GEMLCA: grid execution management for legacy code architecture design.

Thierry Delaitre
Ariel Goyeneche
Peter Kacsuk
Tamas Kiss
Gabor Terstyanzsky
Stephen Winter

Cavendish School of Computer Science

GEMLCA*: Grid Execution Management for Legacy Code Architecture Design

T. Delaitre, A. Goyeneche, P. Kacsuk, T. Kiss, G.Z.Terstyanszky and S.C. Winter
Centre for Parallel Computing,
Cavendish School of Computer Science,
University of Westminster
115 New Cavendish Street, London W1W 6UW

E-mail: `testbed-discuss@cpc.wmin.ac.uk`

Abstract

The Grid Execution Management for Legacy Code Architecture (GEMLCA) describes a solution for exposing and executing legacy applications through an OGSI Grid Service. This architecture has been introduced in a previous paper by the same authors where the general concept was demonstrated by creating an OGSI/GT3 version of the MadCity traffic simulator. In this paper, the class structure of the architecture is described presenting each component and describing the relationships between them. Also, the current architecture implementation is evaluated through test results gained by running the MadCity traffic simulator as a C/PVM legacy application.

1. Introduction

The computational Grid aims to facilitate flexible, secure and coordinated resource sharing between participants. In a Grid computing environment many different hardware and software resources have to work together seamlessly. These resources can include legacy code programs that were originally implemented to be run on single computers or on computer clusters. The deployment of these programs in a Grid environment can be very difficult and usually require significant re-engineering of the original code.

Most recent approaches to realize the Grid, like OGSI [1] and GT3 [2] as its first implementation, are all based on a service-oriented architecture. One possibility offered by service-orientation is to deploy legacy applications as Grid services without re-engineering the original code. The Grid Execution Management for Legacy Code Architecture [3] was designed to fulfill this requirement. Any legacy application can be offered using GEMLCA through a front-end OGSI Grid service without modifying the original code. The architecture offers a number of interfaces to the Grid client in order to submit computational jobs, to check their status, and to get the results back.

In [3], the authors described the general architecture and demonstrated the concept by creating a GT3 version of the MadCity parallel traffic simulator. This paper details the architecture introducing its three-layer design and explaining deployment files. Also, the current status of implementation is outlined and the results gained by running MadCity through this architecture are presented and analysed. Finally, future development tasks are summarised.

2. Related works to use legacy code in Grid systems

Many large industrial and scientific applications are available today that were written well before Grid computing or service-oriented architectures appeared. To integrate these legacy code programs into service-oriented Grid architectures with the smallest possible effort and best performance, is a crucial point in more widespread industrial take-up of Grid technology.

There are several research efforts aiming at automating the transformation of legacy code into a Grid service. Most of these solutions are based on the general framework to transform legacy applications into Web services outlined in [4], and use Java wrapping in order to generate stubs automatically. One example for this is presented in [5], where the authors describe a semi-automatic conversion of legacy C code into Java using JNI (Java Native Interface). After wrapping the native C application with the JACAW (Java-C Automatic Wrapper) tool MEDLI (MEdiation of Data and Legacy Code Interface) is used for data mapping in order to

*The work presented in this paper is supported by an EPSRC funded project (Grant No.: GR/S77509/01): A proposal to evaluate OGSA/GT3 on a UK multi-site testbed.

make the code available as part of a Grid workflow.

Different approaches from wrapping are presented in [6] and [7] but unfortunately these solutions only describe the principles and do not give a generic tool to do the automatic conversion.

Compared to Java wrapping GEMLCA is based on a different principle. It offers a front-end Grid service layer that communicates with the client in order to pass input and output parameters, and contacts a local job manager through Globus MMJFS [2] (Master Managed Job Factory Service) to submit the legacy computational job. To deploy a legacy application as a Grid service there is no need for the source code and not even for the C header files as in case of JACAW. The user only has to describe the legacy parameters in a pre-defined XML format. The legacy code can be written in any programming languages and can be not only a sequential but also a parallel PVM or MPI code that uses a job manager like Condor [8] and where wrapping can be difficult. The current implementation of GEMLCA is based on GT3 but the architecture itself is more generic and can be easily adapted to other service-oriented approaches like WSRF or a pure Web services based solution.

Besides substantial advantages offered by GEMLCA it is also important to note that, as most of the other solutions, it supports decomposable or semi-decomposable software systems where the business logic and data model components can be separated from the user interface. The former can then be transformed into a Grid service using GEMLCA, and the latter have to be re-implemented, for example as part of a Grid portal. An approach to deal with non-decomposable legacy programs is described in [6] using screen proxies and redirecting input/output calls. However, this solution is language dependant and requires modification of the original code.

3 Grid Execution Management for Legacy Code: General Architecture

MadCity, as many other legacy code programs, has been developed and implemented to run on a computer cluster and does not offer a set of OGSI complying interfaces in order to be published and made available as a Grid Service. One approach to achieve this is to re-engineer the legacy code, which in many cases implies significant efforts. Another solution to make available existing parallel legacy code programs as OGSI Grid services without having to re-engineer them is supported by the Grid Execution Management for Legacy Code Architecture (GEMLCA) proposed in this paper.

GEMLCA can be seen as a client front-end OGSI Grid service layer that offers a number of interfaces to submit and check the status of computational jobs, and get the results back. As any other Grid Service, GEMLCA has an interface described in Web Services Description Language (WSDL) [9] that can be invoked by any Grid services client to bind and use its functionality through Simple Object Access protocol (SOAP) [10].

The general architecture to deploy existing legacy code as a Grid service is presented in this paper based on OGSI and GT3 infrastructure. However, the concept of GEMLCA is more generic and can also be applied to other service-oriented architectures. A similar solution is described in [11] for Jini, where Java RMI was used as communication protocol and the Jini lookup service to connect clients and services. Using different platforms the communication and the actual service implementation is different, but the concept of the architecture remains the same. This way the transition to new emerging standards like Web Services-Resource Framework (WSRF) [12] and GT4, will be straightforward.

As a general introduction, GEMLCA supports the following characteristics:

- Offers a set of OGSI interfaces, described in a WSDL file, in order to create, run and manage Grid service instances that offer all the legacy code program functionality.
- Interacts with job managers, such as Fork, Condor, PBS or Sun Grid Engine, allocates computing resources, manages input and output data and submits the legacy code program as a computational job.
- Administers and manages user data (input and output) related to each legacy code job providing a multi-user and multi-instance Grid service environment.
- Ensures that the execution of the legacy code maps to the respective client Grid credential that requests the code to be executed.
- Presents a reliable file transfer service to upload or download data from the Grid service master node.
- Offers a single sign-on capability for submitting jobs, uploading and downloading data.
- A Grid service client can be off-line waiting for compute jobs to be completed, and can request jobs status information and results any time before the GEMLCA instance termination time expires.
- Reduces complexity for application developers by adding a software layer to existing OGSI services and by supporting an integrated Grid execution life-cycle environment for multiple users/instances. The Grid execution life cycle includes: upload of data, submission of job, check the status of computational jobs, and get the results back.

Figure 1 describes the GEMLCA implementation and its life-cycle. The Condor management system is used by the Westminster cluster as the job manager to execute legacy parallel programs.

The scenario for submitting legacy code using the GEMLCA architecture is composed of the following steps:

- (1) The user signs his certificates to create a Grid proxy for authenticating to Grid services. The user Grid credential will later be delegated by GEMLCA from the client to the Globus Master Managed Job Factory Service (MMJFS) for the allocation of resources.
- (2 & 3) A Grid service client, using the Grid Service Management for Legacy Code Factory, creates a number of Grid service Management for Legacy Code instances giving them a lifetime.
- (4) The Grid Client, using OGSI interfaces, calls a specific instance providing a set of input parameters needed by the legacy code program. The instance creates a Globus Resource Specification Language (RSL) file and a multiuser/instance environment to handle input and output data.
- (5 & 6) The instance, using the RSL file, contact the Globus MMJFS on behalf of the client. The client credential is mapped to a specific local user using the *Grid-mapfile* mechanism.
- (7, 8 & 9) If the client credential is successfully mapped, MMJFS contacts the Condor job manager which allocates resources and executes the parallel legacy code in a computer cluster.
- (10) At any time after the instance is created, the client can contact it in order to check the job status.
- (11) The Grid service instance can notify the client about any job status change using the OGSI notification framework if the client is on-line.
- (12 to 15) If the Grid Service instance's time has not expired and the job status is finished, the Grid client contacts the Reliable File Transfer service instance to send results back to the client.

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

Figure 2 summarises the GEMLCA life-cycle on a sequence diagram.

4 GEMLCA class structure

GEMLCA is a three-layer architecture (Figure 3) that enables any general legacy code program to be deployed as an OGSI Grid Service. The layers can be introduced as:

- The front-end layer called Grid Services Layer is published as a set of Grid Services which is the only access point for a Grid client to submit jobs and retrieve results from a legacy code program. This layer offers the functionality of publishing legacy code programs already deployed on the master node server. A Grid client can create a *GLCProcess* and a number of *GLCJob* per process that are submitted to a job manager. This allows the user extra flexibility by adding the capability of managing several similar applications using the same Grid service process and varying the input parameters.
- The Internal Layer is composed of several classes that manage the legacy code program environment and job behaviour.
- The GT3 backend Layer that is closely related to Globus Toolkit 3 and offers services to the Internal Layer in order to create a Globus Resource Specification Language file (RSL) and to submit and control the job using a specific job manager. This layer essentially extends the classes provided by Globus version 3 offering a standard interface to the Internal Layer. The main inspiration behind this idea is to disconnect the architecture's main core from any third party classes, such as GT3, offering a level of abstraction for future releases, like GT4 and its new WSRF [12] implementation.

5 Description of GEMLCA classes

The *GLCListLegacyCodePrograms* class is one of the front-end layer Grid Services that publishes a list of available legacy code programs that can be used as a Grid Service by the GELMCA architecture. The Legacy Code List File (LCLF) contains a list of programs published by the front-end layer Grid Service. Each LCLF is associated with its respective Legacy Code Definition File (LCDF).

LCDF stores data related to the legacy code program. Among other information, this file describes the folder and program name, the job managers that are able to support this program, information about security and the list of user subject certificates that can be accepted to run this program, and the minimum and maximum number of processors required. Input and output parameters are also listed in this file describing which ones are fixed or can be changed by the Grid client user, and which of them are command line or file providing default values for all the mandatory parameters. The LCDF file and parameters are represented and managed by the *GLCEnvironment* and *GLCParameters* classes.

Using the *GLCListLegacyCodePrograms* Grid Service, a client can retrieve a list of available legacy code pro-

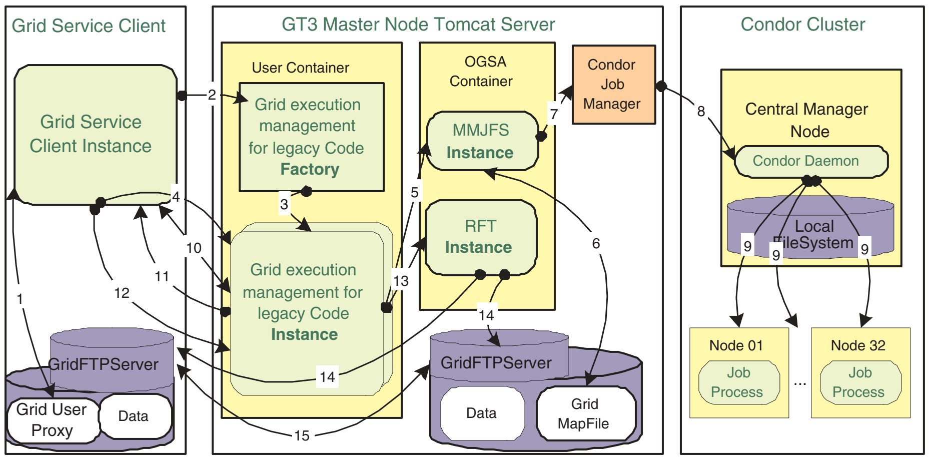

Figure 1. GEMLCA life-cycle.

grams. A client that meets the security requirements can create *GLCProcess* instances invoking another GEMLCA Grid Service, the *GLCFactoryProcess*. The factory uses the LCDF file to create the environment and sets the default program parameters.

A *GLCProcess* object represents a legacy code process in this architecture. This process cannot be submitted to any job manager if the environment and all the mandatory input parameters have not been created. A client Grid service can submit a job using the default parameters or change any non-fixed parameter before submission. Any time a process is submitted, a new *GLCJob* object is created using a different environment. The process LCDF file gives the maximum number of jobs related to one process.

The *GLCJob* uses the process environment and the classes provided by the GT3 layer to create an RSL file that is used to submit the legacy code program to a specific job manager. The *GLCBasicRslFile* object sets the file skeletal description that is completed by the more specific *GLCRslFile* class.

A Grid Service client can check the general process status or a specific jobs behaviour. Also, a client can destroy a complete process or a specific job with in the process.

6 GEMLCA Implementation

The architecture presented in the previous section is implemented by deploying a secure Grid execution management service and tested by developing a secure Grid client. The implementation of the current development focuses on a subset of th e architecture life-cycle and in particular on submitting jobs by the Grid client and executing them on the Westminster Condor pool cluster. The Grid client and service are both implementing GT3 security and in particular GT3 security proxy delegation [13] which allows the caller's Grid credential to be passed from the client via the Grid execution management service to the Master Managed Job Factory Service. MMJFS submits the job to the Condor pool via the Condor job manager and maps the execution of the program to the respective Unix user login name of the requestor's Grid credentials by using the GT3 Grid-mapfile mechanism.

The Grid client is currently a Java program executed by the Java virtual machine from a Unix terminal window. The Grid execution management service and MMJFS are deployed in two separate Java servlet engine containers, and in particular Tomcat web application contexts, hosted by a single Tomcat server running on the Westminster GT3 master node. The reason to configure multiple containers hosted by the same Tomcat server is to ease the administration of a well-maintained GT3 server for Grid services deployments for multiple Grid developers on our GT3 master node. Another advantage is that each Grid developer can restart its own web application context using the Tomcat web application context manager without having to restart the whole Tomcat server.

The architecture presented in the previous section requires a specific job manager such as Condor, PBS or Sun Grid Engine to be configured for submitting computational jobs to clusters. Condor is selected as the job management facility for the Westminster cluster and it requires the Condor job manager interface to be installed and configured as well as the GT3 master node to be configured as a submit host to the Parsifal Condor pool. The default installation of GT3 only installs the Fork job manager and an additional step is required to install and configure Condor which is bundled with GT3. However, an updated file for the Condor job manager needs to be downloaded from the GT3 bugzilla facility (bugid 1425) as the native system condor.pm file contains a software fault which prevents Condor from get-

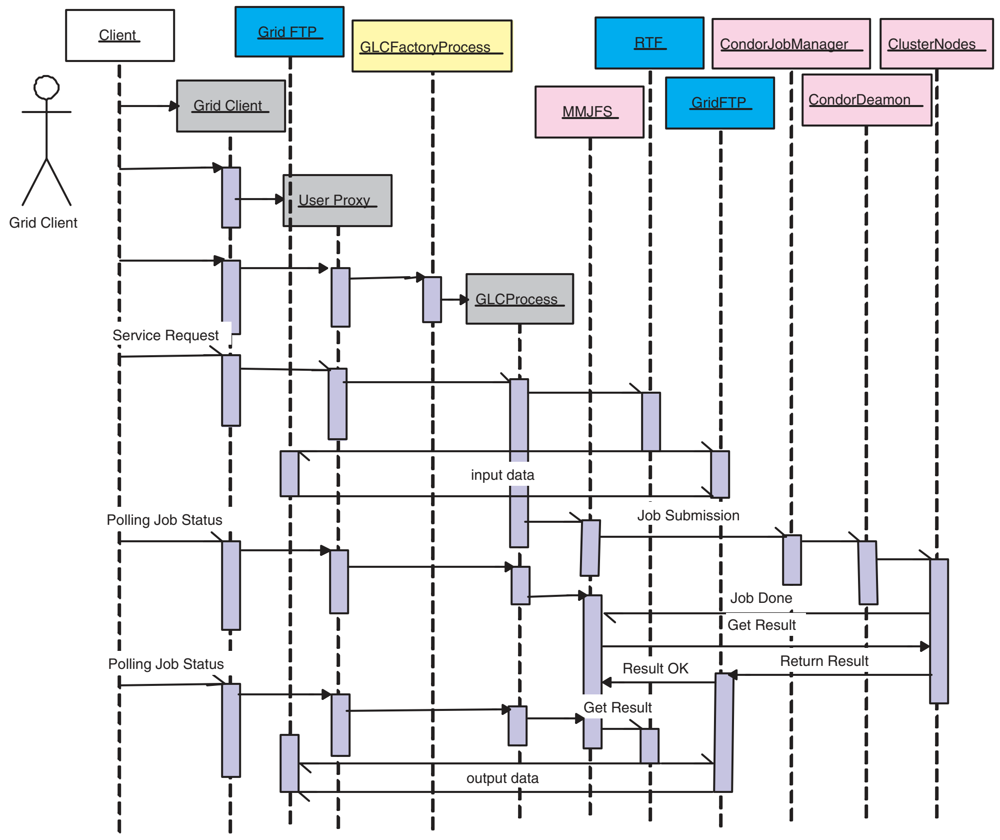

Figure 2. GEMLCA sequence diagram.

ting the correct status of Condor jobs.

In order to test the architecture, the MadCity traffic simulator was deployed as a GEMLCA Grid service. The parallel versions of MadCity are implemented using PGRADE [14, 17] and Spider [15] relying on PVM [16] as the message passing interface. This therefore requires the MadCity computational job to be submitted to the Condor pool using the PVM Universe. However, one of the issues of using Condor with the Globus Toolkit is that there is no native solution to set the PVM Universe in the Globus resource specification language (RSL). The solution to set the PVM Universe and specific Condor execution parameters is to use a Condor hash table in the RSL file to overcome the native RSL limitation.

The execution management Grid service uses the GT3 GramJob API to interface with MMJFS. The GramJob event notification does not work in a Grid service for interactive job submission because the GramJob API has been designed by Globus as a client API. Solutions include the use of a separate thread for the GramJob code, using the native OGSI APIs and extend the NotificationSink portType or use batch mode submission. Our approach is to submit jobs in batch mode to MMJFS which does not require an event listener to notify jobs completion.

7 Results and Future Developments

The general concept of GEMLCA is demonstrated by deploying the MadCity traffic simulator as an OGSI Grid service. The simulator is run through the architecture and test results are analysed.

MadCity [17], a discrete time-based microscopic simulator, was developed by the Centre of Parallel Computing at the University of Westminster. The Simulator is organised around a compound data structure that represents the road network. The network file may contain several thousands of roads and hundreds of junctions. At each step of the simulation, each vehicle uses a set of simple localised rules to compute its new state and position taking into account its surrounding conditions.

The performance results presented in Figure 4 were gained by running a parallel version of the traffic simulator implemented by P-GRADE [14] through GEMLCA. The performance results are in the same range as presented in [17].

Future development includes the implementation of the

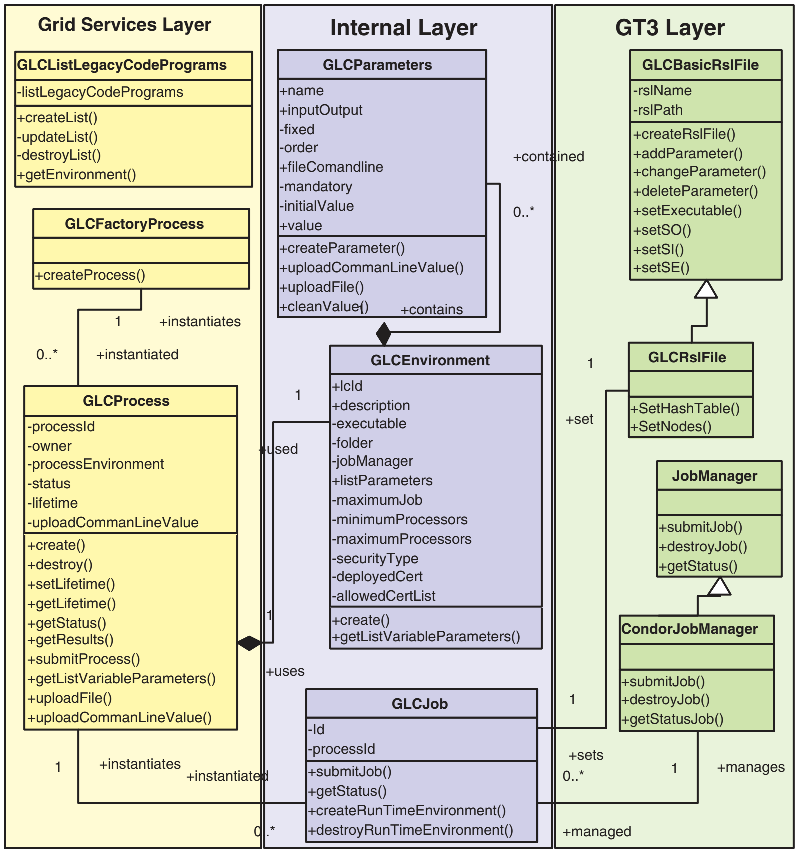

Figure 3. GEMLCA classes design.

full execution management Grid service life-cycle which includes the upload and download of data prior and after the submission and completion of jobs, the handling of data transfer using the GT3 Reliable File Transfer (RFT) service for the upload and download of data between file systems located on different hosts. Future versions will also integrate with the Grid Application Monitoring Infrastructure (GAMI) [18] which includes the GRM trace collector and the Mercury monitor service developed by SZTAKI in the DataGrid and GridLab projects. It also envisaged that the Grid client could be embedded into a GridSphere portlet.

8 Conclusions

There is a clear need to deploy existing legacy code programs as OGSI Grid services. Two approaches can be identified to solve this problem. The first solution is to re-engineer the legacy code which in many cases implies significant efforts. The second one is to develop a front-end OGSI Grid service layer that contacts the target host environment. A solution for the second approach is addressed by the Grid Execution Management for Legacy Code Architecture (GEMLCA) which has been presented in this paper. An initial implementation of GEMLCA addressing a subset of the Grid service life-cycle has been developed and tested on the Westminster Condor pool cluster with the MadCity parallel simulator application. Future work is planned to implement the full life-cycle of the Grid service and integrate with the Grid application monitoring infrastructure.

Acknowledgements

The authors wish to acknowledge the support and contributions of Damian Igbe, Agathocles Gourgoulis and Noam Weingarten in the traffic simulation aspects, and Kreeteeraj Sajadah and Alexandre Beaudouin in investigating and programming GT3.

References

[1] S. Tuecke et al: Open Grid Services Infrastructure (OGSI) Version 1.0, June 2003, `http://www.globus.org/research/papers/Final_OGSI_Specification_V1.0.pdf`

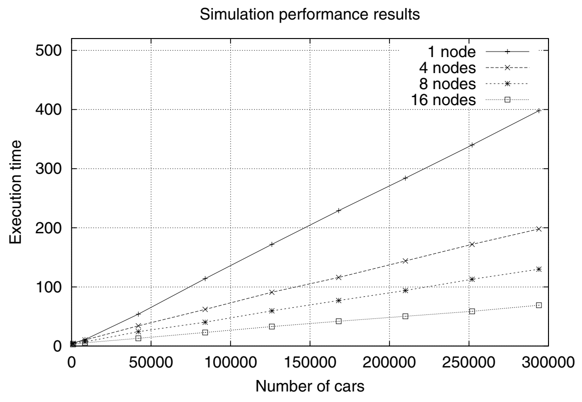

Figure 4. MadCity performance results through GEMLCA architecture.

[2] Globus Team, Globus Toolkit, `http://www.globus.org`

[3] T. Delaitre, A. Goyeneche, T. Kiss and S.C. Winter, Publishing and Executing Parallel Legacy Code using an OGSI Grid Service, Conference proceedings of the 2004 International Conference on Computational Science and its Applications. Editors: A. Lagana et al. LNCS 3044, pp. 30-36, S. Maria degli Angeli, Assisi(PG), Italy, 2004.

[4] D. Kuebler, and W. Eibach, "Adapting legacy applications as Web services", IBM DeveloperWorks, `http://www-106.ibm.com/developerworks/webservices/library/ws-legacy/`

[5] Y. Huang, I. Taylor, D. Walker, and R. Davies, "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

[6] T. Bodhuin, and M. Tortorella, "Using Grid Technologies for Web-enabling Legacy Systems", in Proceedings of the Software Technology and Engineering Practice (STEP), The workshop Software Analysis and Maintenance: Practices, Tools, Interoperability, September 19-21, 2003, Amsterdam, The Netherlands, `http://www.bauhaus-stuttgart.de/sam/bodhuin.pdf`

[7] B. Balis, M. Bubak, and M. Wegiel, "A Framework for Migration from Legacy Software to Grid Services", In Cracow Grid Workshop '03, Cracow, Poland, December 2003, `http://www.icsr.agh.edu.pl/~balis/bib/legacy-cgw03.pdf`

[8] D. Thain, T. Tannenbaum, and M. Livny, "Condor and the Grid", in Fran Berman, Anthony J.G. Hey, Geoffrey Fox, editors, Grid Computing: Making The Global Infrastructure a Reality, John Wiley, 2003

[9] Web Services Description Language (WSDL) Version 1.2, `http://www.w3.org/TR/wsdl12`

[10] Simple Object Access Protocol (SOAP) 1.1. W3C, Note 8, 2000 Center for telecommunication research, Columbia University.

[11] G. Sipos, P. Kacsuk, Connecting Condor Pools into Computational Grids by Jini, 2nd European Across Grid Conference, Nicosia, Cyprus, January 2004,

[12] Ian Foster, et al. Modeling Stateful Resources with Web Services, January 2004, `http://www.globus.org/wsrf/`

[13] Globus Team, Message Level Security, `http://www-unix.globus.org/toolkit/3.0beta/ogsa/docs/message_security.html` 2003.

[14] P. Kacsuk, G. Dozsa, R. Lovas: The GRADE Graphical Parallel Programming Environment, In the book: Parallel Program Development for Cluster Computing: Methodology, Tools and Integrated Environments (Chapter 10), Editors: C. Cunha, P. Kacsuk and S.C. Winter, pp. 231-247, Nova Science Publishers New York, 2001.

[15] D.Igbe, N.Kalantery, S.E Ijaha, S.C Winter, Parallel Traffic Simulation in Spider Programming Environment. In Distributed and Parallel Systems (Cluster and Grid Computing). Edited by Peter Kacsuk et al. Kluwer Academic Publishers, pp 165-172, 2002.

[16] A. Geist, et al. *PVM: Parallel Virtual Machine*, MIT Press, 1994.

[17] A. Gourgoulis, G. Terstyansky, P. Kacsuk, S.C. Winter, Creating Scalable Traffic Simulation on Clusters. PDP2004. Conf. Proc. of the 12-th Euromicro Conference on Parallel, Distributed and Network based Processing, La Coruna, Spain, 11-13th February, 2004,

[18] Z. Balaton and G. Gombos: Resource and Job Monitoring in the Grid. Proc. of EuroPar 2003 Conference, Klagenfurt, pp. 404-411, 2003.