The Porting of a Medical Grid Application from Globus 4 to the gLite Middleware

Károly Bósa and Wolfgang Schreiner

Abstract In this paper, we compare two implementations of a grid-based software system on the grid middleware Globus Toolkit 4 and gLite, respectively. This system called "Grid-Enabled SEE++" is a grid-based simulation software that supports the diagnosis and treatment of certain eye motility disorders (strabismus). First, we developed a parallel version of the software with the help of Globus 4. Since we met with some limitations of Globus 4, we also designed and developed a version of SEE++ based on gLite. We focus on the differences between the initial Globus version and the gLite version of our software system and report on some comparative benchmark results.

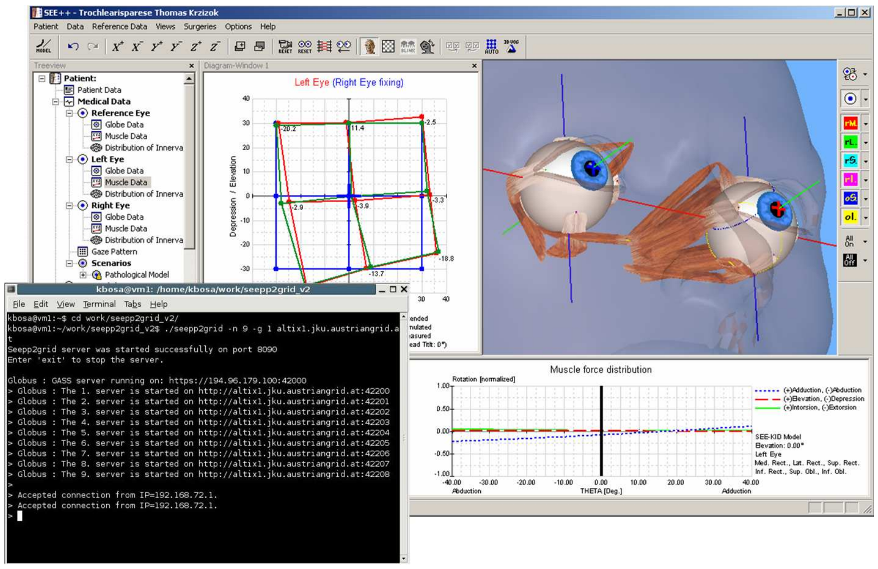
Key words: "Grid-Enabled SEE++", Grid Applications, Grid Middleware, Globus, gLite

1 Introduction

Nowadays various types of grid middleware (Unicore, Globus, LCG, gLite, etc.) are in use by several research projects. These systems have many similar features, but there are almost no reports in literature which compare them next to each other in similar circumstances. We had the opportunity to make such a comparison, since we developed two versions of a grid application on the basis of the popular grid middleware systems Globus Toolkit 4 [9] and gLite [8], respectively.

The core software called "Grid-Enabled SEE++" is a grid version of the SEE++ software system [6, 12, 16] for the biomechanical 3D simulation of the human eye and its muscles (see Figure 1). The software deals with the support of diagnosis and treatment of *strabismus*, which is the common name of the misalignment of the eyes

Károly Bósa and Wolfgang Schreiner
Research Institute for Symbolic Computation (RISC)
e-mail: FirstName.LastName@risc.uni-linz.ac.at


SEE++ - Trochleariasparese Thomas Krzizok
Patient Data Reference Data Views Surgeries Options Help
Treeview
Patient:
Patient Data
Medical Data
Reference Eye
Globe Data
Muscle Data
Distribution of Innerva
Left Eye
Globe Data
Muscle Data
Distribution of Innerva
Right Eye
Globe Data
Muscle Data
Distribution of Innerva
Gaze Pattern
Scenarios
Pathological Model
Diagram-Window 1
Left Eye (Right Eye fixing)
Depression / Elevation
Muscle force distribution
Rotation [normalized]
(+)Adduction, (-)Abduction
(+)Elevation, (-)Depression
(+)Intorsion, (-)Extorsion
SEE-KID Model
Elevation: 0.00°
Left Eye
Med. Rect., Lat. Rect., Sup. Rect. Inf. Rect., Sup. Obl., Inf. Obl.
THETA [Deg.]
Abduction
Adduction
kbosa@vm1: /home/kbosa/work/seepp2grid_v2
File Edit View Terminal Tabs Help

```
kbosa@vm1:~$ cd work/seepp2grid_v2/
kbosa@vm1:~/work/seepp2grid_v2$ ./seepp2grid -n 9 -g 1 altix1.jku.austriangrid.a
t
Seepp2grid server was started successfully on port 8090
Enter 'exit' to stop the server.

Globus : GASS server running on: https://194.96.179.100:42000
> Globus : The 1. server is started on http://altix1.jku.austriangrid.at:42200
> Globus : The 2. server is started on http://altix1.jku.austriangrid.at:42201
> Globus : The 3. server is started on http://altix1.jku.austriangrid.at:42202
> Globus : The 4. server is started on http://altix1.jku.austriangrid.at:42203
> Globus : The 5. server is started on http://altix1.jku.austriangrid.at:42204
> Globus : The 6. server is started on http://altix1.jku.austriangrid.at:42205
> Globus : The 7. server is started on http://altix1.jku.austriangrid.at:42206
> Globus : The 8. server is started on http://altix1.jku.austriangrid.at:42207
> Globus : The 9. server is started on http://altix1.jku.austriangrid.at:42208
>
> Accepted connection from IP=192.168.72.1.
> Accepted connection from IP=192.168.72.1.
> 
```

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

where eyes point in different directions such that a person may see double images. 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.

The doctors intend to work with the software in an interactive manner (changing the eye model parameters by a manual trial and error method), hence the adequate response times are essential for the usability of SEE++. It is also possible to semi-automatize the determination of the patient pathology on the grid (a non-linear optimization problem) by the procedure called *pathology fitting* [3].

In [3, 4], we combined the SEE++ software with the Globus middleware applying both the *pre-Web Service* (pre-WS) and the *Web Service* (WS) frameworks and developed a parallel version of the simulation. By this, we speeded up this simulation by a factor of 14–17.

Furthermore, we reported the prototype implementation of a medical database component for "Grid-Enabled SEE++" [13], which is going to be used for storing patient medical data with eye model parameters. These stored pathological cases will be utilized as initial estimations by the new grid-based pathology fitting algorithm presented in [3].

Since we met with some limitation of the Globus Toolkit 4 [4], we also elaborated an initial design of a SEE++ version compatible with the gLite grid middleware [5] in the frame of *"Enabling Grids for E-sciencE 2" (EGEE2)* project [7].

The topic of this paper is to present a refined architecture adapted to the recently implemented gLite-based SEE++ and to report a comparison between this new ver-

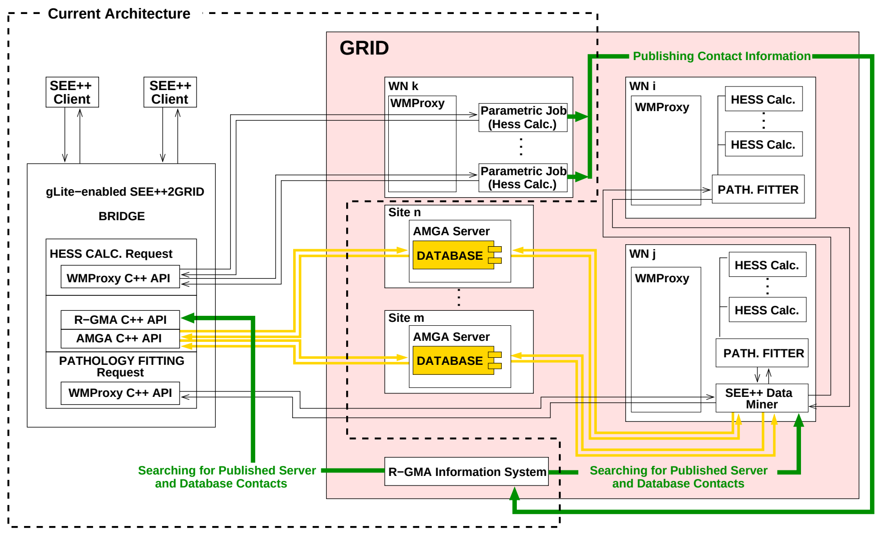


Fig. 2 The Design of the gLite Compatible SEE++

sion of our software system and its other version based on Globus 4. The new design is described in Section 2. In Section 3, we focus on the new features of the gLite compatible SEE++. Finally, we present in Section 4 an experimental comparison with benchmark results between the Globus and the gLite-based versions.

2 The New Architecture based on gLite

As in the case of the Globus-based version, the initial component of the gLite-based version is the “SEE++ to Grid Bridge”, via which the unchanged SEE++ client can get access to the infrastructure of the grid (see the box in Figure 2 bordered by the dashed line). Before the bridge accepts the computational requests from the SEE++ clients, it starts some grid-enabled SEE++ servers in 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. The “SEE++ to Grid Bridge” is able to split calculation requests of clients into subtasks [3] and to distribute them among the servers (data parallelism).

Nevertheless, we found the same problem as in the Globus version of our software, namely how to send back the contact information of the started server/executer processes to the bridge. In the original design, we proposed to exploit an interesting feature of the gLite *Workload Management System (WMS)* [17] called *interactive jobs* which returns the corresponding data via interactive connections. However later we found that such a feature exists only as a theoretical option for job submissions in gLite, but it is not supported by real grid architectures (EGEE [7], int.EU.grid [10], etc.).

So we decided to apply the *“Relational Grid Monitoring Architecture” (R-GMA)* information system [14] of gLite for this purpose, which allows users and grid applications to publish their own data. From time to time, each server announces into a R-GMA table its address (hostname and port) together with the number of subtasks received but not yet calculated by the server. The “SEE++ to Grid Bridge” runs a query on the table R-GMA regularly as well and updates its list of the available servers. This approach is much more versatile and sophisticated, since many kinds of information (e.g. workload) can be published about all available SEE++ servers to more than one bridge component. Each server is always started with two arguments: its identifier (generated by the bridge) and a unique identifier of the bridge (e.g. the address and port where the bridge is listening for the requests of the SEE++ clients). These two pieces of data are used as a primary key in the R-GMA system, when a server publishes its own contact and workload information.

Every “SEE++ to Grid Bridge” can be tuned such that it either uses only those servers that were started by itself or it always chooses for each calculation request some servers from the pool of all servers available on the grid; this choice is made with the help of R-GMA on the basis of the published workload information. A server terminates, if it does not receive any computational request for a predefined time interval (typically one hour).

The approach to apply executer jobs works only if the *worker nodes (WNs)* on where these jobs are executed are not located within private subnetworks. To enforce this constraint, we applied the following *Requirement* condition in the JDL [11] file of the job submission:

```
Requirements = other.GlueHostNetworkAdapterOutboundIP==True;
```

This constraint guarantees that the SEE++ servers are started only on those WNs, that are able to interact through the Internet. We also applied a *Rank* criterion, that helps to choose WNs from those which fulfill the Requirement:

```
Rank = (other.GlueCEStateWaitingJobs == 0 ?
            -other.GlueCEStateEstimatedResponseTime :
            -other.GlueCEStateWaitingJobs);
```

According to this criterion, those WNs are preferred which either are idle (there is not any scheduled job in the state ”WAITING”) and have minimal communication latencies or (if there are not enough idle WNs) which have the minimal number of waiting jobs.

The further parts of the design regarding a distributed medical database based on the database access service of gLite called *AMGA* [2] and the grid-based pathology fitting algorithm (depicted on Figure 2) remain as they have already been proposed in [4].

SEE++ Versions Features	Globus−Based SEE++ Version	gLite−Based SEE++ Version
Computations are performed	**by server jobs**	**by server jobs**
SEE++ clients interact with the server jobs	**via a bridge component**	**via a bridge component**
The communication protocol between the software components	**SOAP**	**SOAP**
Server jobs are submitted on the grid	**one by one as single jobs**	**as a special collection of jobs (parametric job)**
Server jobs return their contact information	**by a "forked" and terminated instance of each job**	**via R−GMA tables**
Resource discovery	**not implemented**	**automatic (part of the job submission framework)**
Proxy renewal for long−running jobs	**not implemented**	**automatic (part of the job submission framework)**
"File stage on"	**no (Servers must be preinstalled on the corresponding grid nodes)**	**yes (the executables of the servers are sent to the WNs by the job submission framework)**
Platform dependency	**on every UNIX/Linux based platform where Globus runs**	**only on x86 platforms with Scientific Linux (dependency of the existing gLite versions)**

Fig. 3 A Comparison between the Two Versions of “Grid-Enabled SEE++”

3 New Features of the gLite-Based Version

In Figure 3, we summarized and compared the essential features of the versions of our “Grid-Enabled SEE++” software system based on Globus Toolkit and gLite respectively.

An advantage of the application of gLite is the implicit and automatic support for resource discovery, which is part of the job submission framework i.e. hidden from the API and UI levels. In Globus, if the developers would like to provide their software with this property, they have to implement it on their own by using of a non-trivial API (which we did not apply in our Globus-based SEE++).

One of the essential distinctions between the Globus and gLite-based SEE++ is how the server contact information is returned to the bridge component. In Globus, we applied some kind of "hack" for overcoming this problem. According to this, a server started on a grid node forks itself after it has allocated a port number and terminates [3]. Unfortunately, this technique may induce some problems in the local resource management systems (e.g. PBS). Namely, such a local scheduler may assume the resource is free and may assign some other jobs to it or it may kill the forked process (in order to clean up). Although we could handle these situations in the case of some local batch queueing systems by applying some simple techniques described in [4], we could not find a general solution. By employing the new publishing method based on R-GMA, such a problem cannot arise at all (moreover there exists additional benefits as discussed in Section 2).

We employed user proxy certificates stored on a *MyProxy* [15] server, since the SEE++ servers have to be authenticated (by a valid proxy available on a MyProxy server) for accessing the R-GMA grid service from remote WNs. As a side effect of this requirement, our application is equipped with the automatic proxy renewal function: the long-running SEE++ servers are not killed on the WNs by the local resource management after the proxy of their user/submitter expires as long as the proxy credential can be renewed from a given MyProxy server.

The SEE++ server jobs are executed on the gLite architecture as *parametric jobs* via the *Workload Management Proxy (WMProxy)* [17]. A parametric job is a special type of job collections and it is defined as a set of jobs which are identical apart from the values of their parameters. On the one hand, 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; on the other hand, it is possible to monitor and control each of its jobs separately (via the parametric job handle). At the submission of a parametric job, the JDL file is usually supplemented some additional lines as follows:

```
[
...
JobType="Parametric";
...
Arguments = "_PARAM_ bridge-URL:port";
...
StdError = "stderr_PARAM_.log";
OutputSandbox = {"stderr_PARAM_.log"
...
Parameters = numberOfServerJobs;
ParameterStart = 0;
ParameterStep = 1;
...
]
```

The JDL file for a parametric job may contain a built-in variable called *_PARAM_* and three additional specific attributes *Parameters*, *ParameterStart* and *Parameter-Step*. These attributes represent respectively the maximum value (or in case of non-numeric parameter the set of values) of, the starting value of and at last but not least the step for the modification for the (numeric) values of the parametric variable _PARAM_. In our case, the variable _PARAM_, which has a different value for each single SEE++ server job is employed to determine the identifier of a SEE++ server (see the first argument in the line of JDL attribute *Arguments* above). Additionally, we also assign a log file with a unique name (generated with help of the parametric variable) to the standard error of each server job; these files will be collected if the executions of the server jobs are over (see the JDL attributes *StdError* and *OutputSandbox* above).

To avoid the pre-installation of the SEE++ servers on the grid, we exploit the "file stage on" feature of the WMProxy to transfer the executable and some other input files to the corresponding WNs in the job submission phase.

Summarizing this section, our gLite compatible SEE++ has some new features, which we achieved with investing relatively few efforts. To extend the Globus-based version with these properties is either not feasible or it requires much more time and human resources.

4 Experimental Comparison

The basis for this experimental comparison is the simulation of a typical medical examination called *Hess-Lancaster test*, whose parallel gridified implementation can represent a wide group of grid applications, see Section 4.1. In Section 4.2, we present the outcome of some benchmarks performed with this mentioned medical simulation.

4.1 A Medical Simulation as the Basis of the Comparison

In [3], we combined the SEE++ software with the Globus middleware [9] and developed a parallel version of the simulation of the Hess-Lancaster test. Now, we reimplemented this parallel simulation in gLite, too.

From the Hess-Lancaster test the reason for the pathological situation of a patient can be estimated. The outcome of such an examination consists of two *gaze patterns* of blue points and of red points respectively (see the diagram in the middle of the GUI of SEE++ on Figure 1). 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++ comprises 9 points. Bigger gaze patterns with 21 and 45 points are possible and provide more precise results for the decision support in case of some pathologies, but their calculations are more time consuming. The size of the gaze pattern determines the size of the problem, too. The maximum number of grid jobs we used in a session was 45, because gaze patterns used in medical examinations can consist of at most

Number of SEE++ Server Jobs	1	3	9	25	30	45
Submission via Globus pre−WS architecture	0,85s	0,92s	0,98s	1,06s	1,09s	1,15s
Submission via Globus WS architecture	9,5s	10s	11s	15s	16s	20s
Submission via WMProxy (gLite) including Resource Discovery, File Staging (2Mb) and Publishing Contact Information via R−GMA	38s	46s	91s	142s	156s	224s

Fig. 4 Startup Times in Globus and in gLite Versions

45 gaze points (in the case of the application of 45 jobs, only one gaze point was computed by one server job in a session).

Our experiences with the simulation of the Hess-Lancaster test on different middlewares can help and facilitate the work of many grid application developer research groups, because its implementation represents the following very frequently applied programming strategies of nowadays grid applications:

Parameter Study Since the calculations of each gaze points is completely independent from each other, there is no communication among the server processes. Hence our simulation is a typical example for the *parameter study*, where the same algorithm is executed on several grid node but with different arguments.

Interactivity "Grid-Enabled SEE++" has other important characteristics that are a distributed simulation backend connected to an interactive real-time user interface (the doctors change the eye parameters by a manual trial and error method and in turn they wait for the results of the simulation). These characteristics are present in many classical grid monitoring applications and additionally there are numerous research efforts for establishing interactive grid architectures [10].

4.2 Benchmarks

In some benchmarks, we have compared the effectiveness of the two versions of "Grid-Enabled SEE++". Figure 4 and Figure 5 depict the average execution time of 5 computations in different situations where 1, 3, 9, 25, 30 or 45 processors were used on the grid (one server process was started on each processor).

The reported measurements were accomplished on different hardware architectures in case of Globus and gLite respectively. The reason for this fact that we do not have access to any grid testbed, where both required grid middlewares are deployed and available on the same computational resources. Moreover we intended to investigate the behaviors and the applicabilities of our SEE++ versions on some real grid

Number of Jobs/Servers	**1**	**3**	**9**	**25**	**30**	**45**
Hess Test with Globus Compatible SEE++	**27.18s**	**18.81s**	**9.11s**	**2.17s**	**2.10s**	**1.89s**
Hess Test with gLite Compatible SEE++	**39.48s**	**28.05s**	**16.87s**	**4.63s**	**4.21s**	**3.03s**

Fig. 5 Execution Times in Globus and gLite

architectures (as described in detail below) providing *"production services"* instead of within ideal circumstances on an artificial grid testbed.

The test cases based on Globus 4 were executed on the Austrian Grid site `altix1.jku.austriangrid.at`, which contains 64 Intel Itanium processors (1.4GHz) and resides at the Johannes Kepler University (JKU) in Linz. The "SEE++ to Grid Bridge" and SEE++ clients were always executed at the RISC institute located in Hagenberg which has a one Gigabit/sec connection to the JKU. In case of 25 or more processors, we used some processes on the grid site `altix1.uibk.ac.at` in Innsbruck that comprises 16 CPUs of the same type.

The test cases based on gLite were performed on some clusters of the architecture of the *Int.EU.Grid* Project [10]. The server jobs were randomly disseminated among some clusters in Germany (122 CPUs), Poland (32 CPUs), Slovakia (32 CPUs) and Spain (20 CPUs). All of these CPUs are based on Intel x86 and x86-64 architectures, but their speed characteristics is unknown.

As a first step, we have compared the costs of the submissions of our SEE++ server processes via Globus pre-WS and WS architectures and gLite WMProxy, see Figure 4. We found it quite challenging to start more than 20–25 jobs on the gLite architecture, because in these cases some jobs often got stuck in the submission procedure with the state "WAITING" for a long time (from 10 minutes to several hours). Therefore, the values related to gLite are the average values for 5 "successful" job submissions.

From the values listed in Figure 4, we can see obvious differences among the overheads of the job submissions in the different architectures. Globus (both the pre-WS and the WS architectures) seems much more efficient. Nevertheless, this comparison is not completely fair with respect to gLite, since Globus performs only simple job submissions to one or two dedicated sites, while in the startup phase gLite additionally discovers resources, transfers files to the WNs (with a total file size of approx. 2Mb) and finally publishes the server contact information via R-GMA.

In the second step, we have investigated the performance of our parallel grid-based simulation on Globus and gLite, see Figure 5 and Figure 6. We do not report different results for the tests run with Globus based on pre-WS and WS architectures, because apart from how the SEE++ servers are started on the grid, there is not difference the operation of the two Globus-based versions of our software. In these test cases, we speeded up the simulation by a factor of 12–14 in Globus and by a factor 9–13 in gLite.

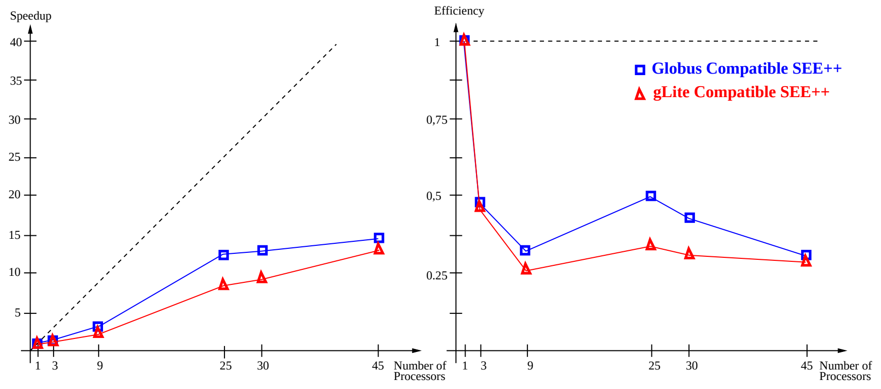


Fig. 6 Speedup and Efficiency Diagrams in Globus and gLite

Apparently the results achieved with Globus look better again, but as in the previous comparison the measured values do not reflect the whole picture: in the tests based on Globus we employed homogeneous hardware and there were fast connections between the bridge and the servers with relatively consistent quality. In the gLite tests the hardware environment was heterogeneous and the communication latencies were higher with large variations. Nevertheless, the average values in Figure 5 are closer to each other when we applied 25 or more jobs, because in the case of more jobs, the load can be more balanced among the various grid nodes. These facts imply that the differences between the values concerning to Globus and gLite on Figure 5 and Figure 6 are caused mostly by the disparity of the hardware architectures of the two testbeds rather than by the applied grid middleware.

5 Conclusions and Future Works

Our comparisons show that while the Globus Toolkit 4 is faster and more efficient, gLite is much more sophisticated and developer friendly. Therefore, it seems more appropriate as a basis of further development.

Another difference of the middlewares is the quantity of existing documentation. In case of Globus 4, the documents are often sketchy and the complete examples are mostly missing (especially in case of the WS C APIs). On the contrary, there is sufficient information and example source codes available in gLite.

Our next step will be the porting of our medical database to AMGA [2], which provides a unified access to them with the grid style certificate-based authentication and authorization. Since AMGA supports among other database systems MySQL as well, it would be possible to reuse the same medical databases in the Globus Toolkit 4 and the gLite environments. On the basis of these developments, we are going to continue the implementation of the grid-based pathology fitting. These achievements should make SEE++ an effective grid-based tool for giving effective decision support to the surgeons before eye surgeries.

Acknowledgements The work described in this paper is partially supported by the Austrian Grid Project [1], funded by the Austrian BMBWK (Federal Ministry for Education, Science and Culture) under contract GZ BMWF-10.220/0002-II/10/2007.

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 [7].

References

1. Austrian Grid home page. `http://www.austriangrid.at`
2. AMGA Project home page `http://amga.web.cern.ch/amga/`
3. Károly Bósa, 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. Károly Bósa, Wolfgang Schreiner, Michael Buchberger, Thomas Kaltofen. *A Grid Software for Virtual Eye Surgery Based on Globus and gLite* ISPDC 2007, Hagenberg, Austria, July 5-8, 2007. IEEE Computer Society, pp. 151-158.
5. Károly Bósa, Wolfgang Schreiner, Michael Buchberger *The Porting of a Grid Software for Virtual Eye Surgery from Globus 4 to gLite*, Poster on the 3rd EGEE User Forum, Clermont-Ferrand, France, Februar 10-14, 2008.
6. 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`
7. EGEE-II home page, 2008. `http://www.eu-egee.org`
8. gLite 3.0.0 home page, 2008. `http://www.glite.org`
9. The Globus Tookit home page, 2008. `http://www.globus.org/toolkit/`
10. Int.EU.Grid Project home page, 2008. `http://www.interactive-grid.eu/`
11. *Job Description Language Attributes Specification*, `https://edms.cern.ch/file/590869/1/EGEE-JRA1-TEC-590869-JDL-Attributes-v0-8.pdf`
12. 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`
13. Daniel Mitterdorfer, *Grid-Capable Persistance Based on a Metamodel for Medical Decision Support*, Diploma thesis, Upper Austria University of Applied Sciences, Hagenberg, July 2005.
14. Relational Grid Monitoring Architecture (R-GMA) home page. `http://www.r-gma.org/`
15. MyProxy home page, 2008. `http://grid.ncsa.uiuc.edu/myproxy/`
16. SEE-KID home page, 2008. `http://www.see-kid.at`
17. Workload Manager Proxy (WMProxy) C++ API Manual, 2008. `http://egee-jra1-wm.mi.infn.it/egee-jra1-wm/api_doc/api_docwmproxy_cpp/`