A Grid Software for Virtual Eye Surgery Based on Globus 4 and gLite

Károly Bósa, Wolfgang Schreiner
Research Institute for Symbolic Computation (RISC), Johannes Kepler University
FirstName.LastName@risc.uni-linz.ac.at

Michael Buchberger, Thomas Kaltofen
Department for Medical Informatics
Upper Austrian Research (UAR)
FirstName.LastName@uar.at

Abstract

"Grid-Enabled SEE++" is based on the SEE++ software system for the biomechanical simulation of the human eye. "Grid-Enabled SEE++" extends SEE++ in several steps in order to develop an efficient grid-based tool for "Evidence Based Medicine", which supports surgeons in choosing optimal surgery techniques for the treatment of certain eye motility disorders.

Recently, we refined the design of "Grid-Enabled SEE++" and we worked on an extended version of the software, which is able to utilize the "Web Service Resource Framework" *architecture of the* Globus Toolkit 4. *Since we met with some limitations of Globus 4, we also designed a version of "Grid-Enabled SEE++" compatible with the* gLite *grid middleware.*

In this paper, we report on our experience of porting a grid application to Globus 4 and gLite, describe the problems we encountered and discuss possible solution strategies. This may assist the porting of other applications to the grid using these middleware products.

1 Introduction

"Grid-Enabled SEE++" is based on the SEE++ [4, 10, 14] software system for the biomechanical 3D simulation of the human eye and its muscles. SEE++ simulates the common eye muscle surgery techniques in a graphic interactive way that is familiar to an experienced surgeon (see Figure 1).

SEE++ deals with the support of diagnosis and treatment of *strabismus*, which is the common name given to usually persistent or regularly occurring misalignment of the eyes where eyes point in different directions such that a person may see double images. SEE++ is able to simulate the result of the *Hess-Lancaster test*, from which the reason for the pathological situation of the patient can be estimated.

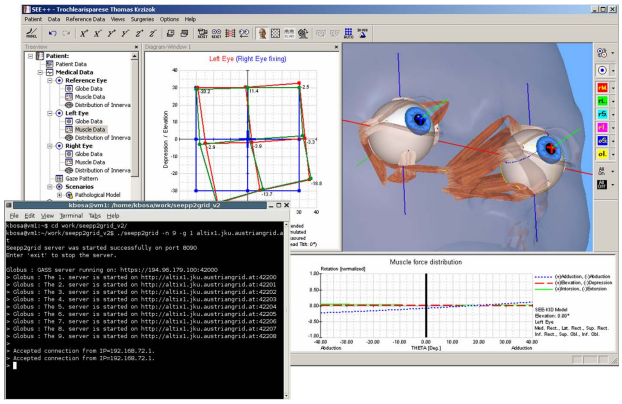


Figure 1. The Output of the "SEE++ to Grid Bridge" and the GUI of SEE++

The outcome of such an examination consists of two *gaze patterns* of blue points and of red points respectively (see the diagrams on Figure 2). The blue points represent the image seen by one eye and the red points the image seen by the simulated other eye; in a pathological situation there is a deviation between the blue and the red points. The default gaze pattern that is calculated from the patient's eye data by SEE++ contains 9 points. Bigger gaze patterns with 21 and 45 are possible and provide more precise results for the decision support in case of some pathologies, but their calculations are more time consuming.

It is also possible to give the measured gaze pattern of a patient as input. In this case, SEE++ takes some default or estimated eye data and modifies a subset of them until the calculated gaze pattern of the simulated eye (red points) matches the measured gaze pattern (green points). This procedure is called *pathology fitting*.

Strabismus can be rarely corrected sufficiently after the first surgical treatment. One of the main goals of the SEE++ software system is to give support to make the treatment of strabismus easier and more efficient. Still the doctors have

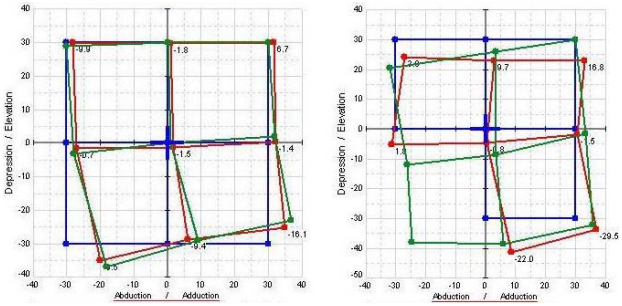


Figure 2. Gaze Patterns in SEE++: Intended (blue), Measured (green) and Simulated (red)

to spend lots of time with changing the eye parameters by a manual trial and error method and waiting for the results. The current pathology fitting algorithm is time consuming (it runs several minutes) and gives only a more or less precise estimation for the pathology of the patient. Doctors want to see quickly the results from such a decision support system, but for reaching adequate response times it is not sufficient to use only local computational power. For this, some large-scalable distributed resource would be appropriate, that provides the ability to perform higher throughput computing by taking advantage of many networked computers; such as the *grid* is.

The goal of "Grid-Enabled SEE++" is to adapt and to extend SEE++ in several steps and to develop an efficient grid-based tool for "Evidence Based Medicine", which supports the surgeons in choosing optimal surgery techniques for the treatments of different syndromes of strabismus.

Recently, we refined the design of "Grid-Enabled SEE++" and we started to work on an extended version of the software, which is able to utilize the *"Web Service Resource Framework"* architecture of the *Globus Toolkit 4*, see Section 3. Since we joined the *"Enabling Grids for EsciencE 2" (EGEE2)* project [5], we designed a version of "Grid-Enabled SEE++" compatible with the *gLite* [6] middleware used in EGEE2, see Section 4. According to this new design, we intend to further develop the "Grid-Enabled SEE++" software system on the basis of the higher services of the EGEE2 middleware (compared with the low-level services of the Globus Toolkit).

2 Former Results

In [3], we combined the SEE++ software with the Globus (pre-Web Service) middleware [7] and developed a parallel version of the simulation of the *Hess-Lancaster test*, see Section 2.1.

Furthermore, we reported the prototype implementation of a medical database component for "Grid-Enabled SEE++", which is going to be used for storing and sorting patient data with gaze patterns and eye data, see Section 2.2.

Finally, we designed a so called grid-based *Pathology Fitting* algorithm, which would be able to determinate (or at least estimate) automatically the pathological reason of a patient's strabismus, see Section 2.3.

2.1 Parallel Gaze Pattern Calculation

The initial component of "Grid-Enabled SEE++" is the "SEE++ to Grid Bridge" [3], via which the unchanged SEE++ client can get access to the infrastructure of the Austrian Grid [1]. The "SEE++ to Grid Bridge" acts as a SEE++ server to the SEE++ clients and as a Globus client to the Grid. The usage of grid resources is completely transparent to the clients. When the "SEE++ to Grid Bridge" is executed, it starts SEE++ server processes on one (or perhaps more) grid site(s) and then waits for computational tasks from its clients.

The "SEE++ to Grid Bridge" is able to split gaze pattern calculation requests of clients into subtasks (which contain only some gaze points of the original pattern) and to distribute them among the servers (data parallelism). Since the calculations of each gaze points is completely independent from each other, there is no communication among the server processes. By this, we demonstrated how a noticeable speedup can be achieved in SEE++ by the exploitation of the computational power of the Austrian Grid.

In benchmarks (see Figure 3), we have investigated the effectiveness of our grid-parallel solution in different situations where 1, 3, 9, 25, 30 or 45 processors were used on the grid (one SEE++ server process was started on each processor).

Each value in Figure 3(a) depicts the average execution time of 5-7 computations of 45 gaze points. The test cases were executed on the Austrian Grid site `altix1.jku.austriangrid.at`, which contains 64 Intel Itanium processors (1.4GHz).

In case of 25 or more processors, we speeded up the simulation of the Hess-Lancaster test by a factor of 14-17, see Figure 3(b). If we started and used some processes on the far grid site `altix1.uibk.ac.at` in Innsbruck as well, we got slightly worse results with the speedup limited to a factor of 12-14.

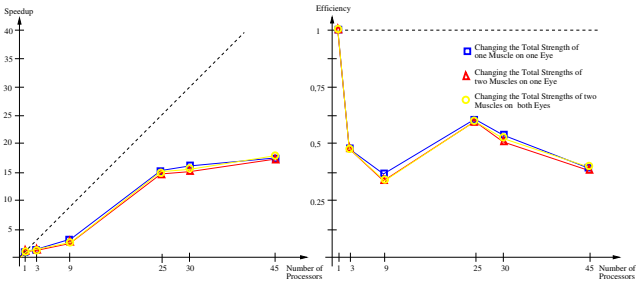
2.2 A Grid-Enabled Medical Database

As a starting point for the goal of developing a grid-enabled database for SEE++, a database was designed [12] and prototyped as a Web Service application.

Since the "Grid-Enabled SEE++" database is designed for storing patient records, security is a very important aspect. The security implementation ensures that every

Machine Name	altix1.jku.austriangrid.at						altix1.jku.austriangrid.at altix1.uibk.ac.at		
Number of Processors	1	3	9	25	30	45	25	30	45
Changing the Total Strengths of one Muscle on one Eye	25.27s	17.44s	7.58s	1.65s	1.57s	1.43s	1.87s	1.80s	1.71s
Changing the Total Strengths of two Muscles on one Eye	27.18s	18.81s	9.11s	1.82s	1.78s	1.57s	2.01s	1.96s	1.88s
Changing the Total Strengths of two Muscles on both Eyes	28.68s	20.04s	9.80s	1.90s	1.85s	1.59s	2.09s	2.03s	1.92s

(a) Execution Times



(b) Speedup and Efficiency Diagrams

Figure 3. Benchmark Results for Gaze Pattern Calculations with 45 Points

Web Service call is secured appropriately by checking the caller's identity. Furthermore, the access layer employs many techniques to maximize security (e.g.: supporting username/password based authentication; applying strong encryption of stored user passwords with a SHA-512 salted hash code).

Our next steps concentrate on developing a distributed grid-enabled database system that allows "Grid-Enabled SEE++" to give efficient support to "Evidence Based Medicine".

To establish this proposed grid-based database without any major modification in the existing data access layer, an abstraction layer has to be introduced. Therefore, the proposed grid database will be based either on the *Grid Semantic Data Access Middleware* (G-SDAM) [8] developed by the Institute for Applied Knowledge Processing (FAW) or on the "Web Service Resource Framework" integrated in Globus 4.

As for the first possibility, G-SDAM is an open and easy extensible grid-based software system focusing on seamless data access. It is developed as a standalone grid middleware, which does not require any underlying software layer (like Globus), however it takes over and uses the applications Grid CA and GridFTP developed by the Globus project.

2.3 Design of Grid-Based Pathology Fitting

Pathology Fitting is essentially a non-linear optimization problem in a multidimensional parameter space, where a subset of the patient's eye model parameters is modified until the calculated gaze pattern matches the measured one.

Unfortunately, a gaze pattern does not uniquely determine the values of eye model parameters. Furthermore, the gaze patterns cannot be measured with perfect precision, hence, the simulated gaze patterns cannot be completely the same as the measured one (see the differences between the green and the red patterns in Figure 2). At the moment, we use a heuristic that is able to exclude most of the pathologically irrelevant solutions (solutions which are possible in the mathematical model, but cannot occur in a real human eye) and give an approximation of the correct solution.

We have already extended the pathology fitting component of SEE++ by parallel gaze pattern calculation, since a pathology fitting process often requires the calculation of approx. 60-100 gaze patterns. The speedup achieved by this implementation was limited to a factor of two, because the gaze pattern calculations are triggered by consecutive optimization steps.

Since a gaze pattern does not uniquely determine a simulation model and the current algorithm may not find always the best solution (despite of the introduced heuristic, the quality of outcome still depends on the initial estimation for the current pathological case), we can exploit the grid infrastructure to attempt to find better solutions:

1. by searching in the database concurrently for similar cases as the one presented to the pathology fitter and
2. by starting concurrent pathology fitting processes with these cases as the starting points of the optimizations (parameter study).

The computed results will then be stored in the database as feedback for providing better and better initial estimations for later computations.

3 New Architecture based on Globus 4

Based on the existing implementation of "Grid-Enabled SEE++" which uses the pre-Web Service features of Globus, we ported the software to the "Web Service

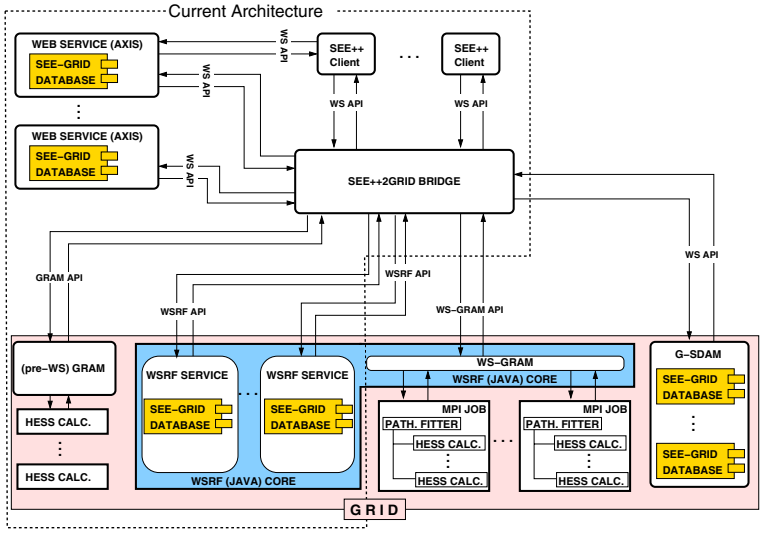


Figure 4. The Extended Architecture of "Grid-Enabled SEE++" based on Globus Toolkit 4

Resource Framework" (WSRF) supported by the newer Globus Toolkit 4.

The new architecture (the box in Figure 4 bordered by the dashed line) consists of the original Web Service based database services, new WSRF-based database services, the "SEE++ to Grid Bridge", the grid-enabled SEE++ servers (which are started via pre-WS GRAM and perform the gaze pattern calculations) and the SEE++ clients.

According to this picture, the SEE++ Clients can connect to all other components located on the grid via the "SEE++ to Grid Bridge" (the underlying grid-based infrastructure is hidden from the clients). Furthermore, the clients can also reach every Web Service based database component via the bridge, although a client is also able to interact only with one such database directly.

In the following, we report on the experience we gained on developing the WSRF interface of the SEE++ medical database and making use of the WS-GRAM service of Globus Tookit 4 for starting up the SEE++ server processes on the grid.

3.1 Experiences with the Prototype of the WSRF-based SEE++ Medical Database

We finished the elementary integration of the prototype Web Service based implementation of the SEE++ database into the WSRF framework. At the moment, the only differences between the two software components are the interfaces via which they connect to the differing underlying infrastructures (Web Services vs.WSRF).

We also kept the originally introduced security concept [12], which was designed for the specific requirements of SEE++. Later, we will combine these authorization and authentication mechanisms with the grid style transport level security and certifications for the interactions of the bridge and of the WSRF services (the communication between the bridge and the clients take place outside of the grid).

When we have started to extend the "SEE++ to Grid Bridge" with the client side functionality for this database service, we discovered some strong restrictions of the current WSRF framework existing in Globus 4. Since the "SEE++ to Grid Bridge" is implemented in C/C++, we should apply the APIs of the "Globus C WS Core" for implementing the client side functionality of our database service. For this, we should generate client side stub files from the interface file of the SEE++ medical database with `globus-wsrf-cgen` tool.

However, we notice, that this tool only generates ANSI-C bindings, C++ bindings are not supported. The solution for this problem is not easy, since we employed a very complex data structure of the mathematical eye model as arguments in the SOAP messages, whose bindings were implemented in C++ on the "SEE++ to Grid Bridge" (whose implementation was taken from the original SEE++ software). Therefore, we must implement a conversion between the generated ANSI-C bindings and the already applied C++ data structure (since we do not want to re-implement the whole "SEE++ to Grid Bridge" based on only the ANSI-C data structure).

We were most surprised, when we checked the generated ANSI C stubs. Most of the data structure was omitted from the generated C file. After a little while, we realized that

Number of Server Processes	1	3	9	25	30	45
Submission via (pre-WS) GRAM	0,85s	0,92s	0,98s	1,06s	1,09s	1,15s
Submission via WS-GRAM	7,5s	8s	9s	13s	14s	18s

Table 1. Comparative Benchmark of the Submission of SEE++ Server Processes via pre-WS GRAM and WS-GRAM

implementation of the data structure called *"SOAP Encoded Array"* [11] is completely missing from the C WS Core of Globus. Since our applied eye model strongly depends on this data type, we cannot apply the WSRF framework in our software system.

It also does not make sense to re-implement the whole eye model without SOAP arrays, because the development of "Grid-Enabled SEE++" would diverge from the development line of the original SEE++, which could lead to serious incompatibility problems between future versions of these software packages. Consequently the WSRF integration of the SEE++ database is stalled at the moment.

3.2 A WS-GRAM Compatible Extension

We also extended the "SEE++ to Grid Bridge" with the WS-GRAM Client C API. The scenario of the parallel simulation of Hess-Lancaster is similar as before. Before the bridge accepts the computational requests from the SEE++ clients, it submits in advance some grid-enabled SEE++ servers into the grid. These processes behave as some kind of "executer" programs for the computation tasks such that the remarkable latencies of the job submissions for the computational requests can be avoided (since the parallel Hess-Lancaster test simulation takes only approximately 1 up to 15 seconds on the grid).

Nevertheless, we found the same problem in WS-GRAM as in the pre-WS services, namely how to send back the contact information of the started server/executer processes to the "SEE++ to Grid Bridge". Therefore, we applied the same solution as previously in case of the pre-WS GRAM. According to this, such a process started on a grid node forks itself after it allocated a port number and terminates. By this approach, WS-GRAM perceives the termination of the program and checks whether there is any predefined *fileStageOut* procedure. We can use this procedure to send back the output files to the "SEE++ to Grid Bridge", while the forked process still runs on the grid and waits for connection requests.

Unfortunately, it is quite easy to confuse the local resource management applications, if a job spawns/forks a child process and terminates. In this case, the local schedulers may consider that the job is finished, while the child process is still active. This may induce some problems, because a batch queueing system assumes the resource is free and may assign some other jobs to it or it may kill the forked process (in order to clean up).

In order to avoid the mentioned difficulties which this *"fork and exit"* method may cause, we applied a simple technique outlined in [13]. According to this, we created a session identifier for the forked process, such that a local batch scheduler, like *Globus fork-jobmanager* or *OpenPBS* can track the job. However in some other kind of resource manager systems (e.g.: Condor), this may still not overcome these problems.

We have compared the overheads of the submissions of our SEE++ server processes via pre-WS and WS-GRAM in different situations where 1, 3, 9, 25, 30 or 45 processes were executed on the grid.

Each value in Table 1 depicts the average job submission time of 5-7 computations. The test cases were executed on the Austrian Grid site `altix1.jku.austriangrid.at`. The "SEE++ to Grid Bridge" and SEE++ clients were again executed at the RISC Institute located in Hagenberg.

From the measured values contained in Table 1, we can see obvious differences among the overheads of the job submissions in pre-Web Service and Web Service Architectures of Globus, i.e. the latter is an order of magnitude slower. One reason for this is certainly the more robust architecture and the new and more sophisticated services. Another reason may be that the WSRF-based services were implemented in Java (on the top of the Java WS Core) and not in C.

We hope that in the future versions of Globus the C WS Core will be also extended to those Web Service-based services and features which have already been in the Java part, because this could cause an appreciable improvement in the performance of the Globus Toolkit. At the moment, the WS interface is due to this performance bottleneck and the limitations described in Section 3.1 hardly useable for our purposes.

4 A gLite Port of the Architecture

In the frame of the EGEE2 [5] project, we are currently developing a "Grid-Enabled SEE++" version that is compatible with the new gLite 3.0.0 grid middleware [6].

4.1 The Initial Version

Our first step was to migrate and integrate the already existing components of our software system, like parallel gaze pattern calculation, into the gLite architecture (see Figure 5).

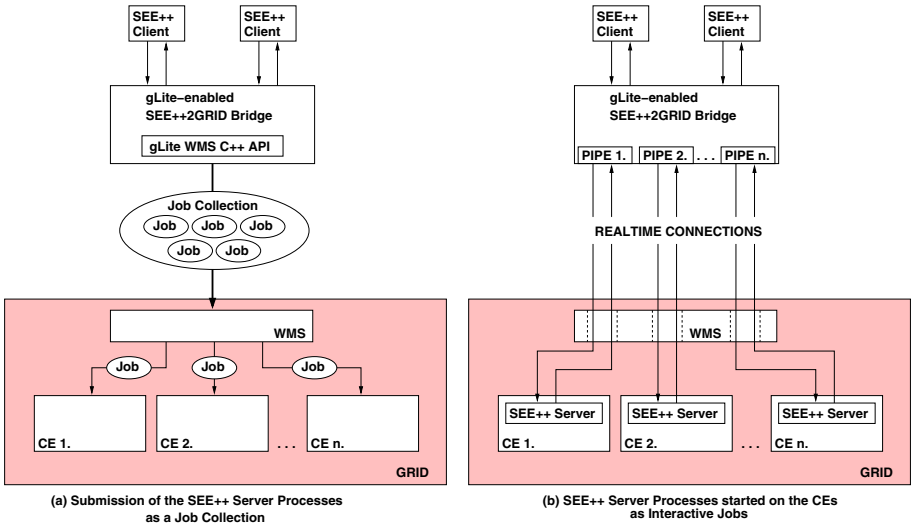


Figure 5. The Initial Architecture of SEE++ based on gLite

We again use some kind of server jobs (as executers for parallel Hess calculations) started via the *Workload Management System (WMS)* of gLite by the "SEE++ to Grid Bridge". For this, we modified the bridge and implemented some new functionalities based on the WMS Job Submission C++ API.

Nevertheless, instead of forking and terminating these jobs as before to return the allocated port number to the bridge, we investigated the exploitation of the interactive job submission feature of gLite. After such a job is submitted, the gLite environment starts a listener process for the job on the client side and establishes a real time connection with the remote host such that the standard I/O streams (e.g.: stdin, stdout, stderr) are redirected from/to remote host. Any user application (like the "SEE++ to Grid Bridge") is able to interact with this process through named pipes. In this way, the server jobs started on some Computing Elements (CEs) on the grid are easily able to return their contact information to the "SEE++ to Grid Bridge"; in the other direction the bridge can send any signal or instruction to them via these connections. However, the computational tasks carrying eye model data and calculated gaze pattern parts are transferred via the SOAP protocol (as before in case of the Globus Toolkit), because it is more efficient and secure (by use of the https transport protocol).

If more SEE++ server processes are started at the same time on the grid, they are submitted as a *collection* of interactive jobs [9], see Figure 5. Job collections are useful functionalities of WMS [6], defined as sets of independent jobs. This speeds up the job submission time, compared to individual jobs and it saves a lot of processing time by reusing the same authentication for all the jobs in the collection.

Our SEE++ server jobs do not have any special requirements on CEs (e.g.: deployment of certain softwares); we apply them for numerous but relatively short gaze pattern calculations (like a typical distributed application designed for a cluster). We should therefore prefer those available CEs which have the shorter estimated response time and the highest number of free CPUs (practically this means that we should properly adjust the *Rank* of the job submission [9]).

Since our SEE++ server jobs act as long-running executer processes, they may fail due to the expiration of the user proxy. To avoid this, we apply a usual proxy renewal mechanism provided by a *MyProxy server*, where the corresponding certificates can be stored.

Later, we will employ and examine the *Workload Management Proxy (WMProxy)* for the submission of the SEE++ server processes as well. WMProxy is a service providing access to the WMS functionality through a Web Service based interface. If the capabilities of WMProxy meet our requirements, we will change from the solution based on WMS C++ API to the one based on WMProxy, because of the numerous advantages of the Web Services.

4.2 The Proposed Architecture

As the next step, we plan to replace the software architecture and authentication methods applied earlier for the SEE++ medical databases by an AMGA-based solution, see Figure 6. AMGA [2] is a database access service for grid applications, which is part of the latest release of gLite. It is able to hide the differences of the user interfaces of the supported underlying database systems and provides a unified

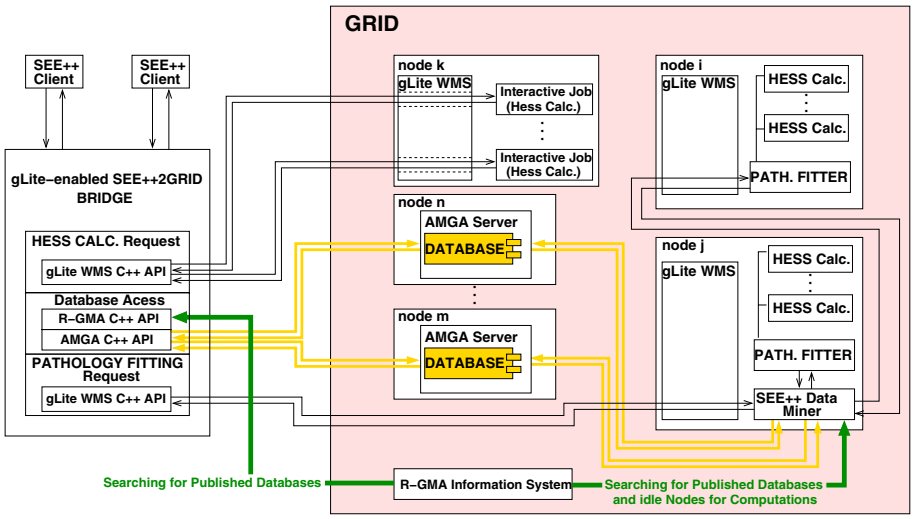


Figure 6. The Design of the gLite Compatible SEE++

access to them with the grid style certificate-based authentication. Since AMGA supports among others MySQL as well, it would be possible to use the same medical databases in the Globus Toolkit 4 and the gLite environments.

In the gLite compatible version of "Grid-Enabled SEE++", we will make one more step further in the development and reach that the system will be able to discover automatically the available databases and the executer jobs on the grid. For this purpose, we plan to apply the *"Relational Grid Monitoring Architecture" (R-GMA)* [6] information system in gLite, which also allows users to publish their own data.

According to this, the "SEE++ to Grid Bridge" creates the corresponding data structure for the SEE++ server processes in the R-GMA system and registers them after it started them and received their contact information. If the bridge terminates a server process running on a CE (or the WMS system reports a failure of such a process), then it removes the corresponding entries from the R-GMA system.

If a pathology fitting is triggered by the user on the client, the "SEE++ to Grid Bridge" starts a "SEE++ Data Miner" process with a measured gaze pattern, see Figure 6. It first looks up for available grid-based SEE++ medical databases in R-GMA, and then searches for similar gaze patterns among the former medical cases in the databases. Each result of this parallel data mining will be stored in a separate file (the functionalities of storing the eye model structure in files have already been implemented in the original SEE++), with which the subsequently started Pathology Fitting processes are fed (an example of a parameter study).

Pathology Fitting is proposed to execute on the gLite architecture as a *parametric job* [6]. A parametric job is a job collection where the jobs are identical apart from the values of their parameters. Since identifiers are assigned to them by WMS, it is possible to monitor and control each of them separately (via the parametric job handle). So, each pathology fitting job will pick up one file established by "SEE++ Data Miner" as a parameter. For accomplishing the necessary gaze pattern calculations, each fitting process looks up in R-GMA for the contact information of some idle SEE++ server processes running on the grid and sends the gaze pattern computation tasks to them for processing.

Since security is a very important issue for the proposed SEE++ medical databases, we will extend the already existing security mechanisms, too. We plan to introduce grid-style certifications for the user authorization in the database access (instead of the existing password-based solution) and encrypt every network transfer in the interactions of the "SEE++ to Grid Bridge" and the database services.

Another important security concept is the managing of the *Virtual Organizations (VOs)*, since we need to control the access to some kinds of grid resources. A VO is an entity which typically corresponds to a particular organization or group in the real world. Membership of a VO grants specific rights to a user or a grid resource. Such an authentication and authorization of the data sources on the grid is a critical issue for "Grid-Enabled SEE++", because we have to be sure that the published medical data will be hosted only by certain trusted grid nodes.

5 Conclusion

In this paper, we reported on our experience of porting a grid application to Globus 4 and gLite, described the problems we encountered and discussed possible solution strate-

gies. This may assist the porting of other applications to the grid using these middleware products.

Our goal is to make "Grid-Enabled SEE++" an efficient tool for supporting and improving the medical treatment of strabismus. Initially we just focused on the development of a SEE++ version based on Globus 4, but we met some difficulties and restrictions in the Globus Toolkit which prevented us to finish the development of a fully functioning version of the software based on WSRF. Therefore, we are implementing a variant of "Grid-Enabled SEE++" which uses the Web Service based interfaces of gLite services (e.g.: MWSproxy and SOAP-based interface of AMGA) which will overcome these problems; we plan to compare its performance with the other "Grid-Enabled SEE++" version based on the Globus Toolkit 4.

On the basis of these developments, we are going to implement a variant of pathology fitting that applies a grid-based parallel search technique to find cases in a distributed medical database of SEE++ that are similar to measured patient data; using these cases as starting points, the method executes multiple independent pathology fitting processes on the grid. These achievements should make SEE++ an effective grid-based tool for giving decision support to the surgeons before eye surgeries.

Acknowledgment

The work described in this paper is partially supported by the Austrian Grid Project, funded by the Austrian BMBWK (Federal Ministry for Education, Science and Culture) under contract GZ 4003/2-VI/4c/2004.

This work makes use of results produced by the Enabling Grids for E-sciencE project, a project co-funded by the European Commission (under contract number INFSO-RI-031688) through the Sixth Framework Programme. EGEE brings together 91 partners in 32 countries to provide a seamless Grid infrastructure available to the European research community 24 hours a day. Full information is available at `http://www.eu-egee.org`.

The G-SDAM framework mentioned in Section 2 is developed by the Institute for Applied Knowledge Processing (Institut Für Anwendungsorientierte Wissensverarbeitung — FAW) as a partner of the "Grid-Enabled SEE++" project.

References

[1] Austrian Grid home page. `http://www.austriangrid.at`

[2] AMGA User's and Administrator's Manual `http://project-arda-dev.web.cern.ch/project-arda-dev/metadata/downloads/amga-manual_1_2_3.pdf`

[3] Karoly Bosa, Wolfgang Schreiner, Michael Buchberger, Thomas Kaltofen, *SEE-GRID, A Grid-Based Medical Decision Support System for Eye Muscle Surgery*, 1st Austrian Grid Symposium, December 1-2, 2005, Hagenberg, Austria. OCG Verlag, pp. 61-74.

[4] Michael Buchberger, *Biomechanical Modelling of the Human Eye*,Ph.D. thesis, Johannes Kepler University, Linz, Austria, March 2004. `http://www.see-kid.at/download/Dissertation_MB.pdf`

[5] EGEE-II home page, 2006. `http://www.eu-egee.org`

[6] gLite 3.0.0 home page, 2007. `http://www.glite.org`

[7] The Globus Tookit. `http://www.globus.org/toolkit/`

[8] *A Report on a Unified Grid-aware Access Layer for SEE-GRID Data Sets*, Austrian Grid Deliverable M-4aA-1c, FAW Institute and RISC-Linz Institute, Johannes Kepler University, Linz, August 2005.

[9] *Job Description Language Attributes Specification for the gLite Middleware*, `https://edms.cern.ch/file/555796/1/EGEE-JRA1-TEC-555796-JDL-Attributes-v0-8.pdf`

[10] Thomas Kaltofen, *Design and Implementation of a Mathematical Pulley Model for Biomechanical EyeSurgery*, Diploma thesis, Upper Austria University of Applied Sciences, Hagenberg, June 2002. `http://www.see-kid.at/download/Pulley_Model_Thesis.pdf`

[11] *Mapping WSDL and XSD schema to C in Globus 4*, `www.globus.org/toolkit/docs/4.0/common/cwscore/WSDLtoCBindings.pdf`

[12] Daniel Mitterdorfer, *Grid-Capable Persistance Based on a Metamodel for Medical Decision Support*, Diploma thesis, Upper Austria University of Applied Sciences, Hagenberg, July 2005.

[13] Herbert Rosmanith, Peter Praxmarer, Dieter Kranzmller, Jens Volkert, *Towards Job Accounting in Existing Resource Schedulers: Weaknesses and Improvements*, The 2nd International Conference on High Performance Computing and Communications (HPCC-06), Munich, Germany, September 13-15, 2006.

[14] SEE-KID home page, 2006. `http://www.see-kid.at`