



DELIVERABLE D5.2.6

CURRENT CROSSGRID ARCHITECTURE ASSESSMENT REPORT

WP5 – Task 5.2

Document Filename:	CG5.2-D5.2.6-v1.3-CYF103-CGArch.doc
Work package:	WP5
Partner(s):	CYFRONET
Lead Partner:	CYFRONET
Config ID:	CG5.2-D5.2.6-v1.3-CYF103-CGArch
Document classification:	PUBLIC

Abstract: This document presents the current architecture of the CrossGrid project, with emphasis on services and APIs that are crucial for Grid architecture definition [12]. In addition to a general overview of the current state of Project architecture, a detailed description of interfaces, support for interaction and compatibility with the OGSA approach are given. UML charts are provided as references, depicting both the placement of each module within the overall Project's architecture and its connection to other modules. A separate chapter is devoted to assessment of changes with respect to previous architecture definitions, thus positioning this document as the third in a series of progressive descriptions (with another, final description expected towards the end of the Project's development cycle). This description of the CrossGrid architecture should provide general orientation regarding software development and integration during the last year of the Project.

Delivery Slip

	Name	Partner	Date	Signature
From	CrossGrid Technical Architecture Team	CYFRONET	20/02/2004	
Verified by	CrossGrid Internal Review Board	n/a	26/02/2004	
	Robert Pająk	CYFRONET	1/03/2004	
Approved by	Michał Turała	CYFRONET	1/03/2004	

Document Log

Version	Date	Summary of changes	Author
0.1	24/01/2004	Draft version	Katarzyna Rycerz, Maciej Malawski
1.0	19/02/2004	Major modifications, diagrams/descriptions, proofreading and editing	Marian Bubak, Maciej Malawski, Piotr Nowakowski, Katarzyna Rycerz
1.1	20/02/2004	Corrections after TB comments	Katarzyna Rycerz, Maciej Malawski
1.2	26/02/2004	Corrections after the internal review	Katarzyna Rycerz, Maciej Malawski, Piotr Nowakowski
1.3	1/03/2004	Verified by the Quality Engineer	Robert Pająk

CONTENTS

LIST OF ACRONYMS	5
REFERENCES.....	7
1. EXECUTIVE SUMMARY.....	9
2. INTRODUCTION.....	10
3. OVERVIEW OF CURRENT ARCHITECTURE	11
3.1. OVERALL PICTURE	11
3.1.1. <i>Subsystems</i>	11
3.1.2. <i>Dependencies</i>	12
3.2. CROSSGRID CONTRIBUTION TO GRID ARCHITECTURE DESIGN	13
3.3. INTERFACES TO OTHER GRID PROJECTS	13
4. DETAILED DESCRIPTION OF INTERFACES	14
4.1. APPLICATIONS	14
4.1.1. <i>Medical Application</i>	14
4.1.2. <i>Flood Application</i>	14
4.1.3. <i>HEP Application</i>	14
4.1.4. <i>Integration of Parallel Code for Air Quality Models Into the GRID Structure</i>	14
4.1.5. <i>Wave Models</i>	14
4.1.6. <i>Data Mining – SOM</i>	15
4.2. TOOLS.....	15
4.2.1. <i>MARMOT</i>	15
4.2.2. <i>GridBench</i>	15
4.2.3. <i>PPC Performance Prediction Tool</i>	16
4.2.4. <i>G-PM Performance Measurement Tool</i>	16
4.2.5. <i>Portal</i>	16
4.2.6. <i>Migrating Desktop</i>	17
4.3. SERVICES	18
4.3.1. <i>Roaming Access Server</i>	18
4.3.2. <i>Scheduler</i>	18
4.3.3. <i>Application Monitoring - OCM-G</i>	18
4.3.4. <i>SANTA-G</i>	18
4.3.5. <i>JIMS</i>	19
4.3.6. <i>Postprocessing</i>	19
4.3.7. <i>Optimization of Data Access</i>	19
4.3.8. <i>Grid Visualisation Kernel</i>	19
4.3.9. <i>User Interaction Services</i>	19
5. ASSESSMENT	20
5.1. TOOLS.....	20
5.1.1. <i>MARMOT</i>	20
5.1.2. <i>GridBench</i>	20
5.1.3. <i>PPC Performance Prediction Tool</i>	20
5.1.4. <i>G-PM Performance Measurement Tool</i>	20
5.1.5. <i>Portal and Migrating Desktop</i>	21
5.2. SERVICES	21
5.2.1. <i>Roaming Access Server</i>	21
5.2.2. <i>Scheduler</i>	21
5.2.3. <i>OCM-G Application Monitoring</i>	21
5.2.4. <i>SANTA-G</i>	21

5.2.5. <i>JIMS</i>	22
5.2.6. <i>Optimization of Data Access</i>	22
5.2.7. <i>Grid Visualisation Kernel</i>	22
5.2.8. <i>User Interaction Services</i>	22
6. USER INTERACTION SERVICES	23
6.1. INTERACTIVE STARTUP.....	23
6.2. ONLINE OUTPUT CONTROL	24
6.3. RUNTIME STEERING.....	24
6.4. QUALITY CONTROL	26
7. TOWARDS OGSA COMPATIBILITY	27
8. CONCLUSIONS	29

LIST OF ACRONYMS

API	Application Programming Interface
CA	Certificate Authority
CG	Shorthand for CrossGrid
CrossGrid	The EU CrossGrid Project IST-2001-32243
DataGrid	The EU DataGrid Project IST-2000-25182
EDG	Shorthand for Datagrid
GRAM	Globus Resource Allocation Manager
GSI	Grid Security Infrastructure
G-PM	Grid Performance Measurement
GT	Globus Toolkit
GUI	Graphical User Interface
GVK	Grid Visualization Kernel
HLA	High Level Architecture
HLAC	High Level Analysis Component
HPC	High Performance Computing
HTC	High Throughput Computing
JIMS	JMX-based Infrastructure Monitoring System
JMX	Java Management Extensions
JS	Job Shadow
MD	Migrating Desktop
MDS	Meta Directory System
MPI	Message Passing Interface
MPICH	Implementation of the Message Passing Interface library
OCM-G	Grid-enabled OMIS-compliant Monitoring System
OGSA	Open Grid Services Architecture
OGSI	Open Grid Service Infrastructure
OMIS	Online Monitoring Interface Specification
PMC	Performance Measurement Component
PPC	Performance Prediction Component
QoS	Quality of Service
RAS	Roaming Access Server
RGMA	Relational Grid Monitoring Architecture
RTI	Runtime Infrastructure
SANTA-G	Grid-enabled System Area Network Trace Analysis
SOAP	Simple Object Access Protocol
SOM	Self-Organizing Maps

T	Task
UDAL	Unified Data Access Layer
UIVC	User Interface and Visualization Component
UML	Unified Modelling Language
VNC	Virtual Network Computing
WP	Work Package
WSRF	Web Services Resource Framework
XML	Extensible Markup Language

REFERENCES

- [1] Berman, F., Fox, G., and Hey, T. (Eds): Grid Computing. Making the Global Infrastructure a Reality. Wiley 2003; pp. 873-874.
- [2] Bubak, M., Malawski, M., Zając, K., Architecture of the Grid for Interactive Applications, in: Sloot, P. M. A., et al. (Eds.), Proceedings of Computational Science - ICCS 2003, International Conference Melbourne, Australia and St. Petersburg, Russia, June 2003, Lecture Notes in Computer Science Vol. 2657, Part I, Springer, 2003, pp. 207-213
- [3] Bubak, M., Malawski, M., Zając, K., Towards the CrossGrid Architecture, in: Kranzlmüller, D., Kascuk, P., Dongarra, J., Volkert, J. (Eds.), Recent Advances in Parallel Virtual Machine and Message Passing Interface - 9th European PVM/MPI Users' Group Meeting Linz, Austria, September 29-October 2, 2002, Proceedings, Lecture Notes in Computer Science, Vol. 2474, Springer, 2002, pp. 16-24
- [4] Foster, I., Kesselman, C., Tuecke, S. The Anatomy of the Grid. Enabling Scalable Virtual Organizations. International Journal of High Performance Computing Applications 15 (3) 200-22 (2001) <http://www.globus.org/research/papers/anatomy.pdf>
- [5] Gerndt, M. (Ed.): Performance Tools for the Grid: State of the Art and Future. APART White Paper, Shaker Verlag 2004, ISBN 3-8322-2413-0
- [6] Zając, K., Bubak, M., Malawski, M., Sloot, P., Towards a Grid Management System for HLA-based Interactive Simulations, Proceedings of The Seventh IEEE International Symposium on Distributed Simulation and Real Time Applications October 23-25, 2003 Delft, The Netherlands, IEEE Publishing, 2003, pp. 4-11
- [7] Zhao, Z., van Albada, G. D., Tirado-Ramos, A., Zając, K., Sloot, P. M. A., ISS-Studio: A Prototype for a User-Friendly Tool for Designing Interactive Experiments in Problem Solving Environments, in: Sloot, P. M. A., et al. (Eds.), Proceedings of Computational Science - ICCS 2003, International Conference Melbourne, Australia and St. Petersburg, Russia, June 2003, vol. I, no. 2657, Lecture Notes in Computer Science, Springer, 2003, pp. 679-688
- [8] D1.4 Demonstration and report on WP1 2nd Prototype, <http://www.crossgrid.org/M24deliverables.htm>
- [9] D2.5 Demonstration and report on WP2 2nd Prototype, <http://www.crossgrid.org/M24deliverables.htm>
- [10] D3.5 Report on the results of the WP3 2nd and 3rd prototype, <http://www.crossgrid.org/M24deliverables.htm>
- [11] D5.2.2 First definition of CrossGrid architecture, <http://www.crossgrid.org/M3deliverables.htm>
- [12] D5.2.5 Detailed report on the CrossGrid Architecture., <http://www.crossgrid.org/M15deliverables.htm>
- [13] The CrossGrid Project Homepage, <http://www.crossgrid.org>
- [14] The WS-Resource Framework, <http://www.globus.org/wsrf/>
- [15] Globus Project HomePage <http://www.globus.org>
- [16] DataGrid Project HomePage <http://www.eu-datagrid.org>
- [17] Grid Visualisation Kernel page <http://www.gup.uni-linz.ac.at/~hr/UVA+GVK/>

- [18] DataGrid WP3 website <http://hepunx.rl.ac.uk/edg/wp3/documentation/doc/api/java/>
- [19] JIRO technology <http://www.jini.org/>
- [20] JMX technology <http://java.sun.com/products/JavaManagement/>
- [21] Condor-G <http://www.cs.wisc.edu/condor/condorg>
- [22] VNC <http://www.realvnc.com/what.html>
- [23] HLA specification <http://www.sisostds.org/stdsdev/hla/>

1. EXECUTIVE SUMMARY

This deliverable presents the current CrossGrid architecture with emphasis on protocols, services and APIs that are crucial for Grid architecture definition [1]. The previous deliverable [12] presented the second version of the CrossGrid software architecture, general overview of software components (applications, tools, and CrossGrid-specific Grid services), their connections as well as dependencies on external components from the DataGrid project and Globus Toolkit. This deliverable should serve as the next step in the evolution of CrossGrid architecture, taking into account changes that have affected the Project over the last several months and the development work carried out in that period.

It goes without saying that the architecture of a complex project, especially one that bases on newly-developed technology (which certainly applies to the concept of the Grid and its current implementation) undergoes permanent evolution, to keep up with technological progress and changing environment. Thus, some changes have inevitably made their way into the CrossGrid architecture definition as presented in this document. This should not be perceived as a warning sign, but rather as a manifestation of the fluid nature of today's Grid technologies and their underlying concepts. It should be noted that the last two years of CrossGrid development saw no less than two major paradigm shifts in the Grid community - from the "traditional" set of tools as exemplified by the Globus 2.x packages, through the revolutionary concept of Grid Services and the OGSA architecture, to the brand-new WSRF standard, which ushers in a new generation of Grid middleware.

The attempt at summarizing the CrossGrid architecture, contained in this document, is geared towards presenting the Project as a series of modules (or subsystems), which can be further developed and/or reused once the Project terminates. All the subsystems of CrossGrid are developed as independent software pieces to facilitate the distribution of work between partners, to allow code reusability and exploitation of produced software in the future. However, there are many connections between these subsystems that make CrossGrid an integrated software environment for Grid application developers and users. Thus, each software module is described separately as well as in conjunction with other elements of the Project. The description of the CrossGrid architecture closely follows the 2nd prototype of the Project's software delivered during the integration workshop in Cyprus (January 2004) and reflects the changes introduced during that meeting.

The current description of the CrossGrid architecture is the third of a series of documents (with one final definition expected towards the end of the Project's development cycle). As the Project has now entered a mature phase, the focus of this document shifts from a purely theoretical design to a practical assessment and evaluation of the solutions used in implementation of the Project. Thus, in addition to presenting the next step in the aforementioned architectural evolution, the authors also try to look back and evaluate the progress that has been made over 2 years of development, focusing on conformance with the initial goals of CrossGrid and explaining the reasons behind each of the adopted changes.

This document consists of the following sections:

- general description of CrossGrid architecture, including UML diagrams and description of the current hierarchical model used by CrossGrid,
- detailed description of the role each module plays in the CrossGrid architecture, including dependencies on other modules and/or expressed interfaces,
- assessment of the current state of each architecture component, explaining changes and departures from the original architecture design and rationalizing each of these modifications,
- description of user interactivity services utilized in CrossGrid,
- some remarks on OGSA compatibility.

The description of the CrossGrid architecture presented in this document should provide general orientation regarding software development and integration during the last year of the Project.

2. INTRODUCTION

One of main objectives of CrossGrid is to propose and develop a unified approach to running interactive distributed applications on the Grid, and for this reason, the following issues are the most important for the Project:

- porting applications to the Grid environment; this requires a new approach to the way in which application components are distributed,
- development of user interaction services for interactive startup of applications, online output control, parameter study in a cascade and runtime steering; these services are also oriented on MPI applications,
- development of the Migrating Desktop which is an advanced user interface and enables straightforward integration of applications and tools (it relies on a piece of Grid middleware called the Roaming Access Server),
- tools for online, interactive performance analysis (G-PM) combined with online monitoring of Grid applications,
- development of a scheduler for distributed interactive applications,
- advanced benchmarks and performance prediction for interactive applications,
- optimization of data access for various storage systems.

The architecture of the CrossGrid software was defined at the very beginning of the Project as a result of detailed analysis of requirements from applications [9], [3], and in its first form it was recently presented in [1]. During the progress of the Project the architecture was refined - see [12], [2].

The usage of Globus Toolkit 2.x was decided upon at the beginning of the Project for stability reasons and because of close collaboration with the DataGrid project. Meanwhile, the Globus Alliance has introduced the Open Grid Services Architecture (OGSA) as a general vision of future Grid systems and the Open Grid Services Infrastructure (OGSI), as the basic standard. In January 2004, the Globus Alliance announced the next step: the OGSI standard would evolve towards the new Web Services Resource Framework (WSRF). For our Project it is, therefore, important to develop software in such a way that it would be easy to use in future Grid systems based on OGSA standards. The adoption of Web Services not only makes integration of components an easy task, but also prepares them for migration to future OGSA-compliant architectures, as Grid Services and Web Services merge.

Another important issue is related to running applications using HLA (High Level Architecture) as the basic communication infrastructure. For these applications we are working on solutions based on OGSA [6].

The CrossGrid tools and services are complementary to those of DataGrid and GridLab (with which CrossGrid maintains close collaboration), as well as the US GrADS Project [5].

3. OVERVIEW OF CURRENT ARCHITECTURE

3.1. OVERALL PICTURE

3.1.1. Subsystems

The CrossGrid architecture consists of a set of self-contained subsystems, developed within the Project. These subsystems can be divided into layers that contain applications, software development tools and Grid services. The components and layers are shown in Fig. 1. The application subsystem represents all the applications from WP1. The supporting tools are those developed within WP2 plus the user interfaces for accessing CrossGrid environment, such as Portal and Migrating Desktop. The Grid services layer consists of all the subsystems from WP3, one from WP1 (GVK) and external ones (EDG, Globus). In addition to the CrossGrid components, we also distinguish the most important external subsystems used, namely Globus Toolkit, EU DataGrid and MPI libraries.

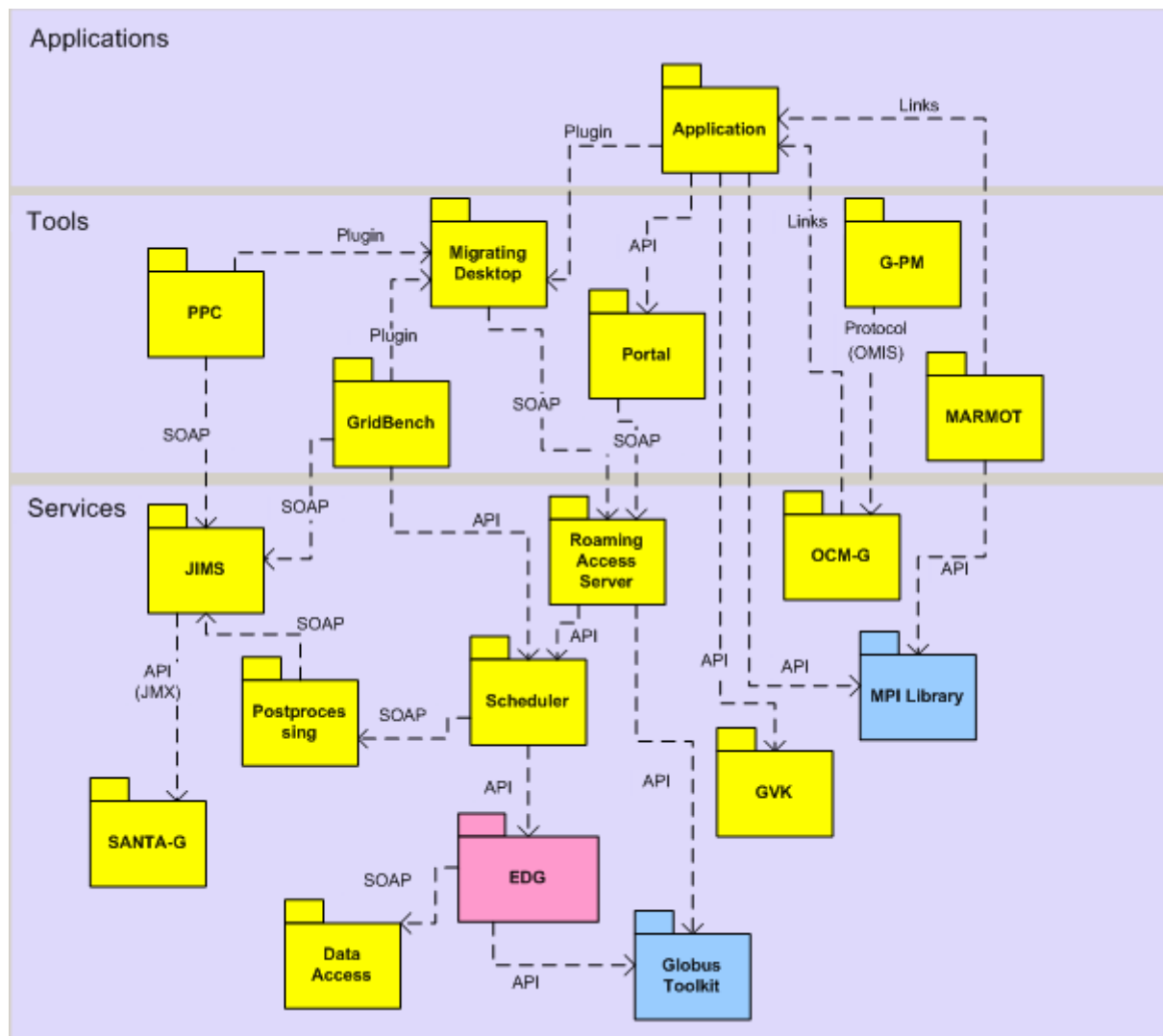


Fig. 1 The layered architecture of CrossGrid

The subsystems are shortly described in the table below, while detailed descriptions follow in subsequent sections of this document.

Subsystem	Short Description	WP/Task
Application	CrossGrid Applications	1.1-1.4
GVK	Grid Visualization Kernel	1.1
MARMOT	MPI Verification Tool	2.2
GridBench	Grid Benchmarks	2.3
PPC	Performance Prediction Tool	2.3
G-PM	Performance Analysis Tool	2.4
Migrating Desktop	User GUI Front-end to CrossGrid	3.1
Roaming Access Server	Server for Migrating Desktop	3.1
Portal	Lightweight Web-based frontend	3.1
Scheduler	CrossGrid scheduling extensions to EDG Resource Broker	3.2
Postprocessing	Service for postprocessing of monitoring data for the Scheduler	3.2
SANTA-G	System for network infrastructure monitoring	3.3
JIMS	JMX-based Infrastructure Monitoring System	3.3
OCM-G	Application Monitoring System	3.3
Data Access	Subsystem for optimization of data access	3.4
EDG	EU DataGrid Software	
Globus Toolkit	Globus Toolkit components from EDG distribution	

3.1.2. Dependencies

All the subsystems of CrossGrid are developed as independent software pieces to facilitate the distribution of work between partners, to allow code reusability and exploitation of produced software in the future. However, there are many connections between these subsystems that make CrossGrid an integrated software environment for Grid application developers and users. These connections are depicted in Fig. 1 using an UML dependencies diagram. Arrows run from subsystems that use other CrossGrid modules to the ones being directly used by them. The picture also includes types of interfaces. Thick lines correspond to interfaces that are developed and running, while thin lines show the interfaces that are agreed upon, but still under development.

The following kinds of interfaces are used:

- SOAP – the Web Service interface is used,
- Plugin – the applications and tools use plugins that are installed in the Migrating Desktop,
- Protocol – the components communicate using a specific protocol other than SOAP,
- API – the components use the Application Programmer Interfaces to communicate directly.

- Links – the tools are directly linked to the application executable, for other cases than for using its API (i.e. in OCM-G the application is linked to OCM-enabled MPICH implementation; for technical details see [10])

3.2. CROSSGRID CONTRIBUTION TO GRID ARCHITECTURE DESIGN

The approach to Grid architecture presented in [12] emphasizes the identification and definition of protocols and services, first; and the Application Programming Interface (APIs), second. We have identified interfaces between CrossGrid components as follows:

Protocols

Protocols are critical to interoperability [12]. A protocol definition specifies how distributed system elements interact with one another in order to achieve a specified behavior, and the structure of the information exchanged during this interaction. This focus on externals (interactions) rather than internals (software, resource characteristics) has important pragmatic benefits.

Apart from standard Web Service protocols (described below), CrossGrid uses and develops a standard protocol for on-line monitoring called the Online Monitoring Interface Specification (OMIS) used as an interface between the performance monitoring tool and the application process being monitored.

Services

A service [12] is defined solely by the protocol that it speaks and the behavior that it implements. The definition of services abstract away resource specific details that would otherwise hinder the development of VO applications.

The CrossGrid architecture contains several services based on Web Services standards. Roaming Access Server, Optimization of Data Access, and Infrastructure Monitoring JIMS expose their services as a WSDL description and speak the SOAP protocol.

APIs

APIs can accelerate code development, enable code sharing, and enhance application portability. However, APIs are an adjunct to, not an alternative to, protocols [12].

In our architecture, the components that use APIs are: RAS (uses the APIs of the Scheduler and the Globus Toolkit), Scheduler (uses the EDG workload management API), Santa-G infrastructure monitoring (uses the RGMA API), applications (use MPI, GVK and JetSpeed APIs) and finally the Marmot tool (uses the MPI API).

3.3. INTERFACES TO OTHER GRID PROJECTS

Dependencies:

Globus [15]. GRAM and Globus MDS are directly used by the EDG workload management system [16], which is, in turn, used by the Scheduler for supporting job submission services.

Globus GridFTP is directly used by the Roaming Access Server for file transfer.

Globus Security Infrastructure (GSI) is used throughout the Project for identifying users in order to perform job submission and file transfer.

EDG RGMA is used by the CrossGrid infrastructure monitoring tool Santa-G.

Interfaces:

The EDG Data Management System uses CrossGrid Optimisation of Data Access. The second prototype is fully integrated with EDG Optor/Reptor [16]. The presented solution has been able to function for both projects and makes mutual integration seamless.

4. DETAILED DESCRIPTION OF INTERFACES

4.1. APPLICATIONS

4.1.1. Medical Application

The Medical Application requires a distributed environment consisting of simulation, interaction and visualization components which will allow the user to change simulation parameters in near-real time (**remote steering**).

The medical application is going to be integrated with the Migrating Desktop and GVK (in progress) [8].

4.1.2. Flood Application

A second type of interactive applications is required for the flood prediction and protection system. An interactive Grid system fulfilling the needs of this application should allow experts to prepare **cascades of** meteorological, hydrological and hydraulic **simulations** basing on the assumption that each preceding step of the cascade produces input for the next simulation. After each of the steps is completed, the expert should be allowed to decide whether there is a need for the next simulation step in the cascade to be performed.

The flood application is integrated partially with the MD for submitting single simulations. It is also integrated with the Portal for submitting cascades of simulations (simple workflow). It uses RAS client API and the JetSpeed library for developing its own portlet for automatically submitting such cascades.

4.1.3. HEP Application

The High Energy Physics [8] application requires support from an interactive system that allows for **on-line progress monitoring** of their results in order to help operators decide about further job execution (i.e. interrupting the execution or letting it finish). The jobs are mainly MPICH-G distributed jobs.

The application is integrated with the Migrating Desktop via a VNC mechanism (see Section 6.2)

4.1.4. Integration of Parallel Code for Air Quality Models Into the GRID Structure

This application consists of two major components: weather prediction, providing input to air pollution simulations. The purpose of this application is to show the usefulness of Grid tools in air pollution modelling. STEM-II, an Eulerian air quality model, is used to simulate the environment. Model forced by meteorological data generated by atmospheric model COAMPS simulates gaseous and aqueous concentration fields of each modelled species, reaction rates, amount of deposited species and ionic concentrations. This application is integrated with the Migrating Desktop using the plugin mechanism.

4.1.5. Wave Models

The purpose of this application is to show the usefulness of Grid tools for maritime applications. The user responsible for prevention of sea pollution and warnings will use the atmospheric model to simulate the nested spectrum of weather conditions over the Baltic Sea and to produce initial and boundary data for sea wave models. The application is integrated with the Portal.

4.1.6. Data Mining – SOM

The purpose of the application is to implement Grid tools for a sophisticated statistical pattern recognition algorithm. The method called SOM (self-organising maps) is one of clustering algorithms in which the training process includes a neighbourhood adaptation mechanism. Thus neighbouring clusters are quite similar, while more distant clusters become increasingly diverse. The main goal is to design an adaptive scheme for distributing data and computational load according to the changing resources available for each Grid job submitted. The data mining algorithms have been migrated to Grid and clusters. The application is integrated with the Portal.

4.2. TOOLS

4.2.1. MARMOT

The MPI verification tool MARMOT [9] is a tool that tries to verify the correct usage of MPI. Both the C and Fortran language binding of MPI standard 1.2 are supported. The Fortran interface is implemented as a wrapper on top of the C interface.

MARMOT is a library that is linked to the MPI application in addition to the existing MPI library and that allows a detailed analysis of this application at runtime.

Interfaces:

Output is written to standard output streams (stdout and stderr) by default and can be redirected to a file by the user. The format is human-readable.

Dependencies:

As MARMOT is intended to be a portable tool running on any environment, there is no specific hardware required. The following software is required to compile MARMOT and link it to applications: MPI library, C++ compiler, Fortran compiler

4.2.2. GridBench

The CrossGrid benchmark suite, GridBench [9] allows to investigate the performance of Grid constellations as well as Grid applications by users or administrators. In this way, benchmarks may provide reference data for the high-level analysis of applications as well as parameters for performance prediction.

Interfaces:

The GridBench provides the user with the GUI that is integrated in the Migrating Desktop. The GUI gives the following operations:

- Defining a benchmark
- Browsing the Results
- Creating charts

The output of the benchmark is an XML document that is stored in a Xindice database. This database can be queried in order to retrieve the results of benchmarks.

Dependencies:

The GridBench is launched like a normal grid application via Migrating Desktop. There is work in progress to include the output from JIMS and Santa-G infrastructure monitoring in the benchmark results.

4.2.3. PPC Performance Prediction Tool

The tool enables predicting performance of applications running on the Grid [9]. After analysis of performance of application kernels depending on input parameters, the tool can predict how long it will take to run a specific application on a specific site.

Interfaces:

The interface to the tool is a GUI that is integrated in Migrating Desktop. The tool allows choosing the available application kernels and site configuration parameters. The output is presented in the form of charts.

Dependencies:

The PPC is a standalone tool and runs as a whole inside Migrating Desktop. There are plans to use monitoring data from NWS or JIMS monitoring systems.

4.2.4. G-PM Performance Measurement Tool

The performance measurement tool [9], called G-PM, consists of three components:

- a performance measurement component (PMC),
- a component for high level analysis (HLAC),
- a user interface and visualization component UIVC.

The PMC component provides the functionality for basic performance measurements of both Grid applications and the Grid environment. The results of these basic measurements can be directly visualized by the visualization component. In addition, they can serve as an input to the high level analysis component and the performance prediction component.

The HLAC component supports an application and problem specific analysis of the performance data. On the one hand, it allows to measure application specific performance metrics. On the other hand, it provides a specification language which allows combining and correlating different performance measurements in order to derive higher-level metrics. The objective of this approach is to provide more meaningful data to application developers.

The UIVC component allows to request performance measurements and displays the resulting performance information in various graphical ways.

Interfaces:

The interface to the GP-M is provided as a GUI (UIVC). It is an X11 application running on Linux.

The input to the HLAC is a file containing measurements defined in the Performance Measurement Specification Language (PMSL).

Dependencies:

The G-PM tool uses the OCM-G application monitoring service. It connects using OMIS interface, which is a text-based protocol using direct TCP connection.

4.2.5. Portal

The Portal [10] is a user interface that allows running DataGrid commands for job submission. It exposes a user interface, and it uses the web Service interface (SOAP over HTTP) to communicate with the Roaming Access Server.

Interfaces:

The Portal provides a graphical interface to users.

In addition, the Portal provides an interface to the flood application by using JetSpeed and RAS client (Java API) for job submission (the flood application team wrote a separate portlet using this library).

The Portal is also used as a user interface to application of Wave models and Weather prediction (Neural Networks.)

Dependencies:

The portal uses the interface of the Roaming Access Server - RAS through Web services (using SOAP over HTTP)

4.2.6. Migrating Desktop

The Migrating Desktop [10] acts as an advanced user interface to the Roaming Access Server.

The various CrossGrid applications and tools are integrated with Migrating Desktop by means of plugins that are designed to act similar to those used in popular browsers. A plugin is a module that can be easily implemented and integrated with its “parent” application (Migrating Desktop).

Plugins are developed by application and tool developers. The main aim is visualization of Grid application output and integration of Java applications (like e.g. Grid tools implemented within WP2, the VNC viewer, etc) with the Migrating Desktop

Application plugins are used to input application-specific input parameters. This enables a portal framework that is independent of application types so that it is simple to extend it by adding new applications. Tool plugins serve to modify job parameters according to the specific tool’s needs.

Each plugin is delivered as a set of Java archive files (available at some network location) from which the Migrating Desktop loads classes dynamically using the Java reflection mechanism. Details can be found in Task 3.1 deliverables.

Additionally, for interactive application it is possible to use MD to launch a VNC Server on a RAS machine. A VNC client is integrated with the MD. In this way, advanced graphical visualisation tools can be used to interact with the application.

Interfaces:

Provides interface to application and tools by Java API and JAR plugins.

Applications that use the MD:

- Medical (application plugin - development phase)
- Flood (single job submission - application plugin)
- HEP (VNC Server)
- Air pollution (application plugin)
- Atmospheric models - weather prediction (application plugin)

Tools that use the MD:

- MARMOT (tool plugin - development phase))
- GridBench (tool plugin)
- PPC (tool plugin)
- OCM-G (tool plugin)

Dependencies:

Uses the RAS interface (Web services SOAP/HTTP)

4.3. SERVICES

4.3.1. Roaming Access Server

This component provides access to the Grid [10]. In general, its functionality should be accessible for tools and applications via the Migrating Desktop or Portal as described below. RAS exposes its interface as a Web Service (SOAP over HTTP). It uses Scheduler API for job submission and GridFTP API for data transfer.

Interfaces:

Provides interface to MD, and Portal by Web services (SOAP/HTTP).

Dependencies:

Uses interface of Scheduler via Java EDG Job Submission Service API and Globus GridFTP via Java CoG API.

4.3.2. Scheduler

Scheduler is integrated with DataGrid Job Submission Service (JSS). They improve EDG JSS components: Workload Manager, Matchmaker, Resource Broker, Job Adapter and Job Controller [10]. Scheduler exposes its interface as an EDG JSS Java API, which is used by RAS (see above). This interface allows for job submission and retrieval of its status and output.

Interfaces:

Provides interface to RAS by Java EDG JSS API.

Dependencies:

Extends DataGrid code for providing additional functionality and will use postprocessing interface in the future.

4.3.3. Application Monitoring - OCM-G

Application monitoring provided by the OCM-G is used by the G-PM Tool and is available via a standard protocol - OMIS [12]. The OMIS 2.0 specification defines also a set of six API functions, but they are merely for connecting to the monitoring system, sending the OMIS requests, etc. There is no real API which encapsulates the protocol - the user must know it and specify the monitoring requests practically at the level of protocol messages. For a detailed description of the API mentioned above, refer to [10].

Interfaces:

Provides interface to G-PM tool via OMIS protocol.

Dependencies:

Uses MD by means of plugins.

4.3.4. SANTA-G

The Santa-G system [10] for non-invasive monitoring of infrastructure publishes its data into DataGrid R-GMA by using the CanonicalProducer API.

The Viewer module provides a Java Swing GUI that presents a graphical interface to users. It allows users to collect data from the R-GMA, by using the Consumer API, and to graphically view packets stored in the log files.

The API for the CanonicalProducer and CanonicalProducer Servlet form part of the DataGrid R-GMA and can be found at the DataGrid WP3 website [18].

Interfaces:

Provides interface to DataGrid R-GMA, GridBench (in the future) via Java API, JIMS by JMX interface.

Dependencies:

Does not use external components of CrossGrid (low layer of CG architecture).

4.3.5. JIMS

Jims provides infrastructure monitoring data using Java Technology. It exposes its interface as a SOAP/HTTP Web Service interface, that is going to be used by GridBenchmarks, Performance Prediction Tool and Postprocessing. A detailed description of this interface can be found in [10].

Interfaces:

Provides interface to Grid Benchmark, Performance Prediction Tool and Postprocessing via Web services (SOAP/HTTP)

Dependencies:

Uses Santa-G by JMX interface.

4.3.6. Postprocessing

The main objectives of the postprocessing monitoring system are to build a system for gathering monitoring data from the Grid, such as cluster load and data transfers between clusters, to build a central monitoring service to analyze the above data and provide it in the format suitable for the scheduler which is to be the main task's data consumer, and, finally, to use the prediction built for the first prototype to forecast the collected Grid parameters.

This task processes data derived from infrastructure monitoring (JIMS). For details see [10]

Interfaces:

Provides interface to Scheduler.

Dependencies:

Uses interface to JIMS via Web services (SOAP/HTTP).

4.3.7. Optimization of Data Access

This task provides routines to estimate bandwidth and latency for physical data access (for details see [10]). The system exposes its interface as a Web service interface (SOAP/HTTP) that is used by DataGrid Optor [16].

Interfaces:

Provides interface to DataGrid Optor via Web services (SOAP/HTTP).

Dependencies:

Does not use external components of CrossGrid (low layer of CG architecture).

4.3.8. Grid Visualisation Kernel

This tool provides a visualization engine running in the Grid environment [17] and is used by the medical application [8].

4.3.9. User Interaction Services

These services are realized in various ways according to different demands for interactivity. Details are described in section 6.

5. ASSESSMENT

5.1. TOOLS

5.1.1. MARMOT

The tool produced by this task, called MARMOT, has been designed as a library linked to the MPI application. The tool produces its output in the form of a file containing a log of MPI function calls and a list of warnings and possible errors.

The current prototype of MARMOT is being developed according to this design. There have been no changes to its planned functionality and its placement in the architecture of the CrossGrid project. There has been, however, additional functionality implemented, namely integration with the Migrating Desktop. It is done by means of a dedicated plugin that enables switching the debugging on and off in the application submitted to the Grid via the Migrating Desktop.

The MARMOT tool has also been integrated and tested with applications. The results of these tests are included in the current report of WP2.

5.1.2. GridBench

The GridBench tool has been designed to support the execution of benchmarks on the Grid and presenting the results to the user. These functionalities are implemented in the current prototype. As planned, the benchmarks are using the EDG Resource Broker to submit jobs to the Grid and provide output through a specialized plugin running in the Migrating Desktop.

According to the design, the benchmarks would also possibly use information from the monitoring services of CrossGrid. Currently the JIMS infrastructure monitoring is used for the acquisition of the system load on the nodes where benchmarks are running. A Web Services interface is used to connect the tool to the monitoring service. The SOAP protocol was also used for communication between JIMS and GridBench. More work on integrating benchmarking with the monitoring systems is currently in progress.

5.1.3. PPC Performance Prediction Tool

According to the new Technical Annex, the Performance Prediction Tool (PPC) was moved from Task 2.4 to Task 2.3. The functionality of this tool is to predict the performance of the application on the Grid based on the analytical models extracted from the performance measurement. The current prototype provides this functionality. The tool is integrated with the Migrating Desktop, what makes it easily accessible for the CrossGrid user. Integration with the monitoring system, namely JIMS, is currently in progress.

GridBench and PPC teams have worked together to analyze the computational kernels of applications from WP1. This activity resulted in including the VERTLQ kernel from WP1.4 in both tools.

5.1.4. G-PM Performance Measurement Tool

The G-PM tool developed in this task was designed to use the OCM-G (Task 3.3.3) monitoring system. The objective of the tool is to produce a higher-level graphical representation of online performance measurements and to support user-defined metrics.

The current prototype is implemented according to the plans. Co-operation between OCM-G and G-PM is proceeding smoothly. The systems are integrated and communicating through the OMIS protocol. The tool was tested with applications from WP1 (daveF from WP1.2 and HEP NN from WP1.3). The tool is an X11 Linux application so it has to be installed on the local (client) machine, however it is also possible to run it from the Migrating Desktop.

5.1.5. Portal and Migrating Desktop

The Migrating Desktop and Portal developers (Task 3.1) focused on improvement of existing features. Functionality has been extended by a number of integrated applications. The major aim was to achieve better stability of software and support for developers of any Grid application .

Portal is light client with limited functionality. It should be used for simple tasks. Migrating Desktop requires downloading Java libraries at user computer, but it provides more features for her.

Additionally, important changes have been introduced since the first prototype of the **Application Portal** (AP). The most important of those is the abandonment of the PHP-Nuke technology and the fact that portal developers have moved from the Apache's Jakarta Project to the Jetspeed framework. Jetspeed is an open source implementation of an enterprise portal system and it is based on the Java programming language and XML.

5.2. SERVICES

5.2.1. Roaming Access Server

In **Roaming Access Server** (RAS) development, technologies have remained unchanged since the first prototype. The Application Portal is based on the RAS machine, which provides Web services. Those services (e.g. Job Submission Services) allow the user to submit jobs and perform other tasks on the CrossGrid testbed, such as obtaining the status of jobs that have been submitted, or obtaining information regarding nodes available in the testbed.

5.2.2. Scheduler

Work within this task is focused on development of a system for **scheduling parallel applications** on the Grid and a **postprocessing monitoring system** delivering information for better tuning of the scheduler. The **scheduling parallel applications** system is built on top of the EDG's Resource Broker. The main changes since the first definition of the architecture include the shift of the postprocessing task from the infrastructure monitoring component (see below) to the scheduler task.

5.2.3. OCM-G Application Monitoring

The **OCM-G's** (Grid-enabled OMIS-compliant Monitoring System) purpose is to be an intermediate layer between tools for application development support and applications running on the Grid. The main changes in OCM-G features are as follows: firstly, full GSI-based security has been introduced. The user now requires a proper certificate to use the monitoring system. Secondly, configuration facilities have been implemented. Local Monitors are now able to discover Service Managers with the use of a "connection string". Consequently, neither a shared file system nor a configuration file is no longer required by the OCM-G. Thirdly, a new service to return a list of functions used by a program has been added. This enables the restriction of defined measurements to the level of functions.

5.2.4. SANTA-G

The **SANTA-G** system provides a generic template for ad-hoc, non-invasive monitoring with external instruments. The first prototype had an incomplete schema of monitoring data available, a limited set of SQL statements supported, incomplete functionality of the Sensor component of the SANTA-G system, and inadequate security access. Currently, the schema of information available is complete. The SQL supported by the SANTA-G QueryEngine component is now the same subset as that supported by the DataGrid R-GMA. Also, the functionality of the Sensor component is now complete. Thus, all problems except security have been fully overcome, as planned. Security issues will be addressed in the remaining time until the end of the CrossGrid project, again as planned.

5.2.5. JIMS

JIMS is an infrastructure monitoring system for exposing operating system parameters (CPU statistics, the number of processes, memory used, filesystem statistics), and network infrastructure (SNMP attributes) parameters to external monitoring applications through Web Services, as specified in the OGSA/OGSI specifications. The second prototype of JIMS has been completely redesigned due to the support for JIRO [19] technology, on which the first prototype was based, being withdrawn by SUN.

The main changes in architecture include the use of R-GMA, which turned out to be useful only for SANTA-G purposes, not for both infrastructure monitoring systems as stated in previous definition of CrossGrid architecture.

As stated, the previous architecture [12] was based on JIRO technology. It used a special tool called Bean Shell for manipulating monitoring stations as Java objects, and instrumenting them administratively (manually). This concept required the use of graphical tools during the monitoring system deployment. The current version of JIMS has been redesigned to overcome these inconveniences. Its name has been changed from the JIRO Infrastructure Monitoring System to the JMX Infrastructure Monitoring System [20]. In fact it is now based on a pure JMX reference implementation and uses no graphical interface during deployment. This allows the process to be automated using shell scripts. The main part of the work involved was complete refactoring of the source code, and redesigning of the parts in which JIRO was used. The dynamic discovery and heartbeat facilities were implemented.

5.2.6. Optimization of Data Access

Task 3.4 is involved in data access optimization for interactive applications, and also copes with storage heterogeneity. The proposed **Unified Data Access Layer** (UDAL) is expected to simplify access to Grid storage and make it simpler and more efficient. As a result of cooperation with EDG WP2, the presented solution was partially integrated with EDG Optor [16].

This second more advanced prototype is fully integrated with EDG Optor/Reptor. The presented solution was able to work for both projects and makes mutual integration seamless.

5.2.7. Grid Visualisation Kernel

In the previous design of the CrossGrid architecture [12] this tool was planned to serve as the visualization engine for CrossGrid applications, especially the biomedical and flood applications. According to reported application functionality, it is currently being employed by the biomedical application.

5.2.8. User Interaction Services

These services were presented in [12] as a conceptual service on the application-specific services layer. The functionality of this service is divided between a few modules in current CrossGrid architecture as described below and realized in various ways according to different demands for interactivity. Details are provided in section 6.

6. USER INTERACTION SERVICES

CrossGrid involves various requirements for support of interactive applications.

The Medical Application [8] is a distributed application consisting of three base components: simulation, interaction and visualization. The interaction component allows the user to change simulation parameters in near-real time. Therefore, support for **remote steering of simulation running on the Grid is required**. Another type of interactive applications is represented by the Flood Protection system [8]. An interactive Grid system fulfilling the needs of this application should allow experts to prepare **cascades of meteorological, hydrological and hydraulic simulations** based on the assumption that each preceding step of the cascade produces input for the next simulation. After each of the steps is completed, the expert should be allowed to decide whether there is a need for the next simulation step in the cascade to be performed. The main requirement for interaction is the need to support online control of cascade execution

High Energy Physic and Air Pollution Modeling Applications [8] require support for **on-line output monitoring** of their results in order to help operators decide about further job execution (i.e. interrupting the execution or letting it finish). The jobs are mainly MPICH-G distributed jobs, making on-line output monitoring even more difficult.

The classification of the application requirements and their solutions are presented below:

6.1. INTERACTIVE STARTUP

One of main requirements for the application to be interactive is to provide a possibility to start it in a predictable future. This requirement is very hard to fulfill, especially in the current GT2 Datagrid/CrossGrid environment, due to the batch-oriented submission system, when the application is distributed over Grid sites (such as MPICH-G jobs).

One important part of the system developed by Task 3.2 is the MPICH-G2 Application Launcher [10] that provides support for executing MPI applications on a distributed set of Grid nodes. The Launcher has been built on top of Condor-G [21] and guarantees that all the sub jobs (jobs that constitute a parallel application) are synchronized at the beginning of its execution. For that purpose the Globus DUROC library was used [15]. Also, as described in deliverable 3.5 of Task 3.2 [10], work currently progresses on two additional services that have been included in the resource manager middleware. The first consists of a temporal reservation mechanism included in the Resource Selector module. This mechanism is used to guarantee a time-limited and exclusive access to a set of resources for any given job submitted to the Grid. The second service consists of a remote execution mechanism that can be used to preempt applications which run under this mechanism, so the less important (batch jobs with lower priority) will be suspended and more important (interactive, with higher priority) can be run. This issue is going to be resolved by means of the Condor Glide-In mechanism that allows the user to remove a job running on a particular machine, without losing control over that machine. (This service is expected to be fully operational in the production testbed at month 30. For the time being, it will be used in the development testbed only.) Additionally, in the future, temporal reservation will provide a limited time reservation of CPU resources for MPI jobs. This will help avoid situations where the same available resources are assigned to more than one resource petition, thus creating a deadlock. Deadlocks may also occur when multiple sites are chosen for execution of two or more MPI jobs. If the sets of sites are not disjoint, two jobs may obtain mutually dependent allocations and hence wait for each other indefinitely. Randomized selection will be used by the Resource Selector to generate different answers when there are multiple choices of resources of which all possess the same rank. The technical details of the Application Launcher can be found in [10], in the description of task 3.2.

6.2. ONLINE OUTPUT CONTROL

Once the application is started, it is desirable to control its output online and to allow the user to decide about canceling it according to the output results. This can be achieved with streaming techniques presented in deliverable 3.5 of task 3.1 [10] (Roaming access and migrating desktop).

In order to submit jobs in CrossGrid, the user has to download Migrating Desktop (MD) to his/her local machine. MD allows the user to interact with the Roaming Access Server (RAS). In order to allow users to submit jobs and see the Application GUI, the VNC¹ [22] mechanism will be put in place between the RAS and the MD (which will download the VNC Client² as Java applet), via ssh tunneling. In the future, both VNC server and Application GUI will be able to run on a machine different from the RAS. VNC can present the Virtual Desktop of the RAS machine where an instance of the Application GUI tool is launched.

Following job submission (done by the scheduling described in the section above), the streams (in, out and error) of a job running on a computing element are sent to the RAS machine by a Console Agent (CA). The CA is a process that is launched together with the user job on the Worker Node. The CA opens a connection towards a Job Shadow (JS) running on the RAS machine. The Job Shadow (JS) receives the job streams from the CA. These streams are processed on the RAS machine as so to be sent to the application using VNC mechanisms.

In the future, to avoid RAS overloading, the VNC server will also be able to run on a machine (called the Application GUI machine) different from the RAS. The MD will invoke a Web service made available by the other Job Submission Service on the RAS machine and allow the user to connect to the Application GUI machine.

Technical details can be found in [10] in description of task 3.1

6.3. RUNTIME STEERING

This issue is similar to the one described above, with the additional requirement being the ability to change the simulation execution parameters while it is running.

This requirement can be fulfilled by using the HLA [23] library. This work is done mainly in WP 1.1 Medical Application, since it is the only application within CG that actually requires such functionality. Within Task 1.1, the agent system [5] built over HLA supports development and execution of interactive distributed components (namely simulation, interaction and visualization). Also, there is an experimental attempt to port HLA-based applications to the Grid. The experimental system based on the OGSA [15] concept has been partly implemented and the results can be found in [6]. The tests were performed on the dutch DAS2 testbed [6], not on CrossGrid testbed, because of their experimental nature of tests and difference in technology. Also, there is an issue that implementation of HLA is not an open source project.

It should be noted that the system is experimental and not involved in the main stream of development of the medical application as described in [8]. The aim of the research is to check if OGSA solutions are suitable for such a purpose. It should also be noted that this system aims at supporting fault tolerant execution of HLA-based applications. This requirement cannot currently be fulfilled by existing implementations of CrossGrid Scheduler, GT2 GRAM [15] and Condor-G [21](namely the

¹ TightVNC (<http://www.tightvnc.com/>). TightVNC can be used as it seems more secure than standard VNC

² The VNC server contains a small Web server. It listens for HTTP connections on port 5800 + display number. So if IC process is using display 2 on RAS machine, we can show it on IF pointing to the URL <http://<hostname>:5802/>: a VNC Client (Java applet) will be downloaded in the IF, containing a prompt for user's password, and after login it will display the desktop.

migration of parts of HLA applications that are connect with other parts using the HLA bus). However, there are other parts of CrossGrid software that we plan to use (see below).

The High Level Architecture (HLA) is one of the most widely-used standards for distributed interactive applications and supports development of simulations (called *federations* in HLA nomenclature) comprised of distributed components (called *federates*). It provides the simulation developer with many useful features such as time and data management needed by simulations that are time-driven or event-driven. However, the HLA standard was developed assuming a certain quality of service in the underlying environment of simulation execution, which is not provided by the Grid.

We have proposed and partially implemented [6] an OGSA-based system that supports running distributed interactive application based on the HLA standard in a Grid environment.

The current architecture of the system is given in Fig. 2.

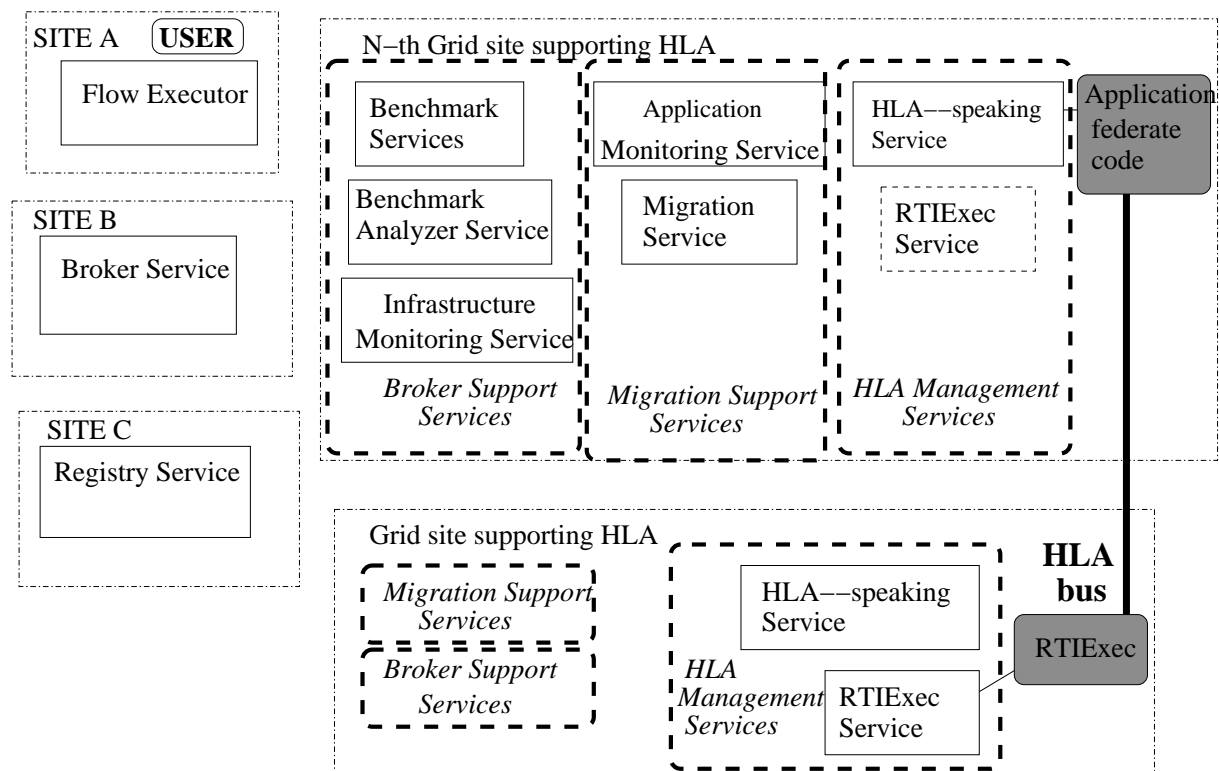


Fig. 2 The architecture of OGSA framework for HLA-based application support

Each site that supports HLA in a Grid environment should provide the following services:

- **HLA Management Services** that interface and manage actual HLA installation on that site,
- **Migration Support Services** that facilitate direct support for fault-tolerant and effective performance of HLA-based applications (work is proceeding on using task 3.3.1 OCM-G for performance monitoring and migration decision support),
- **Broker Support Services** which aim at providing the Broker Service with information necessary for decisions about setup and migration of application components (work is proceeding on using task 3.3.3 JIMS Infrastructure Monitoring that provides a Web service interface).

To explain the role of particular services in Fig. 2 we describe the scenario as follows: the user launches a distributed HLA application using the Flow Executor that is basically a client for the Broker Service. The user supplies the Flow Executor with appropriate codes of the simulation compiled and dynamically linked with the RTI library, as well as the version of that RTI library. Then

the Flow Executor asks the Broker Service to set up the application. The Broker Service asks the Registry Service for a list of appropriate HLA-Speaking Services. An HLA-Speaking Service represents the HLA installation on a particular Grid; it also takes care of interfacing to the local cluster management system. Then, the Broker Service decides which HLA-speaking services to choose and it also chooses an RTI-Exec Service that can start RTIExec coordination process. Apart from an HLA-speaking Service, each Grid site suitable for hosting HLA-based simulations should provide Benchmark Services and Benchmark Analyzer Services that help the Broker Service in making decisions about which HLA-Speaking Service to choose. The Broker Service should take into account the structure of the distributed application and choose appropriate benchmark results according to that structure. Research in task 2.3 (GridBench) is related to that problem however it is oriented towards MPI-based applications rather than HLA-based ones.

6.4. QUALITY CONTROL

Once the application is controlled by the user, near-real time response is desired. Therefore, we must assure a certain quality of service.

Although a CPU reservation mechanism is assumed, there remains the issue of network performance, which can be resolved by diffserv (or other network QoS) mechanism and it is under consideration.

One (partial) solution is to schedule a job according to the network state. This is going to be solved by task 3.2 [10], where scheduling relies on network traffic analysis obtained through monitoring.

Another issue appears when the network changes its state. This can be partly solved by migration. An experimental system supporting migration of HLA-based simulations has been implemented and the results were described in [6]. It assumes that if the network stays busy for some time and the responses take longer, it is better to spend some time on migration to another site, which may provide better response times. It is also less annoying to wait once for migration than to wait for each response. Of course, there is a risk that the network may change its state and thus render migration pointless. It should also be noticed that for some hard real-time interactive applications this scenario is not suitable at all. We aim at applications that assume flexible response times and the CrossGrid medical application is one of them.

7. TOWARDS OGSA COMPATIBILITY

At the very same time when CrossGrid project started, the Globus Project together with IBM and other industrial partners announced an initiative aiming at joining the Grid and Web Services technologies. This initiative included the Open Grid Services Architecture (OGSA) as a general vision of future Grid systems and the Open Grid Services Infrastructure (OGSI), as a basic standard. OGSI is based on Web Services standards, including XML, SOAP and WSDL. OGSI introduces mechanisms for state management in a distributed environment by defining transient and stateful Grid Services as extensions of stateless Web Services. The mechanisms defined in OGSI include lifetime management of service instances, service data for manipulating and querying on service state as well as notification.

OGSI was standardized by the Global Grid Forum and OGSI v.1 was announced in July 2003. By that time, the Globus Project (currently known as Globus Alliance) was already working on the first implementation of OGSI and the new Globus Toolkit 3 (GT3) that was based on the OGSI implementation. GT3 was released in June 2003 and a more stable version 3.2 was expected in the beginning of 2004.

From the beginning of the Project, CrossGrid has been evaluating both the possibility of migrating to the new OGSI-based Globus Toolkit and the necessity of gearing the scientific work towards the service-oriented architecture of the Grid that OGSA envisioned. However, the first implementation of OGSI-based Globus Toolkit 3.0 was only released in June 2003 - the middle of the second year of CrossGrid development. Because of the dependency on DataGrid software, which is based on Globus 2, and also for stability reasons we have decided not to migrate to the new toolkit. Nevertheless, the Project focused on preparing software in such a way that it would be “OGSA-ready”. This entailed the usage of Web Services and XML where appropriate to facilitate the possible future migration to an OGSA-based Grid environment. Moreover, many CrossGrid partners initiated scientific and development work in this area.

In January 2004, the Globus Alliance announced an evolution of OGSA-related standards. The OGSI standard would evolve towards the new Web Services Resource Framework (WS-RF) [9] and other specific standards such as WS-Notification. This evolution is a step towards the unification of (scientific) Grid Services and (commercial) Web Services communities. While it delays the development of Grid systems that are based on these new standards, it is a very important step towards standardization of both scientific and commercial Grid technologies.

From the development point of view, the current plans of the Globus Alliance are to concentrate on a new release, GT4.0, based on the new WS-RF standard. This release is announced to be available in Q3 of 2004, putting it out of reach of the CrossGrid project.

The table below summarizes the current status of work on OGSA compatibility within CrossGrid tasks. It can be seen that Web Services and XML are extensively used in many tasks, hence possible future migration to OGSA should be easy. In many other tasks there is ongoing research in the area of Grid Services or Service-Oriented Architectures, which means that effort invested in the development of applications, tools and services within CrossGrid could be exploited in the future Grid or industrial environments, based on the currently emerging standards.

1.1 Biomed app.	Experiments with OGSI services to set up and control interactive simulations
1.2 Flood app.	Uses OGSI services for metadata management
1.3 HEP app.	Uses XML for distributed data access
1.4 Meteo app.	No plans for OGSA
2.2 MARMOT	Low-level tool, no plans for OGSA migration

2.3 Benchmark	Uses Web Service interfaces and XML, produces output into XML database. Easy migration to OGSA
2.3 PPC	Tool executed locally, no plans for OGSA migration
2.4 G-PM	Initiated research towards performance of service-based applications
1.1 GVK	No plans for OGSA migration
3.1 Portal	Uses Web Services, easy migration to OGSA
3.1 Migrating Desktop	Uses Web Services and XML, possible migration to OGSA
3.1 RAS	Uses Web Services and XML, easy migration to OGSA
3.2 Scheduler + Postprocessing	Uses Web services for communication
3.3.1 OCMG	Initiated research towards monitoring of Grid Services
3.3.2 Santa-G	Plans for migration towards OGSA together with R-GMA
3.3.3 JIMS	Uses Web Services
3.4 Data access	Uses Web Services, possible migration to OGSA

We consider the approach CrossGrid has taken towards OGSA as a good solution for maintaining a long-term project in the midst of substantial changes and evolution of Grid technologies. The usage of GT2 is important for stability reasons and because of close collaboration with the DataGrid project. On the other hand, it is important to prepare the new software in such a way that it will be easy to use in future Grid systems based on OGSA standards. The usage of Web Services not only makes integration of components an easy task, but also prepares them for migration to future OGSA standards. As Grid Services and Web Services merge, it should become possible to migrate to the future common technology.

8. CONCLUSIONS

The deliverable comprises a thorough and in-depth analysis of the current state of CrossGrid architecture following the integration workshop which took place in Cyprus during January 2004. As can be seen, the Project is well on its way to successful completion in the final year on development and the architecture has solidified much when compared with the initial attempts at formulation, expressed at the start of the Project.

The CrossGrid Technical Architecture Team has consistently monitored emerging trends and standards in the global Grid community and has provided input both to Project Management (to make informed administrative decisions) and to individual tasks and developers. Thus, the Project has stayed on course throughout its second development phase and is expected to conclude successfully, fulfilling the obligations set out in the Technical Annex 1.

We hope that the description of CrossGrid architecture presented in this document provides sufficient orientation regarding software development and integration during the last year of the Project.