are listed.

From the implementation point of view, to date two currently available Grid middleware are supported by GEMLCA: GT3 and GT4. The relatively easy implementation based on these middleware justified the design principles described in this chapter. Both implementations have broadly the same class design. Some differences of implementation occur in the *Back End Layer*, where the submission classes are, and in some parts of the *Front End Layer*, to expose the difference between OGSI and WSRF compliant Grid Services.

The installation of the GEMLCA Resource is supported by a Unix install script that automatically deploys the architecture into a Tomcat container on the compute node where the Grid Host Environment is located. GEMLCA could be automatically configured using the general parameters of the architecture. At installation, several questions are asked in order to define the GEMLCA Resource behaviour. All these parameters can be changed at any time after the installation. The main configuration parameters are:

Job Managers. Using this parameter, the list of available job managers in a given installation can be defined, together with the job manager URL. The current implementation of GEMLCA supports the Condor and Fork job managers, but with reasonable effort this list can be extended to other job managers supported by any given Grid middleware.

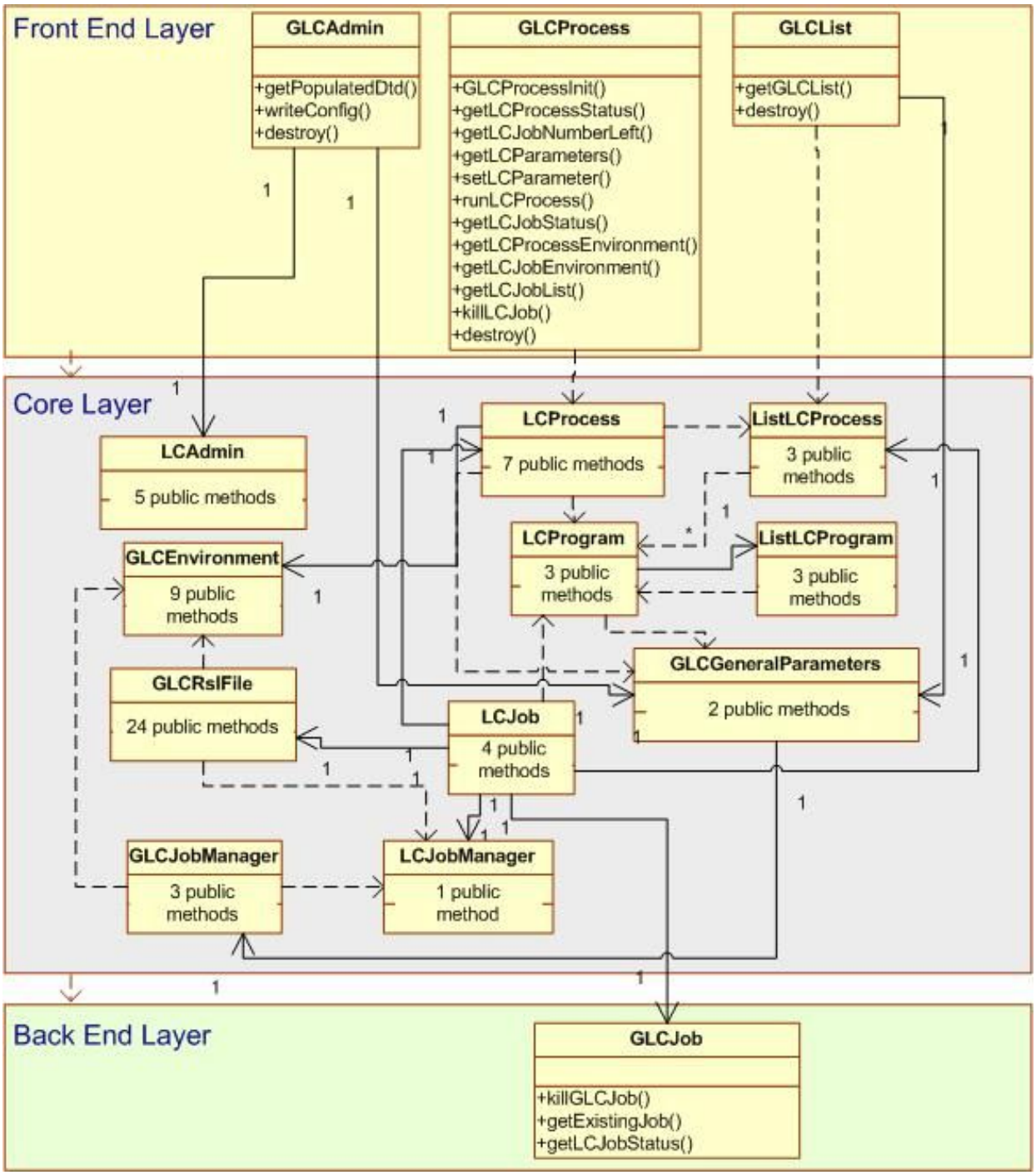


Figure 5: GEMLCA UML class diagram

Locations. There are several folders used by a GEMLCA Resource. The general parameters carry information concerning the location of different types (private or public) of legacy codes, the GEMLCA extended configuration files, and the created process and job environments.

Tuning and reliability. GEMLCA tries to catch and solve Grid middleware problems in order to provide a better consistency to the Grid client. For example, if a job submission fails or the communication with a middleware is not successful, GEMLCA will try to solve this problem rather than expose it back to the client. The GEMLCA behaviour in these cases can be configured using these parameters.

6 GEMLCA Security

GEMLCA legacy code applications should be run as Grid services allowing only authenticated and authorised users to invoke the Grid services. The GT3/GT4 based GEMLCA implementations use the Globus Grid Security Infrastructure (GSI) [26], [14] to enable authentication, implement authorization, support request delegation, and provide data integrity and privacy.

In GEMLCA the server-side security is defined by security descriptors, such as the deployment descriptor, the security configuration file, etc. To achieve client-side security the client code should set the required security modes in the Grid service stubs using the *setProperty() method*.

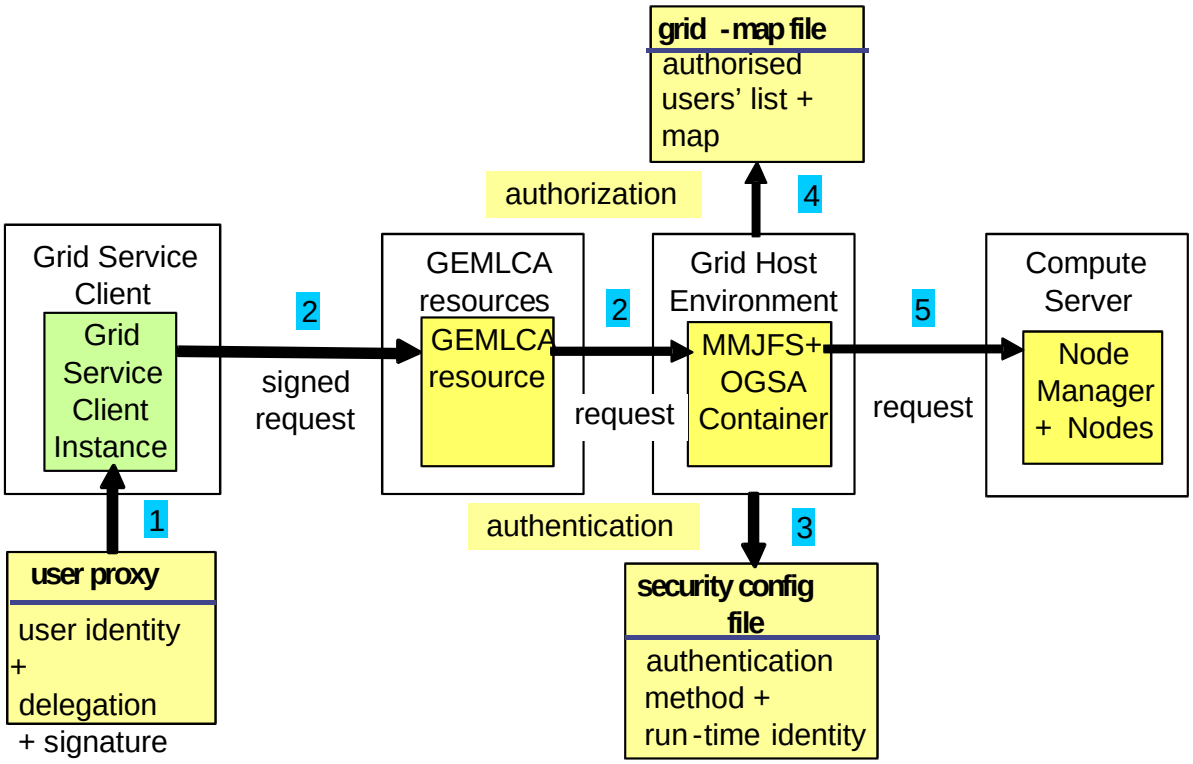


Figure 6: GEMLCA security model

Figure 6 presents the GEMLCA security model, while Table 1 contains the client- and server-side security settings. The GEMLCA security incorporates the following steps:

1. Users should either create or get Grid user proxies, which contain the users certificates with their signatures. (All these activities are hidden and automatically performed by the Certificate Management portlet of the integrated GEMLCA/P-GRADE portal when the user downloads her certificate proxy from the Proxy Server.) The proxies make user identities available in service requests. The user proxy tells the Grid service that the user allows her identity to be used to invoke other Grid services on her behalf. As a result, the Grid service is able to delegate the users credentials when contacting other Grid services.

2. In order to authenticate themselves, users have to add their Grid proxies to service requests. The user creates a service request describing the job to be executed, signs it with her proxy and sends the request to a GEMLCA resource.

3. At the server-side, Grid proxies authenticate users. When a Grid service receives a service request the Grid Host Environment verifies the proxy to authenticate the user.

4. The Grid Host Environment checks whether the user is authorised to access the service using the grid-map file.

5. If the user has the required authorisation and the credential delegation mode is set, the job request is forwarded to the MMJFS on behalf of the user assuming its identity. The MMJFS submits the job to the Compute Server, for example to a Condor cluster, where it is executed on behalf of the user.

client-side security		server-side security	
authentication mode	GSI Secure Conversation with signature	*authorization mode*	Grid user proxy
authorization mode	service identity	*authorization mode*	grid-map
credential mode	full delegation	*credential mode*	full delegation

Table 1: GEMLCA client- and server-side security settings

GEMLCA extends the Globus authorisation mechanisms by providing two security deployment policies for legacy codes. These policies are called *public* and *private*. The *public* policy allows the owner of the GEMLCA resource (system administrator) to deploy new legacy codes which can be made available to any users authorised in the grid-map file. The *private* policy allows other users, who are not the owner of the GEMLCA Resource, to deploy their own legacy codes within their local user repository. As a result, only Grid users who deployed private GEMLCA services are able to access them.

7 Grid Portal as the GEMLCA User Interface

GEMLCA provides the capability to convert legacy codes into Grid services just by describing the legacy parameters and environment values in an XML-based Legacy Code Interface Description file. However, an end-user without specialist computing skills still requires a user-friendly Web interface (portal) to access the GEMLCA functionalities: to deploy, execute and retrieve results from legacy applications. Instead of developing a new custom Grid portal, GEMLCA was integrated with the workflow-oriented P-GRADE Grid portal [20] extending its functionalities with new portlets [18]. In this chapter the features of the integrated GEMLCA/P-GRADE portal are introduced.

The P-GRADE portal has the following advantages compared to other portals:

1. It is built on GridSphere [21] and hence can be easily extended with new portlets.

2. It supports the creation and execution of component-based workflow applications.

3. It provides Grid certificate proxy management for the secure execution of workflows in the Grid.

4. It enables on-line monitoring and visualization of workflow execution.

Integrating the P-GRADE portal with GEMLCA required several modifications in the P-GRADE portal. These are as follows:

1. In the original P-GRADE portal a workflow component can be a sequential or MPI program. The portal was modified in order to include legacy code Grid service components as GEMLCA components (see Figure 9).

2. The *Job properties window* of the P-GRADE portal was changed in order to extend it with the necessary legacy code support (see Figure 9). The *Resources pull down list* contains all those GEMLCA Grid resources to which the user has access. The user can select a GEMLCA Grid resource from this list. Once the Grid resource is selected the portal retrieves the list of legacy code services available on the selected Grid resource. Next, the user can choose a legacy code service from this list. Once the legacy code service is selected the portal fetches the parameter list belonging to the selected legacy code service with default parameter values. The user can either keep these values or modify them.

3. The P-GRADE portal was extended with the GEMLCA Administration Portlet.

As previously mentioned, the deployment of a legacy code Grid service can be done either by the Compute Server administrator or by end-users. The XML definition of an LCID file would not cause any problem for a Compute Server administrator since they are typically well trained IT experts. However, it is expected that most end-users of a Grid system are not IT experts; it is unlikely they would understand XML syntax, and unreasonable to expect them to learn. To support these end-users to deploy their own legacy code Grid services, the GEMLCA Administration portlet was created within the GEMLCA portal. This hides the syntax and structure of the LCID file from users so that users do not have to know LCID specific details, and do not have to be familiar with possible modifications in legacy code description whenever a new GEMLCA release would require it. Figure 7 shows the administration portlet interface of the GEMLCA portal for the *mkdir* legacy code. The user has to specify exactly the same parameters as in the XML file (see Figure 2) but this time using a simple Web form. The LCID file is created automatically and uploaded by the portal to the appropriate directory of the GEMLCA resource.

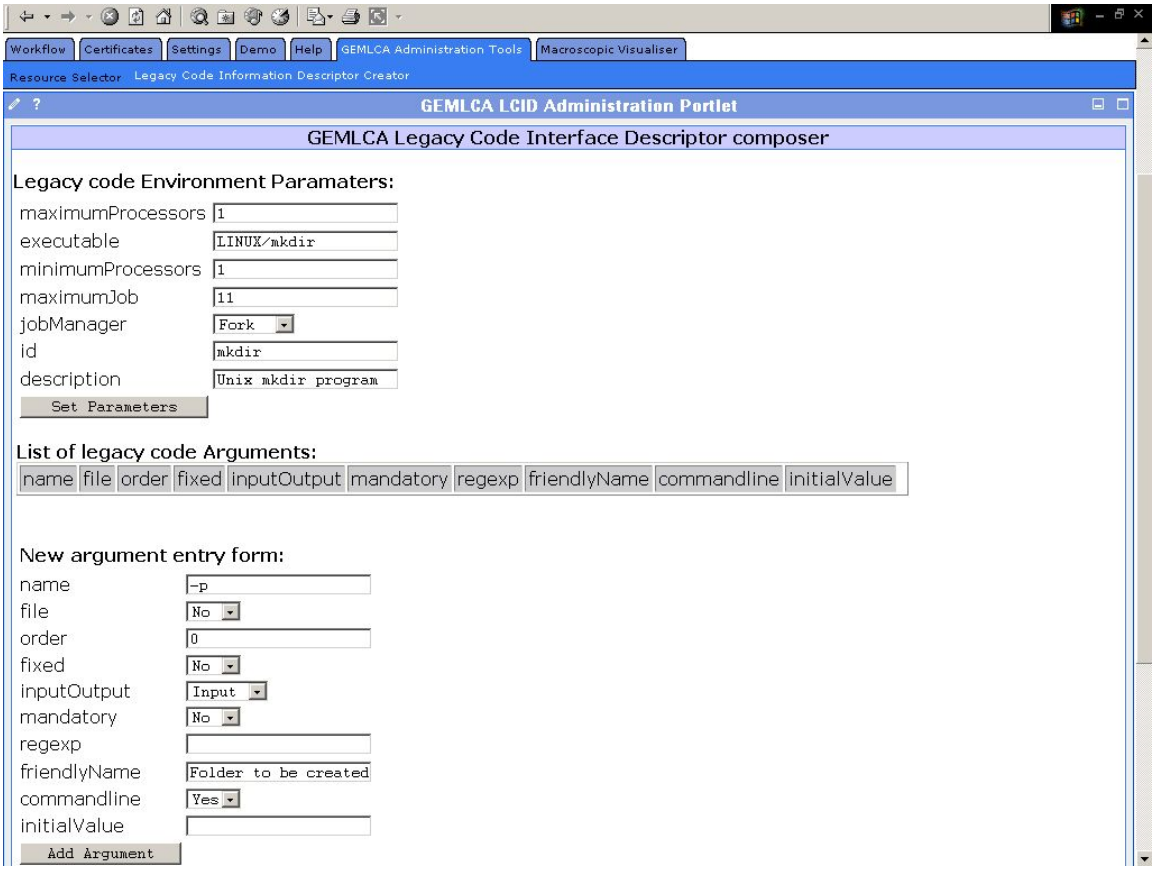


Figure 7: Portal interface to generate the LCID file of ”mkdir”

After these modifications in the portal, end-users can easily construct workflow applications built from legacy code services running on different GEMLCA Grid resources. The workflow manager of the portal contacts the selected GEMLCA Resources, passes them the actual parameter values of the legacy code, and then it is the task of the GEMLCA Resource to execute the legacy code with the actual parameter values. The other important task of the GEMLCA Resource is to deliver the results of the legacy code service back to the portal. The overall structure of the GEMLCA Grid with the Grid portal is shown in Figure 8.

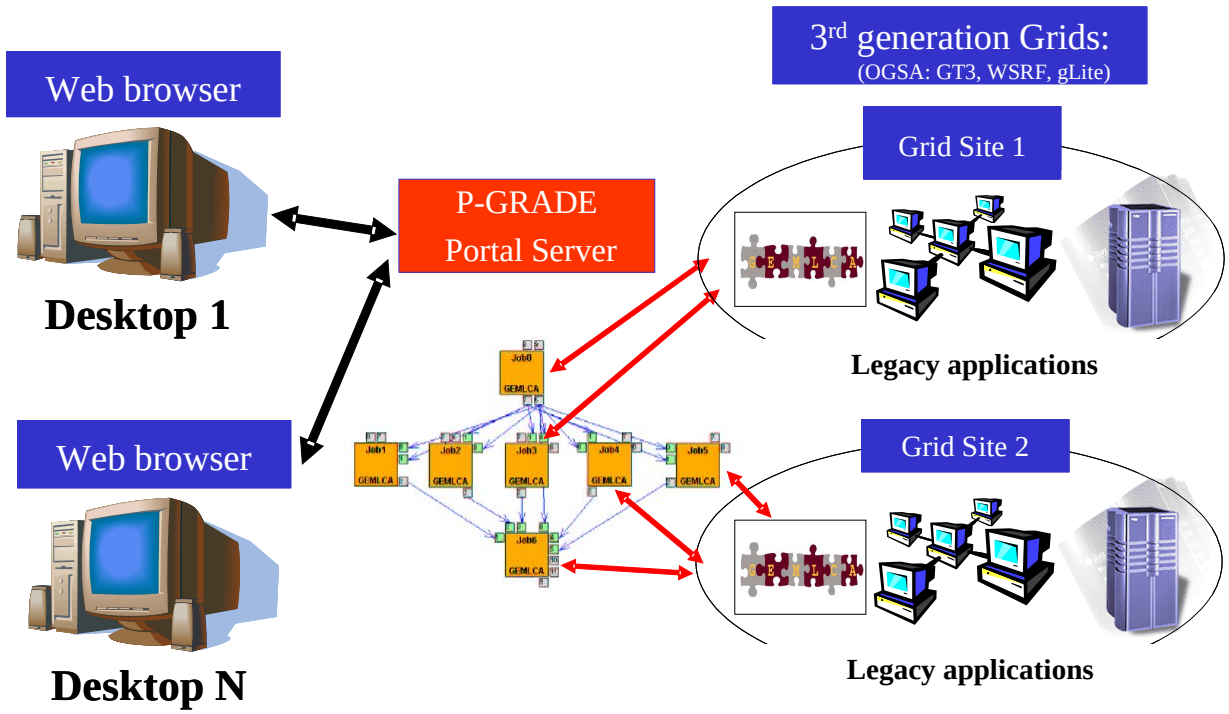


Figure 8: Structure of the GEMLCA Grid

8 Application example

The GEMLCA concept and its GT3-based implementation has been tested and proven by deploying several legacy applications as Grid services, including GAMESS-UK [7], an ab initio molecular electronic structure program for performing SCF-, DFT-, and MCSCF-gradient calculations developed by CCLRC Daresbury Research Laboratory, and MultiBayes [22], a phylogenetic tree construction application developed at the School of Animal and Microbial Sciences at University of Reading. In this paper a third group of legacy applications and their deployment will be described that enables the construction of a complex workflow in order to simulate and compare traffic densities on a road network.

To construct the traffic simulation workflow, three different applications, each of them developed at Centre for Parallel Computing at University of Westminster, were individually deployed as GEMLCA Grid services. The binary executables were copied to different compute nodes of the UK e-Science OGSA Testbed (including Universities of Portsmouth, Reading and Westminter, CCLRC Daresbury Laboratories and SZTAKI Hungary), and the corresponding LCID files were created using the GEMLCA administration interface in the P-GRADE Portal. Access to the source codes or any modification of the programs was not required. The aim of the workflow in this case was to create a parameter study of a legacy traffic simulator by running several instances of the application parallel on different sites and analysing the results on a graph.

The workflow consists of the following legacy code components:

- The *Manhattan legacy code* is an application to generate inputs for the MadCity simulator: a road network file and a turn file. The MadCity road network file is a sequence of numbers, representing a road topology of a road network. The number of columns, rows, unit width and unit height can be set as input parameters to this component. The MadCity turn file describes the junction manoeuvres available in a given road network. Traffic light details are also included in this file.

- *MadCity* [15] is a discrete-time microscopic *traffic simulator* that simulates traffic on a road network at the level of individual vehicles behaviour on roads and at junctions. The simulator models the movement of vehicles using the road network and turn file as inputs. After completing the simulation, a macroscopic trace file, representing

the total dynamic behaviour of vehicles throughout the simulation run, is created. MadCity has been parallelised using the pipeline template of P-GRADE (Parallel Grid Run-time and Application Development Environment) [17].

- Finally a *traffic density analyser* compares the traffic congestion of several runs of the simulator on a given network, with different initial road traffic conditions specified as input parameters. The component presents the results of the analysis graphically.

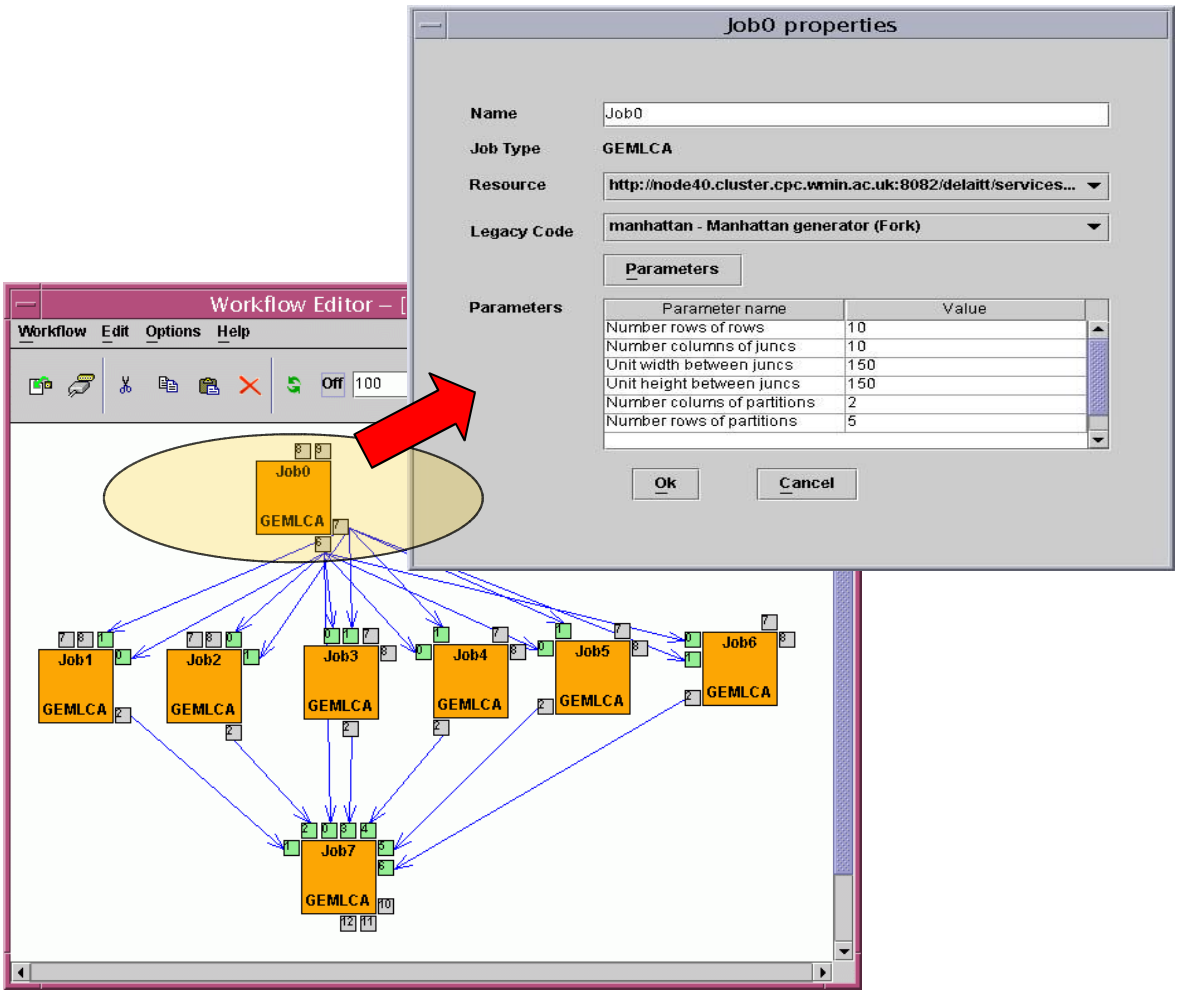


Figure 9: Workflow construction and Job properties window in the integrated GEMLCA/P-GRADE portal

The workflow combining the above described components is shown on Figure 9. Job 0 is the Manhattan Road Network Generator; jobs 1 to 6 are traffic simulators running on different sites with different input parameters; and job 7 is the analyser. When creating the workflow, the properties of each component have to be set, as illustrated in the figure for the road network generator (Job 0). After selecting the GEMLCA resource, the *GLCList* service returns the list of previously deployed legacy applications on the selected resource. Following this, the appropriate legacy code, in this case the Manhattan generator, can be selected, the default parameters retrieved by clicking the parameters button, and changed if required.

The workflow was successfully created and executed on the OGSA testbed sites. A sample output graph illustrating the change of traffic density with respect to time on road 961 of the network, for 6 different initial conditions (the initial number of cars on the junction), is shown in Figure 10.

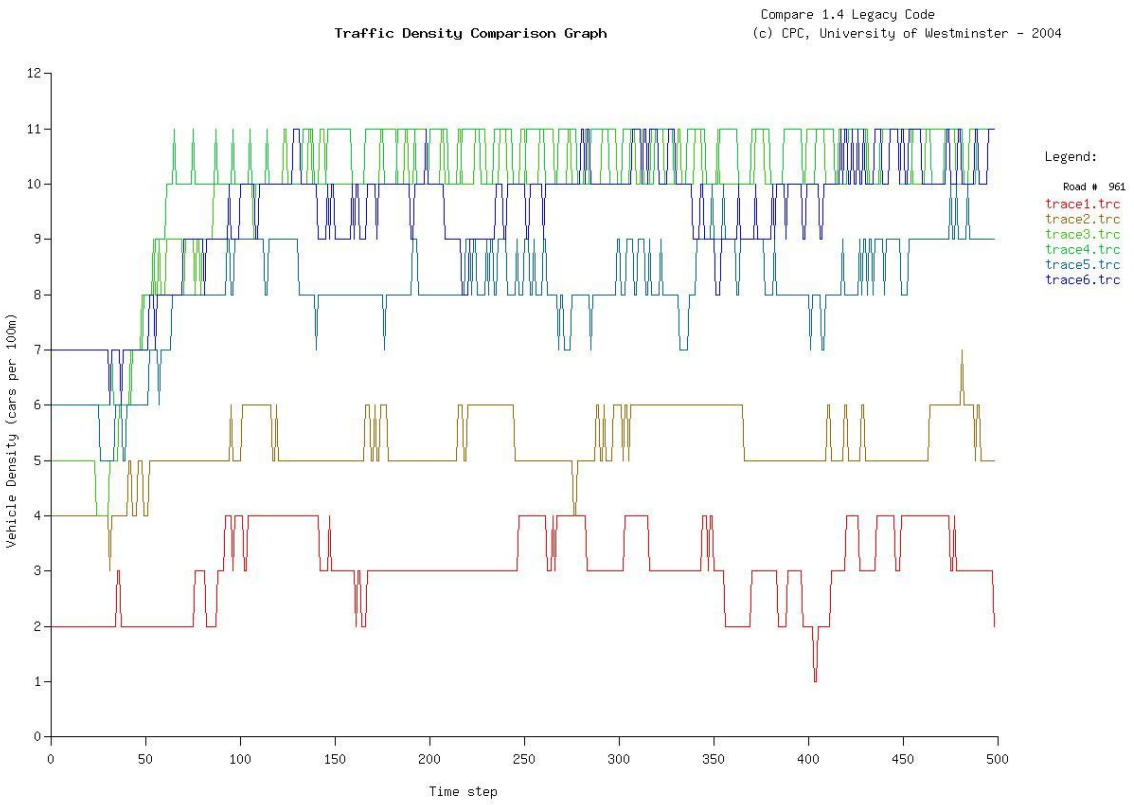


Figure 10: Traffic Density on Road 961 Depending on Initial Number of Cars

9 Future work

After releasing the first GT3-based GEMLCA implementation, the research team reviewed its architecture, features and services. The current GEMLCA version provides support for converting legacy code application to Grid services but does not offer automatic deployment, brokering and performance management service. As a result, users have to deploy Grid services manually. They also have to discover, select and schedule Grid services. Taking into account these limitations, the team has defined the concept of the next GEMLCA releases to support multiple domain Grids with different Grid middleware, minimize user involvement and be user-friendly. Taking into considerations these requirements the team has identified the following improvements.

Migration to other Grid middleware: GEMLCA was defined as a three-layer architecture to minimise efforts required to migrate it to different Grid middleware. As a result, migration of GEMLCA to any other OGSA-based Grid middleware requires a slight modification of the Front End Layer and complete re-writing of the Back End Layer. The GT3-based GEMLCA implementation was completed in June 2004 and it was released in January 2005. The GT4-based GEMLCA implementation was finished by the end of February 2005. Currently, the team is investigating how to migrate GEMLCA to g-Lite [1].

Service-oriented Grid info system: In the next GEMLCA releases, the Grid information system should contain and provide data on available GEMLCA resources and services. Having available a Grid information system [9], [27], users should specify the GEMLCA services they want to invoke and the Grid broker should query the Grid information system to obtain a list of available and suitable services for select and scheduling. Current information systems, such as Globus MDS3 and MDS4 use either attribute-based or name-based look-up methods to find available services. To improve efficiency of service discovery, semantic-based service discovery should be used.

Grid Service Brokering: The current GEMLCA version should be extended with a service-oriented Grid broker, which can discover, select and schedule GEMLCA services to achieve the expected performance of Grid applications. The Grid broker has to make selection and scheduling decisions based on QoS parameters such as availability, cost, performance, response time, security etc. The QoS requirements [4], [11] should be defined by Service Level Agreements (SLA) [24] to guarantee the service delivery. The research team is investigating the performance parameters and requirements of Grid QoS, methods to generate, store and retrieve QoS data of Grid services, and approaches to negotiation and management of SLAs.

Interoperability problems in GEMLCA: In the Grid it is expected that Grid services may be deployed onto different hosting environments, allocated to diverse service domains, follow multiple policy models and enforce different security mechanisms. These multi-dimensional heterogeneities raise interoperability issues in both Grid service deployment and invocation. Two scenarios illustrate the interoperability problems. In the first, a Grid service is copied from one Grid domain and deployed onto a new one. In the second, service requests are submitted to instances of a Grid service deployed in different Grid domains. To solve these interoperability problems the GEMLCA team has to investigate interoperability issues in four main areas: information services, data, resource and security management.. Having GEMLCA adapted to different Grid environments enables the construction of complex workflows executed by Grid services that can be deployed, managed and accessed across different Grid environments in a seamless and transparent manner.

Automatic deployment: GEMLCA services can have multiple instances able to run on multiple computing nodes having well-defined Grid service interfaces. In the Grid it may happen that the deployed GEMLCA services are not available to execute new service requests. To avoid this situation GEMLCA services should be deployed automatically on new computing nodes and be made available on demand. GEMLCA should be extended with an automatic deployment service [23], [25] incorporating the following functionalities: legacy code and computing node dependency detection, discovering and selecting computing nodes for deployment based on legacy code and computing node dependency descriptions and automatic deployment.

Fine-grained access control of legacy codes: GEMLCA currently provides coarse-grained access control to legacy codes, based on grid-map file built authorisation, when publishing or accessing them. It is envisaged that the implementation should be extended with authorisation frameworks like PERMIS in order to provide fine-grained access control of legacy codes for existing public and private security policies. This will allow the end user to deploy a new legacy code as private, and select those users allowed access.

10 Conclusions

GEMLCA was created to make existing legacy applications available as Grid Services. It provides a new approach to deploying legacy code applications as Grid services, by avoiding the need to modify, or access, the source code. GEMLCA adds a software layer to existing Grid middleware such Globus GT3 and GT4, and supports an integrated Grid execution lifecycle environment for multiple users. As a layered architecture it offers a set of OGSA-compliant interfaces to create, run and manage Grid services that offer all the legacy code application functionality.

GEMLCA was integrated with P-GRADE Grid portal and workflow to provide a graphical user-friendly development and execution environment. The user only has to create an XML-based Legacy Code Interface Description File to describe the legacy parameters and environment values using a Grid portlet to convert legacy code applications into Grid services and GEMLCA enables them to be run from a Grid service client. The workflow editor of the P-GRADE portal enables the connection of legacy code applications into complex workflows using a Web-based graphical environment and the workflow manager supervises their execution.

Currently, two GEM LCA implementations are available: the first is built on GT3 and the second on GT4. There is on-going work to migrate GEMLCA to other Grid middleware, for example to g-Lite.

GEMLCA is recommended for users who do not have extensive technical expertise and skills in Grid computing, but who do have legacy code applications they wish to deploy and use as Grid services. There is a GEMLCA installation package, which helps to install and set GEMLCA. Following installation, the Grid portal offers full support for users through a number of portlets. As a result, to learn and use GEMLCA is quite easy.

References

[1] The g-lite website. http://glite.web.cern.ch/glite/.

[2] The globus alliance website. http://www.globus.org/.

[3] R. Al-Ali, K. Amin, and G. von Laszewski. An ogsa-based quality of service framework. In *2nd International Workshop on Grid and Cooperative Computing (GCC2003)*, December 2003. Shanghai, China.

[4] R. Al-Ali, A. Hafid, O. F. Rama, and D. W. Walker. Qos adaptation in service-oriented grids. In *Proceedings of the 1st International Workshop on Middleware for Grid Computing (MGC2003) at ACM/IFIP/USENIX Middleware 2003*, June 2003. Rio de Janeiro, Brazil.

[5] B. Balis, M. Bubak, and M. Wegiel. A solution for adapting legacy code as web services. In V. Getov and T. Kiellmann, editors, *Component Models and Systems for Grid Applications*, pages 57–75. Springer, 2005. ISBN 0-387-23351-2.

[6] T. Bodhuin and M. Tortorella. Using grid technologies for web-enabling legacy systems. In *Proceedings of the Software Technology and Engineering Practice (STEP), Software Analysis and Maintenance: Practices, Tools, Interoperability workshop September 19-21, 2003, Amsterdam, The Netherlands.*, 2003.

[7] CCLRC. Gamess-uk. http://www.cse.clrc.ac.uk/qcg/gamess-uk/.

[8] University of Wisconsin-Madison Condor Team. Condor version 6.4.7 manual. http://www.cs.wisc.edu/condor/manual/v6.4/.

[9] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing. In *10th IEEE International Symposium on High-Performance Distributed Computing (HPDC-10)*. IEEE Press, 2001.

[10] T. Delaitre, A.Goyeneche, T.Kiss, and S.C. Winter. Publishing and executing parallel legacy code using an ogsi grid service. In *Conf. Proc. of the 2004 International Conference on Computational Science and its Applications, Technical Session on Grid Computing, ISBN 3-540-22056-9*, pages 30–36, May 2004. Assisi, Italy.

[11] C. Dumitrescu and I. Foster. Gruber: A grid resource sla broker. Technical report, GriPhyN/iVDGL, 2005.

[12] I. Foster, C. Keselman, J. Nick, and S. Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration, 2002. http://www.globus.org/research/papers/anatomy.pdf.

[13] D. Gannon, S. Krishnan, A. Slominski, G. Kandaswamy, and L. Fang. Building applications from a web service based component architecture. In V. Getov and T. Kiellmann, editors, *Component Models and Systems for Grid Applications*, pages 3–17. Springer, 2005. ISBN 0-387-23351-2.

[14] J. Gawor, S. Meder, F. Siebenlist, and Von Welch. Gt3 grid security infrastructure overview. Draft March, 17, 2005.

[15] A. Gourgoulis, G. Terstyansky, P. Kacsuk, and S.C. Winter. Creating scalable traffic simulation on clusters. In *PDP2004. Conference Proceedings of the 12th Euromicro Conference on Parallel, Distributed and Network based Processing*, February 2004. La Coruna, Spain.

[16] Y. Huang, I. Taylor, and D. W. Walker. Wrapping legacy codes for grid-based applications. In *Proceedings of the 17th International Parallel and Distributed Processing Symposium, workshop on Java for HPC), 22-26 April 2003, Nice, France. ISBN 0-7695-1926-1*, 2003.

[17] P. Kacsuk, G. Dozsa, J. Kovacs, R. Lovas, N. Podhorszky, Z. Balaton, and G. Gombas. P-grade: A grid programing environment. *Journal of Grid Computing*, 1(2):171–197, 2004.

[18] P. Kacsuk, A. Goyeneche, T. Delaitre, T. Kiss, Z. Farkas, and T. Boczko. High-level grid application environment to use legacy codes as ogsa grid services. In *Conf. Proc. of the 5th IEEE/ACM International Workshop on Grid Computing*, pages 428–435, November 2004. Pittsburgh, USA.

[19] D. Kuebler and W. Eibach. Adapting legacy applications as web services. IBM Developer Works, http://www-106.ibm.com/developerworks/webservices/.

[20] Cs. Nemeth, G. Dozsa, R. Lovas, and P. Kacsuk. The p-grade grid portal. In *Computational Science and Its Applications - ICCSA 2004: International Conference, ISBN 3-540-22056-9*, pages 10–19, 2004. Assisi, Italy.

[21] J. Novotny, M. Russell, and "An Advanced Portal Framework" O. Wehrens: GridSphere. In *Conf. Proc. of the 30th EUROMICRO Conference*, 2004. Rennes, France.

[22] University of Reading. Multibayes. http://www.ams.rdg.ac.uk/.

[23] Ai Ting, Wang Caixia, and Xie Yong. *Dynamic Grid Service Deployment*, 2004. http://www.comp.nus.edu.sg/ wangxb/SMA5505-2004/xieyong-report1.pdf.

[24] S. Venugopal, R. Buyya, and L. Winton. A grid service broker for scheduling distributed data-oriented applications on global grids. In *2nd International Workshop on Middleware in Grid Computing*, October 2004. Toronto, Ontario, Canada.

[25] Jon B. Weissman, Seonho Kim, and Darin England. *Supporting the Dynamic Grid Service Lifecycle*, 2004. https://wwws.cs.umn.edu/tech_reports_upload/tr2004/04-041.pdf.

[26] V. Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Czajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman, and S. Tuecke. Security for grid services. In *Twelfth International Symposium on High Performance Distributed Computing (HPDC-12)*. IEEE Press, June 2003. http://www.globus.org/Security/GSI3/GT3-Security-HPDC.pdf.

[27] J. Yu, S. Venugopal, and R. Buyya. Grid market directory: A web services-based grid service publication directory. Technical report, Grid Computing and Distributed Systems Lab, University of Melbourne, Jan 2003.