Project No. FP6-004265

CoreGRID

European Research Network on Foundations, Software Infrastructures and Applications for large scale distributed, GRID and Peer-to-Peer Technologies

Network of Excellence

GRID-based Systems for solving complex problems

D.ETS.03 – Basic Portal's Design Methodology and User Scenarios

Due date of deliverable: 31st August 2005
Actual submission date: 14th October 2005

Start date of project: 1 September 2004 Duration: 48 months

Organisation name of lead contractor for this deliverable:
Cardiff University

Revision: *Final*

Project co-funded by the European Commission within the Sixth Framework Programme (2002–2006)		
Dissemination level		
PU	Public	PU

Keyword list: Design methodologies, component model, tools/system components, Problem Solving Environments, portal components

Contents

1 Introduction **2**

2 User Scenario and Mapping Template **2**
2.1 Grid-Awareness . . . 3
2.2 User Interface . . . 3
2.3 Infrastructure Used . . . 3
2.4 Middleware Used . . . 3
2.5 Service Representation . . . 3
2.6 Workflow Requirements . . . 4
2.7 Runtime Requirements . . . 4

3 User Scenarios **4**
3.1 Bioinformatics applications using BLAST(INRIA) . . . 4
3.2 Weather Forecast Application (UoW, SZTAKI) . . . 5
3.3 Urban Traffic Simulation. (UoW, SZTAKI) . . . 7
3.4 Grid-based e-Marketplace. (UoW, SZTAKI) . . . 9
3.5 Library Level Legacy Code Wrapping. (UWC, FORTH-ICS) . . . 10
3.6 Dynamic Data Driven Application Scenario for Pipeline Modeling. (UWC) . . . 12
3.7 Molecular Dynamics Simulation. (HLRS) . . . 13
3.8 GRID superscalar applications using BLAST. (UPC) . . . 15
3.9 Chemical application - using GAMESS. (UPC) . . . 16
3.10 Performance Analysis Scenario. (UPC) . . . 18
3.11 Hospital Information System. (UoW) . . . 18
3.12 Legacy Systems as Grid Services. (CYFRONET) . . . 21

4 From User Requirement to Portal Design **22**
4.1 Workflow Execution and Representation . . . 23
4.2 Cross VO integration . . . 23
4.3 Middleware Flexibility/Choice - Insulation from Middleware . . . 23
4.4 Dynamic Steering . . . 24
4.5 Dynamic Deployment . . . 24
4.6 Portal Design . . . 24

5 Conclusion **26**

1 Introduction

An effective portal interface is more than just following a set of rules. It requires a user-centered attitude and design methodology. It also requires early planning of the interface and continued work throughout the development process. In this document therefore, we present the design methodology for a portal by examining the requirements from a collection of user scenarios, which have been prepared according to a specific template that was designed to facilitate the mapping between the application primitives and the underlying Grid tools and services. In the following section, we describe the mapping template that shaped the user scenarios. The scenarios themselves are given in Section 3. We then present how these can be mapped into the specification for the Portal design in Section 4.

2 User Scenario and Mapping Template

In the Work package 7 Barcelona meeting, we discussed in detail a number of application or user scenarios that the various partners could provide. From these discussions, we extracted a template, which could be used to serve as a guideline for points that needed to be addressed when constructing such scenarios. We were careful to pay particular attention so that the scenario authors considered the mapping from their scenarios to the available tools and middleware. This template is used as a basis therefore to be able to extract the commonalities in usage patterns and therefore forms the preliminary taxonomy for the design methodology for next-generation portals. This format consisted of two main sections, with points for authors to address within each section, as follows:

1. Scientific User Scenario: Here, the scenario is given from the scientist's perspective. The scenario author should set out the problem paying particular attention to which distributed aspects they require for execution. the author should also specify the Quality of Service (QoS) requirements that are expected to be gained from such a mapping across the distributed resources.

2. Mapping to the underlying infrastructure: Here, the technical mapping from the given user scenario to the underlying infrastructure should be given, paying particular attention to describing the following aspects:

   - Is the application Grid aware or unaware?
   - User Interface
   - Identify type of infrastructure used (e.g. Grid or P2P)
   - Technologies used e.g. GT, GAT, Unicore etc
   - Service representation.
   - Workflow requirements (representation)
   - What kind of monitoring/steering id required?

Each of these categories are expanded upon, in the following subsections.

2.1 Grid-Awareness

A major issue is whether the application the user wishes to run is Grid aware or not. From the users perspective, most would argue, this difference should be transparent. In some cases legacy applications need to to gridified without changing the behaviour from the users point of view. In other cases they may need to be wrapped entirely, for example as a Web services.

2.2 User Interface

As Grid scenarios become more sophisticated, so the user interfaces must keep pace. These need to cope with various underlying resource/service/workflow description technologies, many of which are still evolving, in order to render grid entities. Interfaces need to be flexible but also intuitive and simple in order to handle different user types (for example grid aware users may wish to define things such as resources to use while others may not). The design of resource description mechanisms therefore needs to take these issues into consideration.

2.3 Infrastructure Used

Users may expect differing grid infrastructures depending on the scenario. For example certain applications may require highly dynamic and distributed discovery environments. Others may require server-centric data repositories. The ability to behave flexibly according to users needs in this regard is an important aspect of developing a scalable, generic grid environment.

2.4 Middleware Used

Many existing applications already rely on a middleware layer that may be grid enabled in some way. Users and developers will be reluctant to dismantle existing capabilties in order to experiement with new technologies. It is important therefore to be able to integrate these into an inclusive grid environment, enabling diverse views of a grid to co-exist.

2.5 Service Representation

With new Grid technologies moving towards the service oriented paradigm, shared views of service representation need to be developed. Furthermore, while there are existing standards of service representation and communication with a broad base of acceptance (WSDL and SOAP for example), this area is still in an evolutionary phase. Emerging technologies which are either richer or more efficient need to be able to be integrated when they achieve maturity.

2.6 Workflow Requirements

Workflow is becoming more and more important in Grid user scenarios, in part due to the adoption of SOA which views the network as discreet entities providing well defined services. Understanding the workflow requirements of users scenarios will help in defining generic mechanisms for describing and implementing them.

2.7 Runtime Requirements

The ability to monitor/steer/migrate running applications is paramount in optimising not only application performance, but user performance as well. These requirements become more complex to implement as the underlying distributed topology becomes more complex and more dynamic. For example new workflows may be programatically spawned as a result of previous output, or user preference, and these workflows may themselves be constructed from diverse resources and systems. The interface to this runtime environment must be capable of handling this kind of dynamic activity.

3 User Scenarios

In this section the scenarios received from institutions and projects are listed and, where appropriate, how they relate to the underlying infrastructure.

3.1 Bioinformatics applications using BLAST(INRIA)

Projects: ProActive [21], OASIS [18]

BLAST (Basic Local Alignment Search Tool) [1] is by far the most widely used algorithm for rapid sequence comparison. However, sequence databases are growing at an exponential rate - currently doubling in about 14 months - exceeding Moore's Law for hardware acceleration which is about 18 months. Consequently, sequence comparison against these ever-growing databases is increasingly becoming computationally challenging. To meet these demands, high performance parallel computing methodologies should be applied. However, single query or batch query mode is still used primarily.

There are several freely available softwares which allow users to search the similarity of one biological sequence against others by employing the BLAST algorithm. However, one particular implementation of BLAST developed by NCBI (National Center for Biotechnology Information) [17] stands out of the crowd and remains as an ad-hoc standard till date.

We developed GeB, a GRID-enabled parallel version of NCBI BLAST based on ProActive. The implementation is based on ProActive Group Communication, and uses the standard, unmodified, NCBI software. So the scientist just has to use the normal software, and moreover, when a new version of NCBI BLAST is developed, he or she can just use the new one with ProActive and

the Grid to benefit from the new algorithmic advances.

Mapping to the underlying infrastructure

1. *Grid aware or unaware application* - Grid unaware
2. *Type of infrastructure used* - ProActive environment. We chose the traditional master-slave approach for GeB where the master does most of the non-calculative jobs (like sending the queries to different nodes, merging and building of final results etc.) while the slave nodes do the calculations and return results back to the master.
3. *Technologies used* - The Java code being developed takes advantage of the ProActive deployment infrastructure. Using the notion of Virtual Node, and the capacity to deploy on many protocols and platforms (ssh, rsh, RMI/ssh Tunneling, Globus [8] GT2, GT3, and GT4, sshGSI, LSF, PBS, Sun Grid Engine [25], Unicore [28], EGEE gLite [4], etc.), we can deploy in cluster, Grid, P2P environments.

3.2 Weather Forecast Application (UoW, SZTAKI)

Projects: GEMLCA [7], P-Grade Portal [20]

The main objective of a meteorological nowcasting system is to analyse and predict in the ultra short range (up to 6 hours) those weather phenomena, which might be dangerous for life and property. Typically such events are snow storms, freezing rain, fog, convective storms, etc. The MEsoscale Analysis Nowcasting and DEcision Routines, (MEANDER), developed by the Hungarian Meteorological Service, are the core software components of a nowcasting system.

The MEANDER package consists of five different algorithms. Each calculation algorithm is computation intensive and implemented as a parallel program containing C/C++ and FORTRAN sequential code. The programs are depending on each other, thus the whole MEANDER simulation can be represented by a workflow as it can be seen in Figure 1. An important constraint for the simulation is that the analysis and the 6 hour forecasting should be available within 20 minutes after the measurement time. In order to meet this strict requirement the parallel components must be executed on dedicated, high-availability clusters orchestrated by an automated workflow engine that is capable to perform the execution and the data transfer processes automatically.

The advantage of the workflow approach is that it introduces parallelism at two levels. The top level parallelism comes from the workflow concept, i.e., independent branches of a workflow can be executed simultaneously on different clusters. The bottom level parallelism is the consequence of the fact that the nodes of the workflow are themselves parallel programs.

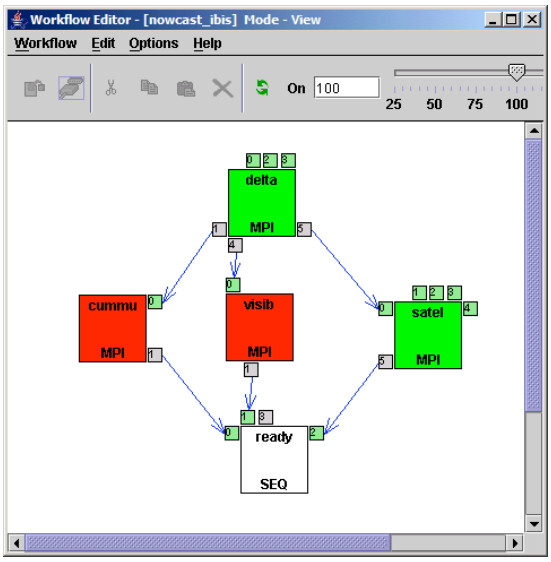

Figure 1: MEANDER weather forecast workflow in the P-GRADE Portal

Mapping to the underlying infrastructure

1. *Grid aware or unaware application* - The components of the MEANDER package have been developed as non Grid aware parallel applications (MPI or PVM) and have been statically deployed on dedicated supercomputers and clusters. These legacy components can be connected as software services into computational Grids using the GEMLCA architecture.

2. *User Interface* - The user interface for workflow creation, execution and visualisation is a Grid portal, like the P-GRADE portal.

3. *Type of infrastructure used* - A grid infrastructure is required where the different components of the workflow are mapped to different clusters and/or supercomputers providing the capability of parallel execution at both job and also at workflow level.

4. *Technologies Used* - The scenario utilises GEMLCA and the P-GRADE portal. GEMLCA is currently using GT4 as underlying Grid middleware but it can be ported to any OGSA compatible service-oriented Grid architecture with reasonable effort. The P-GRADE portal is based on the GridSphere portal framework, and the workflow subsystem of the portal is implemented on top of Condor DAGMan.

5. *Service Representation* - GEMLCA presents each parallel program as a Grid/Web service (a GT4 service in the current implementation) that can be accessed through a standard WSDL interface.

6. *Workflow requirements (representation)* - Once the programs of the MEANDER package are made accessible as Grid services, the different components can be connected into a workflow by using an appropriate developer tool, such as the P GRADE Portal. Although the workflow development process requires significantly less effort than the development of the individual parallel components did, it still assumes some basic programming knowledge. However, once the workflow definition is uploaded to the P-GRADE Portal server, the simulation can be started and controlled by the real users (i.e. the meteorologists) who know where the up to date environmental data (collected by radars, satellites and balloons) can be found.

7. *Runtime requirements* - The whole workflow execution can be monitored through the graphical GUI of the P GRADE Portal. After the simulation terminated the results can be downloaded onto the desktop computer and can be browsed by special editors or visualization tools.

3.3 Urban Traffic Simulation. (UoW, SZTAKI)

Projects: GEMLCA [7], P-Grade [20]

A traffic simulation application is typically built from different functional building blocks. For example one module generates a road network file that describes the topology of a road network and the allowed junction manoeuvres on the roads. A second component, the actual simulator, simulates car movements in time using the previous network file as input. The results of the simulation, typically a trace file, can be loaded into different visualiser and analyser tools. In a realistic scenario traffic analysts wish to run several simulations to analyse the effects of changing network parameters, like traffic light patterns, one way streets or the effects of increased traffic on particular roads. They create a workflow where the results of the road network generator are fed into several simulator components, and then the outputs are sent to analyser/visualiser components. This requires parameter study like execution of several simulations and their subsequent analysis. Distribution appears at two different levels. The components of the workflow could be either sequential or parallel applications that require a computer cluster. Also, some components of the workflow can be executed parallel to each other on different Grid resources. The added value for the end-user of making the legacy applications Grid enabled:

- Workflow can be easily created where the execution of the different components is synchronised according to the workflow execution graph, and the input/output files are automatically transferred between the different sites.

- Different instances of the same application can run on remote clusters/supercomputers at the same time with different input parameters speeding up parameter studies.
- The traffic analyst can utilise simulators running on remote resources. A simple laptop/desktop is enough to run the simulation and retrieve results.
- Using a collaborative portal solution allows different traffic analysts on different locations to create workflows together each adding the component and resource she is authorised to use.

Mapping to the underlying infrastructure

1. *Grid aware or unaware application* - The components of the above described workflow were typically written as non Grid aware applications and are usually massive MPI or PVM parallel codes that run on supercomputers or clusters. The installation process of these applications can be rather demanding and they use proprietary interfaces for communication. These legacy components of the workflow have to be deployed on several Grid sites and exposed as Grid services using GEMLCA. The source codes of these legacy applications are typically not available (especially if we are talking about commercial products) resulting in a need for coarse grained black-box type wrapping.
2. *User Interface* - The user interface for workflow creation, execution and visualisation is a Grid portal, like the P-GRADE portal.
3. *Type of infrastructure used* - A grid infrastructure is required where the different components of the workflow are mapped to different clusters and/or supercomputers providing the capability of parallel execution at both job and also at workflow level.
4. *Technologies Used* - The scenario utilises GEMLCA and the P-GRADE portal. GEMLCA is currently using GT4 as underlying Grid middleware but it can be ported to any OGSA compatible service-oriented Grid architecture with reasonable effort. The P-GRADE portal is based on the GridSphere portal framework, and the workflow subsystem of the portal is implemented on top of Condor DAGMan.
5. *Service Representation* - The legacy applications are represented as Grid/Web services according to the service representation offered by the underlying Grid infrastructure (e.g. GT4 services in the current implementation).
6. *Workflow requirements (representation)* - Once the components are expressed as Grid services workflow can be created using the workflow engine of the P-GRADE portal, for example. The different components of the workflow can be mapped either statically (in the current solution) at workflow creation time, or dynamically (once GEMLCA is extended with a broker and automatic service deployment) at run-time to the available

resources. The communication between workflow components is scheduled by Condor DAGMan and executed as file transfers between the sites.

7. *Runtime requirements* - The success or failure of a particular job have to be monitored. If the execution of a job failed the user can map the execution into another resource (in the current solution), or the job is automatically migrated into another site (once the system is extended with an appropriate broker).

3.4 Grid-based e-Marketplace. (UoW, SZTAKI)

Projects: GEMLCA [7]

An e-marketplace is an internet site where potential business partners can come together in order to exchange goods and services. Although offering substantial business advantages such as reduced transaction costs, integrated processes in supply chains, shortened purchase cycles, greater transparency and lower administrative costs, e-marketplaces are still facing significant technical difficulties. Integrating legacy back-office applications and ERP (Enterprise Resource Planning) systems with marketplaces is a complex, and expensive task, but one which is necessary in order to utilise fully the opportunities of exchange sites. Today, e-marketplaces offer only limited functionality due to the difficulties in integrating existing value-added services. These services often also require large computational power such as logistics optimisation algorithms. In this scenario back-office, marketplace and third party applications are all presented as Grid services allowing the seamless integration of these applications independently of the hardware and software platforms they are running on. This provides marketplace operators to more easily integrate already existing value added services to the marketplace, and marketplace participants to more easily connect their back-office applications to marketplace services.

Mapping to the underlying infrastructure

1. *Grid aware or unaware application* - Both Marketplace and back-office applications are typically legacy components that are not Grid aware. The source codes of these legacy applications are typically not available (especially if we are talking about commercial products) resulting in a need for coarse grained black-box type wrapping provided by GEMLCA. However, the integration with specifically written Grid aware applications is also a requirement.

2. *User Interface* - The user interface is an e-marketplace portal that can be extended with Grid specific functionalities and a workflow engine.

3. *Type of infrastructure used* - A grid infrastructure is required where the different components of the workflow are mapped to different clusters and/or supercomputers providing the capability of parallel execution at both job and also at workflow level.

4. *Technologies Used* - The scenario utilises GEMLCA. GEMLCA is currently using GT4 as underlying Grid middleware but it can be ported to any OGSA compatible service-oriented Grid architecture with reasonable effort.

5. *Service Representation* - Marketplace and back-office applications are represented as Grid/Web services.

6. *Workflow requirements (representation)* - Users should be provided with the capability to create complex workflow applications from more simple building blocks. For this reason a workflow engine, like the workflow engine of the P-GRADE portal, for example, is required.

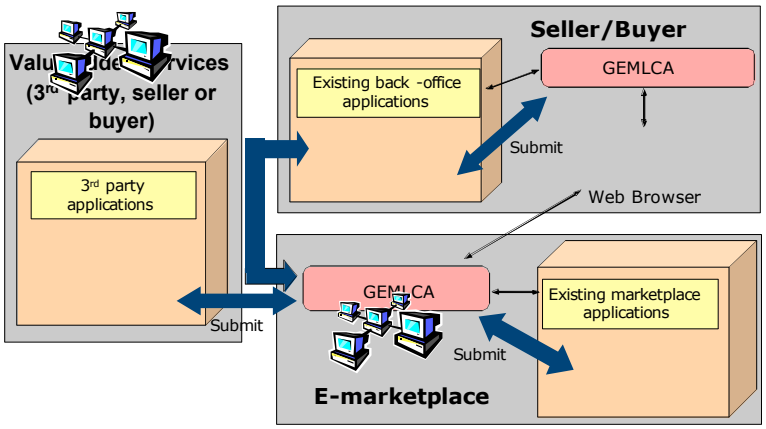

Figure 2: GEMLCA-based e-marketplace architecture

3.5 Library Level Legacy Code Wrapping. (UWC, FORTH-ICS)

Projects: WSPeer [30], REM [22]

The LAL Libraries (LSC Algorithm Library) [14] are a set core routines used in Gravitational Wave Analysis written in ANSI C89. They are typically used as components in a workflow. For example in the case of an inspiral

search (searching an inspiralling compact binary system consisting of a pair of dense, compact objects - either neutron stars or black holes - orbiting around each other) a bank of 'templates' which represent theoretically possible patterns present in the raw detector data, is initally calculated. Typically there may be several thousands of templates in a bank, and the data stretch may be several years long, sampled at 16KHz. These templates are then searched for in the raw data using a correlation. This correlation algorithm runs for each template in the bank and can be run in parallel. Furthermore the correlation itself may also be able to be broken into workflow components and likewise run in parallel. The outputs from the correlations are stored for subsequent analysis.

A physicist wants to compose a series of LAL routines into a workflow and feed raw detector data into this workflow. These routines should be remotely accessible (i.e. the libraries should not have to be installed locally), as should the - potentially large - data sets.

Mapping to the underlying infrastructure

1. *Grid aware or unaware application* - Grid-unaware
2. *User Interface* - Graphical workflow engine, e.g. Triana [27] within a portal framework with capabilities for monitoring process output. From a service provision side, the ability to wrap and deploy legacy code.
3. *Type of infrastructure used* - The infrastructure is grid based, employing a single or multiple servers that supply LAL routines as Web services, a 'template generator' service that initialises the bank of templates and a database exposed as a grid resource to store the final output.
4. *Technologies used* - WSPeer for invoking Web services, library-level legacy code wrapping technology to expose LAL function calls as Web services. FORTH-ICS has incorporated the SWIG [26] code generation tool in a prototype called REM - Remote Execution and Monitoring - that allows remote installation of C application programs, and their subsequent execution via a web browser interface. The program to be installed is packaged as a .zip file that includes C source files and an interface definition file that exposes functions for which a Java language binding is to be generated. The web browser interface allows users to initiate the execution of long-running invocations (via SOAP request/response sequences). Invocations may use data provided in files that can be uploaded by the web browser interface.
5. *Service representation* - Either Web service or WSRF [31] service
6. *Workflow requirements (representation)* - Graphical workflow engine (Triana). There is a requirement for the data to travel directly through the workflow - from component to component - rather than via the local machine. This might require technologies such as Styx Grid Services (SGS) [13] which are wrapped as Web services and allow for streaming of

large data sets, and/or WS specs such as WS-Routing [29] which specifies a path for input and output.

7. *Runtime requirements* - Monitoring of wrapped legacy codes on remote execution hosts. Using REM, it is possible to inquire about the progress of a long-running invocation, by issuing a SOAP request that results in a response that descibes the resource consumption by this invocation (metrics like elapsed time, percentage of time in user vs kernel-space, resident memory size, etc - similar to the output of the *ps* utility). Also possibly SGS could used which allow monitoring and retrieval of output files.

3.6 Dynamic Data Driven Application Scenario for Pipeline Modeling. (UWC)

Projects: Triana [27], Cactus [2]

Extraction of natural gas from oil/gas fields generally requires the presence of several oilrigs (for subsea fields) or wells, which feed gas at high pressure into a gas collection network which delivers the gas to a processing plant where it is split into components for shipping or piping to end-users, e.g. a domestic natural gas distribution system for cooking and heating.

Each source (oilrig or well) on the network produces gas of different composition, e.g. different hydrocarbon fractions and presence of impurities such as hydrogen sulphide, and pipeline operators require to know what composition to expect to deliver to the processing plant at any particular time in order to setup appropriate facilities or to control the quality of the gas delivered to the plant. This has lead to the construction of sophisticated simulation software, e.g. TGNet and RTFlow, which can use live data, such as pressure, temperature, flow-rate, and composition, from instruments on source and delivery termini of the network and model the gas flow and predict composition.

These models are of immense use to pipeline operators both for predicting composition and thus guiding them in the operation of the pipeline, but also in alerting them to significant events, such as discrepencies between predicted and measured flow, indicating a blockage or a leak in a pipeline, or the approach of a slug of gas with a particular composition.

Such systems are widely deployed in modern pipeline systems, and form a good example of a practical dynamically data-driven application scenario.

Modern component and grid based technologies have great potential to significantly enhance these systems, e.g.

- Allowing the integration of several models, such as reservoir models, pipeline flow models, pig and leak location models.
- Spawning of one or more look ahead simulations to predict what would happen in the future with the current flow-regime or any modification to it.

- Integration of agent-based technologies to trigger such a set of look-ahead simulations and present an operator with a set of options and advice.
- Easier configuration, selection and composition of the underlying models to be used.
- Integration of SMS, instant-messaging, and other technologies to alert operators of unusual or specified conditions.
- Secure remote access to monitor and control the software and the underlying pipelines.
- Combinations of these basic scenarios lead to even more exciting possibilities, such as integration of environmental, ocean, or coastal flow models to predict the dispersal and effects of oil or gas from a pipeline leak.

Note that while the above was written specifically with gas collection pipelines in mind, the same technologies are in use or may be of use to liquid or liquid-gas pipelines, and to gas, oil or water distribution networks.

Mapping to the underlying infrastructure

1. *Grid aware or unaware application* - Grid-unaware
2. *User Interface* - Advanced Portal, with intelligent resource brokers, costing QoS and workflow capabilities.
3. *Workflow requirements (representation)* -

   This application needs highly dynamic and re-configurable workflows. The workflows are highly data dependent so they should be able to be constructed on-the-fly depending on the type of analysis that is required.
4. *Runtime requirements* -

   Component-level logging and debugging capabilities. Workflow steering through human interaction to make decisions based on current state of the system or data.

3.7 Molecular Dynamics Simulation. (HLRS)

Projects: SEGL [23] [24]

The problem is to establish a general, generic molecular model that describes the substrate specificity of enzymes and predicts short- and long-range effects of mutations on structure, dynamics, and biochemical properties of the protein. A molecular system includes the enzyme, the substrate and the surrounding solvent. Multiple simulations of each enzyme-substrate combination need to be performed with ten different initial velocity distributions. To generate the model, a total of up to of 3000 (30 variants x 10 substrates x 10 velocity distributions) MD simulations must be set up, performed and analyzed.

Each simulation will typically represent 2 ns of the model and produce a trajectory output file with a size of several gigabytes, so that data storage, management and analysis become a serious challenge. Each simulation can typically require 50 processor days for each simulation. These tasks can no longer be performed interactively and therefore have to be automated.

The scientific user requires an application which is user-friendly (requires no specific programming or GRID knowledge) and can deliver and manage the required computing resources within a realistic time-scale. Such an application requires a workflow system with tools to design complex parameter studies, combined with control of job execution in a distributed computer network. Furthermore, the workflow system should help users to run experiments which will find their right direction according to a given criteria automatically.

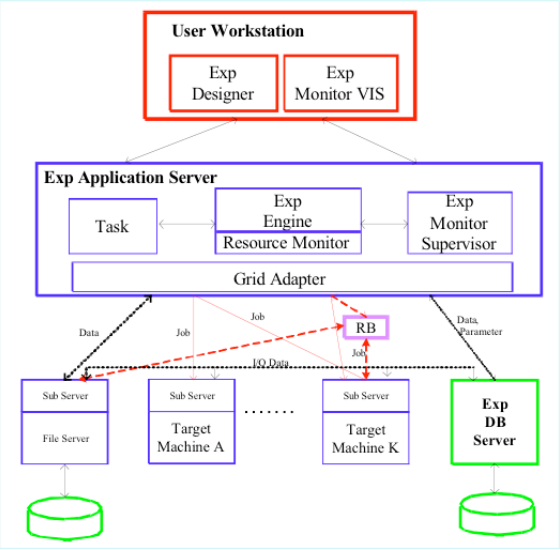

Figure 3: SEGL System Architecture

SEGL is a GRID aware application enabling the automated creation, start and monitoring of complex experiments and supports its effective execution on the GRID. The user of SEGL does not need to have knowledge of specific programming languages or knowledge about the GRID. Figure 3 shows the system architecture of the SEGL.

Mapping to the underlying infrastructure

1. *Grid aware or unaware application* - SEGL is Grid aware. The running application is Grid unaware.

2. *User Interface* - Java graphical system of icons and nested windows to represent workflow.
3. *Type of infrastructure used* - A Grid infrastructure is used.
4. *Technologies Used* - SEGL consists of three main components: the User Workstation (Client), the ExpApplicationServer (Server) and the ExpDB-Server (OODB). The system operates according to a Client-Server-Model in which the ExpApplicationServer interacts with remote target computers using a Grid Middleware Service such as UNICORE [28] and SSH. Integration with Globus [8] is planned for the future. The implementation is based on the Java 2 Platform Enterprise Edition (J2EE) [12] specification and JBOSS Application Server [11]. The database used is an Object Oriented Database (OODB) with a library tailored to the application domain of the experiment.
5. *Workflow requirements (representation)* - The application will have workflows between the calls to different codes Workflow is expressed using the GUI.
6. *Runtime requirements* - Is able to monitor the status of the applications.

3.8 GRID superscalar applications using BLAST. (UPC)

Projects: GRID Superscalar [9]

The application is algorithmically speaking simple, although complex in terms of data size and required computation time. The objective is to compare - one by one - two sets of DNA (the mouse set with the human set). To perform this comparison, both DNAs are split in several files and then each file in the mouse set is compared with each file of the human set using BLAST. The application was initially designed in Perl using specific LoadLeveler functionalities. This application does not have special requirements regarding the resource distribution. The application can be run on a cluster or on a computational grid. The input files can be mirrored in different servers or sent to each server on demand when required. Although the application may show performance reduction due to these file transfers, the cpu time consumed by the BLAST task is coarse enough to hide the transfer time (overlapping file transfers with task executions).

The original application was ported to the GRID superscalar version in c/c++ (this application is earlier for the development of the GRID superscalar Perl interface).

The use of GRID superscalar has simplified the programming of the application. the number of lines of the application has been reduced down to 10% of the original Perl application. Also, the development time was reduced by half, including the GRID superscalar learning process.

Mapping to the underlying infrastructure

1. *Grid aware or unaware application* - The original Perl application was 'cluster aware'. Most of the Perl scripts of the application were used to organize the data, monitor which tasks have finished, which were pending, queuing new tasks, etc. The scripts were based on the prologue and epilogue that can be run before/after each task submitted to a LoadLeveler queuing system. The GRID superscalar application in c/c++ is grid unaware. It can be run sequentially in a single cpu machine or exploit the task concurrency in a computational grid. The GRID superscalar runtime takes care of the file transfers, task-flow control and of all the grid related aspects.

2. *User Interface* - The user interface in the original program was a set of Perl scripts. In the final version, is a sequential program in c/c++. The call to the BLAST binary is wrapped in a function by means of the 'system' call.

3. *Type of infrastructure used* - Original program, cluster. Final version, Grid.

4. *Technologies used* - For this case, the GRID superscalar version was using Globus 2.4.

5. *Workflow requirements (representation)* - The application defines a flat set of tasks without data/control dependencies (all BLAST evaluations are independent of each other). However, potentially the user may want to add a summarising task that depends on the finalisation of all the other tasks. Therefore, the requirements for workflow representation in this application are very low.

6. *Runtime requirements* - The Biologists where interested in being able to know the progress of the application. Being able to know the number of tasks that have finished, the number of tasks being executed and the pending tasks is enough.

3.9 Chemical application - using GAMESS. (UPC)

Projects: GRID Superscalar [9]

The goal of the application in this case is the determination of molecular potential energy hypersurfaces from the results of electronic structure calculations.

This application is of interest in some areas of computational chemistry. Until recently the approach was to hire a student responsible of generating the different input data and calling the electronic structure package one by one in a desktop computer.

As can be observed, this is highly inefficient. A group of researchers of the Universidad de Castilla La Mancha (UCLM, SPAIN) has being collaborating

with the UPC during last two years in the application of Grid techniques to their research and specially to their application.

to this end, a GRID superscalar application has been developed to solve the problem of the generation of molecular potential energy hypersurfaces on a computational Grid. The process is organized in three steps.

1. Molecular structure generation: this step consists of the generation of the set of molecular structures defining the potential energy hypersurface. The result is a large number of data files in format required by the desired electronic structure package.

2. Electronic structure calculation: one evaluation with the electronic structure package is executed for each of the data files generated in step 1. Since the output of each of these evaluations is a large output file, a filtering process that obtains the required information (molecular coordinates and total energy) is applied.

3. Data integration: the data generated by each of the calculations in step 2 is integrated in a single ASCII file.

While steps 1 and 3 are not computationally intensive, the evaluation of the different data of step 2 would have taken a long time in a sequential machine.

The application does not have any special requirement regarding the distribution of the resources. Basically the data should be distributed from the client machine to the servers and the results collected at the end of the execution.

Mapping to the underlying infrastructure

1. *Grid aware or unaware application* - The application is Grid unaware.

2. *User Interface* - Sequential application in c/c++. Calls to the GAMESS binary are wrapped in a function by means of the 'system' call.

3. *Type of infrastructure used* - A Grid infrastructure is used.

4. *Technologies used* - For this case, the GRID superscalar version was using Globus 2.4.

5. *Runtime requirements* - It would be useful to have a monitoring/steering interface that allows one to observe the progress of the GAMESS executions and also to be able to interrupt some of the executions (for some input data, GAMESS may not converge and the execution time lasts much longer than average. These evaluations are of no interest because are outlayers and therefore it is desirable to stop them).

   Although a small monitoring tool has been designed by the UCLM team, the possibility of stopping outlayer evaluations is still not supported.

3.10 Performance Analysis Scenario. (UPC)

Projects: GRID Superscalar [9]

In this case the aim is to develop an automatic performance analysis tool for a large (in number of CPUS and in terms of CPU consumption) MPI applications.

The performance analysis tools developped at UPC - Paraver, Dimemas, and Paramedir - will be used together with other mathematical tools e.g. Mathematica.

The input to the application will be a Paraver tracefile of the MPI application to be analysed. The objective is to identify with an automatic process the possible performance problems of the application (loss of performance in a given CPU, loss of network performance, bottlenecks due to the data dependencies of the application, etc). It should be taken into account that the input tracefile can be huge. The application is still under development at UPC by a PhD student with a mathematical background. The application may have network bandwidth QoS requirements when visualizing shots of the Paraver file where it has been identified as a problem.

Mapping to the underlying infrastructure

1. *Grid aware or unaware application* - The application is Grid unaware.
2. *User Interface* - C/C++ sequential application with functions wrapping calls to external tools (Dimemas, paramedir,...) with the 'system' call.
3. *Type of infrastructure used* - A Grid infrastructure is used.
4. *Technologies used* - For this case, the GRID superscalar version was using Globus 2.4.
5. *Workflow requirements (representation)* - The application will have workflows between the calls to different tools. However, no special representation is initially required.
6. *Runtime requirements* - Currently not supported. However it will be useful to be able to monitor the status of the applications and include possible user interaction through a steering interface.

3.11 Hospital Information System. (UoW)

Flexible Hospital Information system to make collection of patient data, patient monitoring, allocation of staff around the hospital area, monitoring of staff and record updating (among other) easier and more efficient.

The management centre in each hospital presents the ideal entry point to the whole system and will be the head of the higher-level virtual cluster for this hospital [32]. Different virtual clusters are created for the different types of users: doctors, nurses, hospital staff, security staff and so on. Each member

of the hospital team carries a lightweight mobile device like PDA or smartphone. These devices are personalized for the specific person and have the necessary platform components installed along with a set of metadata describing the end user. Virtual clusters can also be established at the highest level between hospitals depending on the trust relationships. Note: in the rest of this text by saying cluster we mean the virtual cluster consisting of limited and mobile devices not necessarily a cluster in its traditional form.

Doctors have access to large and possibly distributed across the departments databases of patient history. Doctors can update the patient records as well as their own status locally and the main database is automatically updated pulling data in a predefined way without the need for constant reporting. In the case of integrated sensor equipment for the measurement of vital patient measurements (like pulse, pressure, temperature and so on) some devices can act as proxies facilitating a special/proprietary communication protocol to retrieve sensor information. The application to support the retrieval of data can be Grid unaware and there is no need for location and other details for the devices: the single aggregator interface that provides the collection of data can be invoked using a mirror function that will enable the collection of all data automatically. Nurses and practice doctors can have access to restricted views of the patient records, knowledge database to assist in diagnosis and more. Doctors and nurses can also ask for help from other staff, providing information (like comments, photos, graphs etc.) and the system will automatically forward the request to all staff that is experienced, skilful and available at that moment to answer the question. When an answer arrives again back at the proxy, it is automatically forwarded to the client with minimal human intervention. Applications to support staff allocation and management can also be Grid unaware as the relevant interface can be invoked with collective operation functionality in order to get locations, staff details (expertise, availability etc.), current status and other dynamic state information, according to a supplied template. This way emergency situations can be dealt with more efficiently and the overall hospital management is now easier. Failures of any kind may trigger automatic staff reallocation according to the monitoring information, in order to have the best possible coverage and fastest response to situations. Available staff that can deal with the failure are automatically informed in order to rectify the situation as soon as possible. Human intervention in the management and allocation processes is kept in a very low level, resulting in very fast responses.

On the highest level, trust relationships between the organizations define the mobility support paradigms that will be used, for example, migration of data between the sites and the more specific virtual clusters. Mobility between virtual clusters may trigger possible migrations depending on the application requirements and the policies in use. Frequent checkpoints are cached in the proxy especially when a failure is predicted. Mobility also triggers proxy-to-proxy communication in this case so that the connection can remain with the mobile node of interest unless a virtual cluster in the new domain doesnt exist in which case checkpointing and migration will be used, again depending on the applied policies. Client applications can access resources and services in a trans-

parent way not generally concerned with possible failure and location details.

Mapping to the underlying infrastructure

1. *Grid aware or unaware application* - There is grid unawareness for the application developer as the complexity of the virtual clusters is hidden behind the proxy. Possible distribution of job can be done automatically (assuming a suitable parallel/distributed programming model); interfaces for mirrored and/or parallel execution are automatically made available; failures are detected internally and rectified immediately when this is possible providing the illusion of a relatively reliable and stable environment even though the underlying core fabric layer is not really reliable but rather is error-prone; critical data can be forwarded to a proxy cache in case of a failure prediction or in predefined time intervals.

2. *User Interface* - The virtual cluster administrator interface consists of local as well as remote secure visual clients that make management of aggregated services easy. Client user interface may consist of typical Grid portals and tools for job submission, service invocation, job monitoring and management.

3. *Type of infrastructure used* - Main infrastructure (if it exists) is a traditional Grid system. Behind the proxy a virtual cluster of possibly mobile and wireless devices like PDAs, smart-phones, laptops, and possibly sensors (with an extra support for sensor connectivity and retrieving sensor data using proxies). The whole virtual cluster is only viewed as a one-hop extension to the Grid complete abstraction.

   A number of virtual clusters are created depending on the different user requirements, credentials, service level agreements etc. Virtual clusters may well be deployed in a hierarchical manner allowing for federations of clusters, depending again on the usage, trust and security policies.

   Supporting the virtual cluster middleware, a component based core platform as specified by the CoreGRID component model and task WP7.1. The platform is reconfigurable and adaptable to changes in the execution environment.

4. *Service representation* - Virtual services that aggregate the underlying resources may be exposed either as Web Services, WSRF Services, RMI active objects or following the component model specified by CoreGRID. Services behind the proxy are exposed through component interfaces. Optionally they can be exposed as Web Services deployed in lightweight WS frameworks specially designed for limited devices (this is difficult as enveloping XML request in SOAP can be expensive and not very efficient in this case).

5. *Workflow requirements (representation)* - Dynamic, distributed workflows that may access many distributed data sources and cross many VO boundaries.

6. *Runtime requirements* - Monitoring is based on the monitoring framework of our platform, consisting of installed agents in the cluster devices to monitor and report on all possible aspects like hardware, energy, software, dependencies, open connections and so on. Lower level steering is dealt with by the failure recovery and prediction system that can predict, detect, diagnose and mask failures internal in the cluster, provide checkpointing and migration support as well as logging and reporting. Application steering is provided by the underlying components based platform. The platform is adaptable and reconfigurable so as to respond to higher-level changes in application requirements by reconfiguring application composition and optimizing the components as well as the composition itself.

3.12 Legacy Systems as Grid Services. (CYFRONET)

Projects: Legacy-to-Grid Adaptation Framework [15]

In some scenarios, at some point of computation access to an external system such as a database, an external analysis system, a monitoring system, etc., is necessary, for example, to fetch input, store output or perform analysis of collected information. Examples of such scenarios are the Gravitational Wave Analysis (section 3.5) where the final output is stored in a database, the Molecular Dynamics Scenario 3.7 in which simulations generate huge amounts of data to be stored in databases, the Hospital Information System 3.11 where information about patients are stored in large databases and accessed or updated when necessary. To enable flexible construction of workflows in such cases, legacy systems need to be exposed as grid services. This can be achieved using the LGF Legacy-to-Grid Adaptation Framework. Using LGF, with minimum assistance from the developer, a legacy system, such as database, can be deployed as a web/grid service and accessed through a WS-interface. In case of databases, it is important that the WS-interface for the database supports transactional interaction with a database. LGF supports transactional processing by exposing a standard set of WS-methods for transactional interactions if this is requested. Once a WSDL description for this interface is available and the service is deployed, the integration with a portal (in the sense of composing the legacy system into a workflow) will be straightforward.

Mapping to the underlying infrastructure

1. *Grid aware or unaware application* - legacy system usually unaware.
2. *User Interface* - workflow composition tool from a grid portal.
3. *Type of infrastructure used* - Grid.
4. *Service Representation* - web service or WSRF

4 From User Requirement to Portal Design

In simple terms, a Grid Portal is a Web interface to the Grid. It provides a (secure) Web-based user interface to a collection of tools and services, which simplify how a user interacts with Grid primitives. Most current portal implementations perform relatively primitive Grid capabilities, that is they are essentially execution and monitoring environments. Core portal capabilities typically include environment management, such as: authentication, profile personalization, administration of the creation of users and groups, and Grid functionality, such as: job submission, file management, information services, remote file browsing and job monitoring services. From the user scenarios above, we can see a wide diversity and complexity of requirements not currently supported my most portals. We believe therefore that portals must support capabilities usually associated with Problem Solving Environments. This development will require more sophisticated user interfaces. From a non-functional perspective, it is required that these interfaces should conform to ISO standards for HCI and usability.

We consider the following list of requirements that arise from the user scenarios particularly important for next-generation portals to address. The scenarios associated with a given requirement are given in the table below. These scenarios may or may not have existing solutions for the associated requirement. For brevity, the scenarios, where possible, are referenced by project name. The numbers next to the scenario titles represent the section of this document in which they appear. Where the association between scenario and requirement is not explicit, a superscript number is given which refers to the list of brief clarifications below the table.

Workflow execution and representation	GEMLCA/P-Grade Portal(3.2, 3.3) WSPeer/REM(3.5) Triana/Cactus(3.6) SEGL(3.7)
Cross VO integration	GEMLCA/P-Grade Portal(3.3)[1] GEMLCA(3.4)[2] Triana/Cactus(3.6)[3] Hospital Info System(3.11)
Middleware Flexibility-Choice Insulation from middleware	Proactive(3.1)[4] Triana/Cactus(3.6)[5]
Dynamic Steering	ProActive(3.1) SEGL(3.7) Grid Superscalar(3.9)
Dynamic Deployment	ProActive(3.1) Triana/Cactus(3.6) Hospital Info System(3.11)[6]

1. implicitly though collaborative workflow
2. implicitly through Internet scale e-Marketplace

3. implicitly through dynamic creation of workflow - i.e. search and discovery of resources
4. implicitly through the ProActive deployment infrastructure
5. implicitly through dynamic creation of workflow - i.e. search and discovery of resources
6. implicitly through migration

We elucidate each requirement below and offer some suggestions on how research efforts and further collaborations within CoreGrid can help address them.

4.1 Workflow Execution and Representation

Most portal implementations do not have the capacity to represent or execute workflow. The collaboration between GEMLCA and the P-Grade portal is a notable exception. The research and lessons learnt in this collaboration need to feed into the next stage of portal design. This process should be conducted in line with developments in the GGF Workflow Management Research Group to ensure that the design specification follows emerging standards.

4.2 Cross VO integration

Most Grid activity currently operates within single VOs. This will have to change if the vision of the Grid is to become a reality. However there are a number of issues which arise from this, for example security and resource management. SZTAKI has been conducting research into the latter (cross VO resource management). This work needs to be considered. Likewise research into more dynamic security models needs to be examined and conducted. The current GSI is too restrictive and static for some of the scenarios listed in Section 3, for example the scenario described in 3.6 which requires dynamic deployment across grids, and the scenario in 3.11 which requires security across mobile devices without static IP addresses.

4.3 Middleware Flexibility/Choice - Insulation from Middleware

Most portals contain bindings to a specific middleware tools, e.g. Condor [3], Globus, etc but some of our user scenarios require the flexibility of switching or choosing the type of middleware depending on the particular task or application. We therefore need to bridge this missing link between the application level and the various grid middleware packages through the use of an application-oriented interface to abstract the portal from needing to adjust its software to provide implementation to changing grid resources, and middleware packages.

One example of such an interface is the GAT [6], which is designed to be easy to use for application developers, supports different programming languages and grid middleware, and is oriented towards dynamic and adaptive grid-aware applications. The developers of the GAT (GridLab [10]) have also been a key driver behind the Simple API for Grid Applications (SAGA) group, a GGF standardization activity, to access a range grid capabilities, such as job submission, file movement and registry of files in logical file servers.

4.4 Dynamic Steering

Real-world Dynamic Data-Driven Application Systems (DDDAS) are pervasive in the modern world. Applications range from industrial processes such as chemical plants or pipeline networks, engine modelling and control, through to scientific data analysis tasks within complex astrophysics experiments. DDDAS systems, such as our pipeline modeling scenario, described in Section 3.6, require a flexible infrastructure to be able to make on-the-fly decisions about workflow and the reconfiguration of components. Future generation portals need to address these aspects to be able to allow the re-configurability of workflow and components based on certain properties, typically from the current data content. Further, users will need more interactivity with the executing components or workflow so that they can steer its direction based on expert knowledge i.e. the user will need to form part of that workflow. This area is particularly open to more research.

4.5 Dynamic Deployment

A major issue arising from the scenarios is the question of dynamic deployment. This raises issues again of security, and what constitutes a grid node (i.e. what is expected of in terms of operating system, configuration etc). Work undertaken at UPC as well as new developments at FORTH-ICS, which allow the bundling and deployment of legacy codes need to be considered in the portal design. INRIA also have some experience with deploying Java code dynamically. Furthermore UoW have begun work on defining a dynamic deployment strategy. This area is as yet not subject to any standarisation. Therefore these various approaches developed within CoreGrid institutes need to be analysed for similarities and differences to enable the development of a generic dynamic deployment interface. This may in itself lead to proposals regarding standardisation.

4.6 Portal Design

Figure 4 shows the proposed technology stack of the next-generation portal. It is designed to integrate with the generic component platform architecture as defined in CoreGRID deliverable D.ETS.02. The primary gateway for the portal's communication with Grid components is through the CoreGRID Runtime Environment (here referred to as CRE) proposed in the generic component platform

(D.ETS.02). The stack addresses the advanced issues described in this section, either directly or through the CRE. In particular the issues of security, steering and insulation from middleware are handled as part of the CRE.

The top layer defines the current portal capabilities - user log-in, job submission and monitoring. Integrated into this layer is the workflow component. The portal acts as an interface to the layers below, in particular the CRE, for simple, non-workflow based operations as well as more complex scenarios which make use of the workflow components.

The workflow sub-component is itself constructed of:

1. Workflow editor - this component consists of graphical elements that as a whole can be integrated into the web-based portal user-interface.

2. Workflow language - this component defines the workflow. The workflow is likely to be defined in XML. There are a number languages in existence - all are similar although some are more control-oriented, and others are more data-oriented. While the Grid community may arrive at a standard workflow language, this has not happened at the time of writing. Therefore, this component should also contain a translation module to handle multiple definition languages.

3. Enactment engine. This component does the actual work of handling the workflow. It must communicate this enactment back to the portal in order for the portal to pass data to the CRE discussed below.

The CRE acts as the glue between various underlying middleware capabilities. As shown in figure 4, a steering interface and security context are explicitly defined by the runtime environment which the portal will make use of. For example, in handling cross VO workflow enactment, the portal will be able to use the security context of the runtime environment to prepare certificate proxies representing identities mapping to different VOs. Because this security context is part of the runtime environment, it sits between the top layers and the underlying middleware allowing messaging from the portal to the underlying middleware to pass through this security sub-component. Depending on the middleware target, this sub-component may either perform actual security related operations such as handshaking or encryption, or may be entirely bypassed by the runtime environment. This approach allows security overhead to only be added when required or requested.

The integration of the steering component into the CRE will give the portal direct access to these capabilities. Therefore the portal needs to support this functionality both directly in terms of user requests and in terms of dynamic changes to the workflow sub-components.

Insulation from middleware implementations is provided by the CRE. This includes the differing technologies used to expose capabilities such as service oriented representations and component oriented representations as well as functionality related to network characteristics such as node reliability and topology.

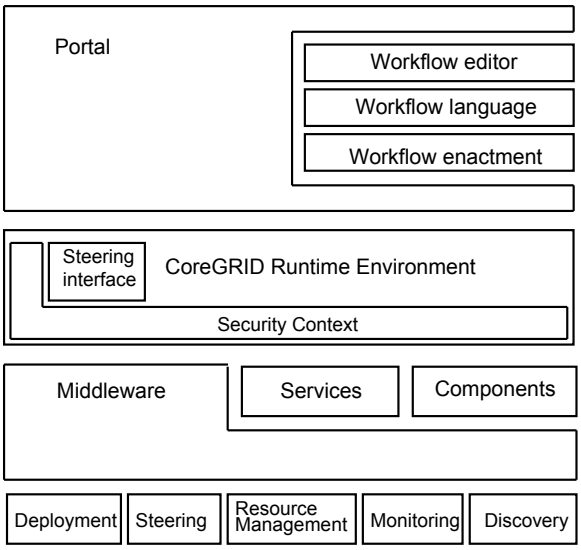

Figure 4: Portal technology stack

The latter includes middleware that handles such things as service discovery and resource management.

Not all of the requirements defined in this document are directly addressed by the portal layer of the technology stack. Rather, we view the portal within the context of the CoreGRID generic component platform (D.ETS.02). As such the portal inherits some of its capabilities from the CRE. For example insulation from middleware diversity and security are handled by the CRE. The area we believe is of greatest importance from the portal perspective and requires specific extensions to current portal capabilities, is workflow enactment. Workflow in brings with it further complexities for steering, deployment and the handling of cross VO interactions as the scenarios described in this document show.

5 Conclusion

In this report we have defined a taxonomy for evaluating portal capabilities. This taxonomy was then applied to user scenarios. Certain areas lacking in current portal implementations which are integral to the user scenarios, were then defined. We discussed how these capabilities might be developed through integration of the skill sets and research projects amongst CoreGrid institutes. Finally we define a technology stack for the portal that addresses how these advanced capabilities are to be addressed in the portal design.

References

[1] BLAST - Basic Local Alignment Search Tool, see *http://www.sp.uconn.edu/ mcb232vc/blast.html*

[2] Cactus see *http://www.cactuscode.org/*

[3] The Condor Project see *http://www.cs.wisc.edu/pkilab/condor/*

[4] EGEE GLite see *http://glite.web.cern.ch/glite/*

[5] GAMESS - The General Atomic and Molecular Electronic Structure System, see *http://www.msg.ameslab.gov/GAMESS/GAMESS.html*

[6] Gabrielle Allen, Kelly Davis, Tom Goodale, Andrei Hutanu, Hartmut Kaiser, Thilo Kielmann, Andre Merzky, Rob van Nieuwpoort, Alexander Reinefeld, Florian Schintke, Thorsten Schütt, Ed Seidel, Brygg Ullmer. The Grid Application Toolkit: Towards Generic and Easy Application Programming Interfaces for the Grid. *Proceedings of the IEEE*, Vol. 93, No. 3, pp. 534–550, March 2005.

[7] GEMLCA: Grid Execution Management for Legacy Code Architecture, see *http://www.cpc.wmin.ac.uk/gemlca/*

[8] The Globus Alliance url:http://www.globus.org

[9] Rosa M. Badia, Jesús Labarta, Raül Sirvent, Josep M. Pérez, José M. Cela and Rogeli Grima, Programming Grid Applications with GRID Superscalar *Journal of Grid Computing*, Vol. 1, No. 2, 2003, pp. 151 - 170.

[10] Allen G, Davis K, Dolkas K, Doulamis N, Goodale T, Kielmann T, Merzky A, Nabrzyski J, Pukacki J, Radke T, Russell M, Seidel E, Shalf J and Taylor I (2003). Enabling Applications on theGrid: A GridLab Overview, *JHPCA Special issue on Grid Computing: Infrastructure and Applications*, August 2003.

[11] JBOSS Application Server see *http://www.jboss.com/products/jbossas*

[12] J2EE - Java 2 Platform Enterprise Editionsee *http://java.sun.com/j2ee/index.jsp*

[13] JStyx - Reading e-Science Centre, UK, see *http://jstyx.sourceforge.net*

[14] LAL - LSC Algorithm Library, see *http://www.lsc-group.phys.uwm.edu/daswg/projects/lal.html*

[15] B. Balis, M. Bubak, and M. Wegiel. A Solution for Adaptating Legacy Code as Web Services. In Proc. Workshop on Component Models and Systems for Grid Applications. 18th Annual ACM International Conference on Supercomputing, Saint-Malo, France, July 2004. Kluwer. see *http://www.icsr.agh.edu.pl/lgf*

[16] MEANDER - The MEsoscale Analysis Nowcasting and DEcision Routines, A. Horvath: Nowcasting System of the Hungarian Meteorological Servce, Fifth European Conference on Applications of Meterorology, OMSZ Budapest, 2001, pp.13.

[17] NCBI National Center for Biotechnology Information see *http://www.ncbi.nlm.nih.gov/*

[18] OASIS Open see *http://www.oasis-open.org*

[19] Foster I, Kesselman C, Nick J and Tuecke S (2002) The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, see *http://www.globus.org/research/papers/ogsa.pdf*

[20] P-Grade Portal , see *http://www.lpds.sztaki.hu/pgportal/*

[21] ProActive, see *http://www-sop.inria.fr/oasis/ProActive/* (2002) The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, see *http://www.globus.org/research/papers/ogsa.pdf*

[22] REM - Remote Execution and Monitoring *http://carv.ics.forth.gr*

[23] Natalia Currle-Linde, Fabian Bs, Peggy Lindner , Jurgen Pleiss, Michael M. Resch , A Management System for Complex Parameter Studies and Experiments in Grid Computing,accepted for publication PDCS 2004 - International Conference on Parallel and Distributed Computing and Systems', MIT Cambridge, MA, 2004.

[24] Natalia Currle-Linde, Uwe Kuester, Michael M. Resch, Benedetto Risio: Science Experimental Grid Laboratory (SEGL) Dynamic Parameter Study in Distributed Systems, accepted for ParCo -Parallel Computing, Spain 2005.

[25] GridEngine see *http://gridengine.sunsource.net/*

[26] SWIG - Simple Wrapper Interface Generator see *http://www.swig.org/*

[27] Triana, see *http://www.trianacode.org*

[28] UNICORE see *http://www.unicore.org/*

[29] WS-Routing see *msdn.microsoft.com/library/ en-us/dnglobspec/html/ws-routing.asp*

[30] WSPeer, see *http://www.wspeer.org/*

[31] The WS-Resource Framework home page, see *http://www.globus.org/wsrf/*

[32] Isaiadis, S. and Getov, V., A Lightweight Platform for Integration of Mobile Devices into Pervasive Grids, in L.T.Yang et al. (Eds.): 1st Internationl Conference on High Performance Computing and Communications, Sorrento, Italy, Sep 2005, LNCS 3726, pp. 1058-1063, Springer-Verlag, 2005