AUSTRIAN GRID

THE INITIAL VERSION OF SEE-GRID

Document Identifier:	**AG-DA1c-1-2005_v1.doc**
Workpackage:	**A1c**
Partner(s):	**Research Institute for Symbolic Computation (RISC)** **Upper Austrian Research (UAR)**
Lead Partner:	**RISC**
WP Leaders:	**Wolfgang Schreiner (RISC), Michael Buchberger (UAR)**

Delivery Slip

	Name	Partner	Date	Signature
From	Károly Bósa	RISC	31.03.2005	
Verified by				
Approved by				

Document Log

Version	Date	Summary of changes	Author
1.0	2005-03-10	Initial Version	See cover on page 3

THE INITIAL VERSION OF SEE-GRID

Karoly Bosa
Wolfgang Schreiner

Research Institute for Symbolic Computation (RISC)
Johannes Kepler University Linz
{Karoly.Bosa, Wolfgang.Schreiner}@risc.uni-linz.ac.at

Michael Buchberger
Thomas Kaltofen

Department for Medical Informatics
Upper Austrian Research (UAR)
Thomas.Kaltofen@uar.at

April 19, 2005

Table of Contents

Abstract

This document describes the functionality of the “SEE++ to Grid Bridge”, which is the initial component of SEE-GRID. SEE-GRID is based on the SEE++ software for the biomechanical simulation of the human eye. SEE++ was developed in the SEE-KID project by the Upper Austrian Research and the Upper Austria University of Applied Sciences. SEE++ consists of a client component for user interaction and of a server component that runs various computations.

Via the “SEE++ to Grid Bridge”, normal SEE++ clients are able to access and exploit the computational power of the Austrian Grid. The bridge starts SEE++ servers on various grid sites and distributes to them computational tasks received from some SEE+ clients. This paper addresses the following issues:

- How the initially proposed design of SEE-GRID was adapted to the current software infrastructure of the Austrian Grid.
- How our environment was configured in order to access to the Austrian Grid.
- How the “SEE++ to Grid Bridge” works together with SEE++ clients and the middleware software layer of the Austrian Grid.
- How SEE++ server processes are started in the grid environment.
- How computational tasks are distributed to the Austrian Grid resources.
- How and with what kind of parameters the “SEE++ to Grid Bridge” can be used.

At the end of this paper, we summarize our first experiences and benchmark results related to this initial version of SEE-GRID.

1 The Current Architecture of SEE-GRID

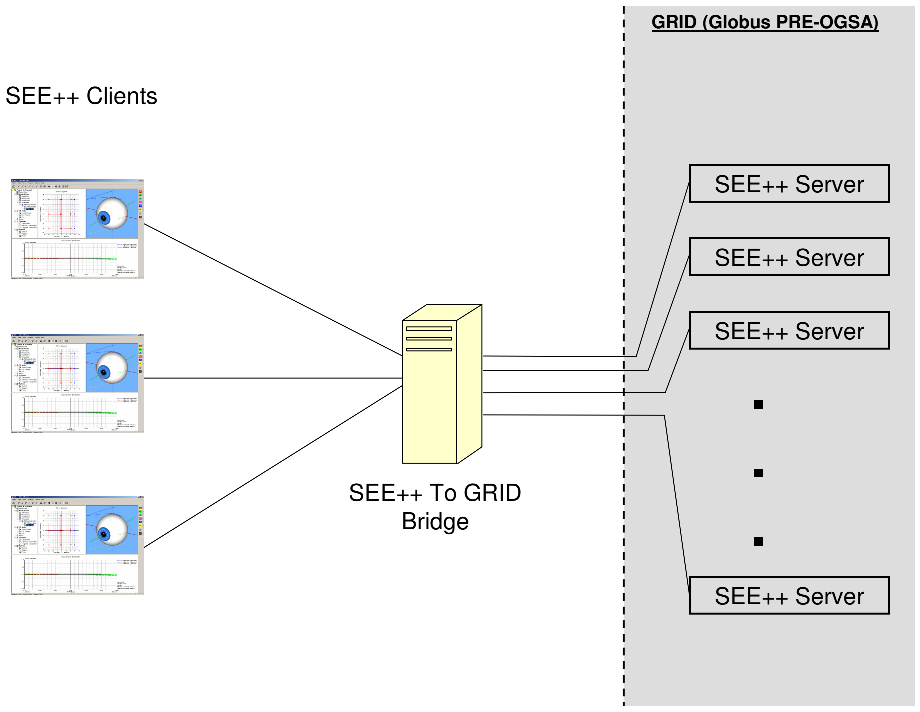


Figure 1: The Current Architecture of SEE-GRID

The design of SEE-GRID is based on the SEE++ software for the biomechanical simulation of the human eye. SEE++ was developed in the frame of the SEE-KID project by Upper Austrian Research and the Upper Austria University of Applied Sciences [SEE-KID, Buchberger 2004, Kaltofen 2002]; it consists of a client component for user interaction and of a server component that runs various computations ("currently "Hess Diagram Calculation").

In the first phase of the SEE-GRID project, our main goals were the followings. First of all, we wanted to try out how the current version of the SEE++ software can be combined with the architecture of Austrian Grid. Then, we wanted to demonstrate how a noticeable speedup can be reached in SEE++ — by applying simple data parallelism — by the exploitation of the huge computational power of the Austrian Grid.

For achieving this, we have implemented the initial component of SEE-GRID, called "SEE++ to Grid Bridge", via which the normal SEE++ client can get access to the Austrian Grid and use its infrastructure to perform computations (see Figure 1). When this application is executed, it starts some SEE++ server on one (or perhaps more) grid site(s) and then waits for computational tasks from its clients. The SEE++ clients can access this application in the same way as in the original SEE++ system; the usage of grid resources is completely transparent to them.

1.1 The Algorithm and the Basic Parallelization Idea

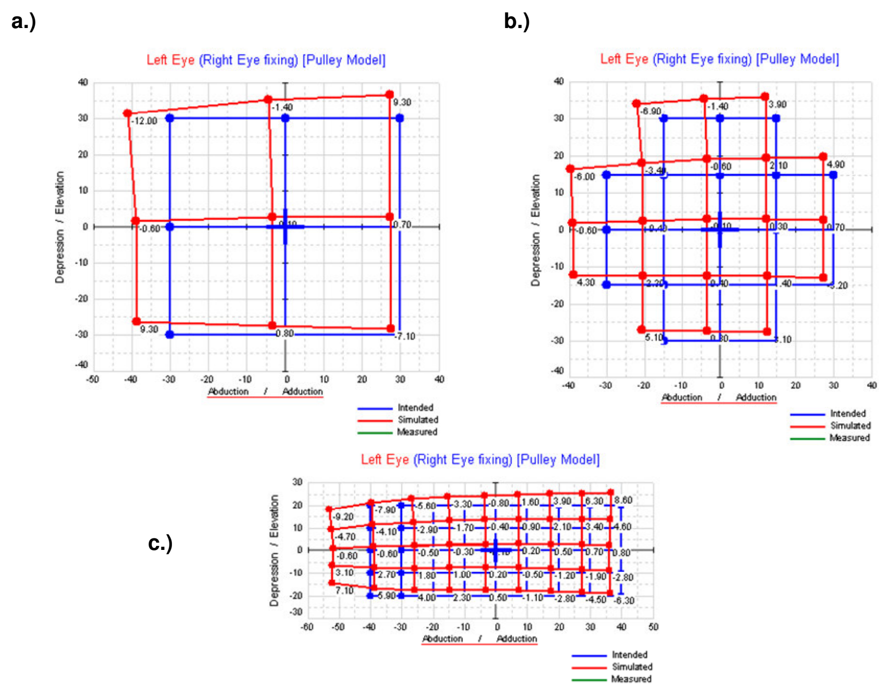


Figure 2: Typical Gaze Patterns in SEE++:
a.) Default **b.)** Orbit and **c.)** Brainstem

SEE++ is able to simulate the result of the Hess-Lancaster test which is a test for binocular functions with separated images for both eyes [SEE-KID, 2004]. A gaze pattern (or "Hess Diagrams") which is the outcome this kind of examination is an eye pair’s image of set of reference points in the plane (the most common gaze patterns used by the software are depicted on Figure 2). If a patient stares such a diagram with a perfectly healthy pair eyes, these points form a regular pattern (called intended gaze pattern) for her (see the blue lines on Figures 2a, b and c). But if somebody has an impaired pair eyes, the seen pattern is distorted (see the red lines on Figures 2a, b and c).

The algorithm of the gaze pattern calculation always receives a complete eye model (fixing eye, following eye, reference eye) [SEE-KID, 2004] and a matrix with the coordinates of the points of the currently used intended gaze pattern. The rough sketch of the algorithm is the following:

```
hessDiagram(E1, E2, E, I):

    for each intended gaze position i ε I do
        C[i] := position(E1, E2, E, i)
    return C
```

```
position($E1, E2, E, i$):

    1. Compute from $I$ and $E$ the 3D gaze position $i'$.

    2. Compute from $i'$ and $E1$ the innervations $I1$ of the fixing eye.
    3. Compute from $I1$ and $E$ the intended gaze position $i1$ of the fixing
       eye.

    4. Mirror $i1$ to yield the intended gaze position $i2$ of the following eye.

    5. Compute from $i2$ and $E$ the innervations $I2$ of the following eye.
    6. Compute from $E2$ and $I2$ the gaze position $c$ of the following eye.

    return $c$
```

The output C is the (calculated) gaze pattern of the eye model for the points in the intended gaze pattern.

From the algorithm sketch, it can be seen that the calculation of the different gaze points is completely independent from each other. We exploited this fact for the parallelization of the gaze pattern calculation in the "SEE++ to Grid Bridge" such that it is able to split gaze pattern calculation requests of clients to independent subtasks and to distribute them among the SEE++ servers (data parallelism).

1.2 Design Adaptation for the Austrian Grid Environment

In the original design document [SEE-GRID Design, 2004], we proposed to use the Web Service (WS) interface of the latest stable release of Globus [Globus, 2004] (Globus 3.2) in order to connect SEE++ to the Austrian Grid. For this, we planed to implement two intermediate components. The first proposed component was a bridge/proxy between SEE++ clients and Globus, which would have been a dual role:

- To hide all the operations related to any interaction with Globus, since the SEE++ client (Windows application or Java applet) must not be affected by the proposed grid extension.
- This component would have been a Java program, which encodes the requests from the clients into strings which are "tunnelled" through the grid middleware layer. This would have been necessary, because the SEE++ server is implemented in C++ and communicates with the clients by exchanging C++ data structures in SOAP [SOAP, 2003] encoding, but the WS interface of the Globus 3.2 toolkit can be programmed only in Java.

The second component would have been another Java program between Globus and each SEE++ server; its role would have been to unpack the requests into the form expected by the server.

Unfortunately, the WS part of Globus has not yet been integrated into the software specification of Austrian Grid [AGRID S. Spec., 2004]. Therefore, we had to modify our conception and to change our design to the pre-WS interface of Globus. The major alterations in our design were that we had to use C/C++ instead of Java, because the Globus pre-WS interface is available only as C libraries.

Usage of C/C++ has some advantages in our case, since the tunnelling of the C++ data structures contained by the SOAP messages became unnecessary and so did the second intermediate component between the Globus and each SEE++ server. However, we lost several benefits offered by the web service architecture. Therefore, after Globus 4 is issued and deployed on the Austrian Grid in the near future, we will change to it, because it will support to implement web services in C/C++ as well.

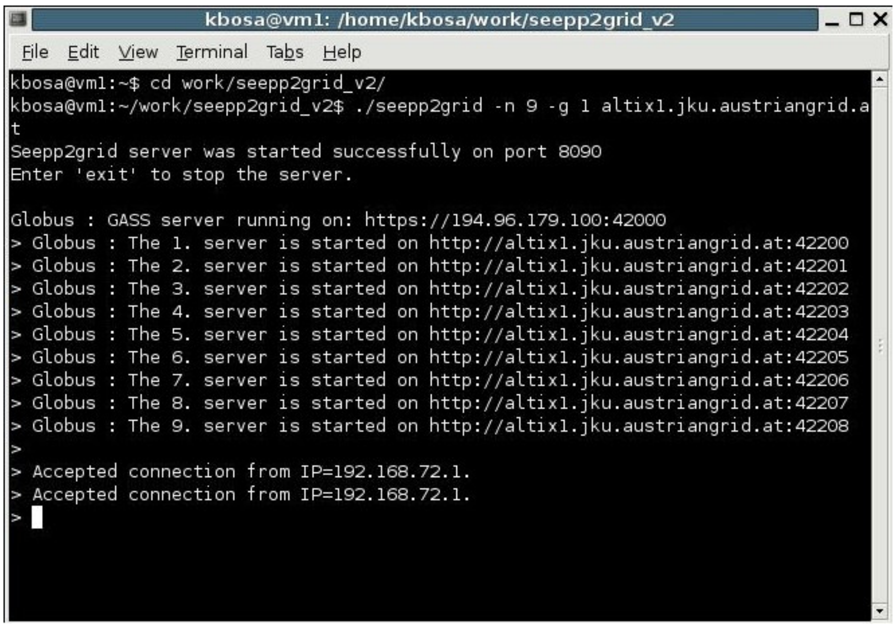
For implementing the communication protocol on the "SEE++ to Grid Bridge", we used gSOAP [gSOAP, 2005] as in the case of the development of the original SEE++ components. gSOAP is collection of (pre)compiler tools that provide a SOAP/XML to C/C++ language binding to ease the development of SOAP/XML Web services and client application in C and/or C++. Since gSOAP does not contain libraries for parsing XML from the source code (in contrast to Java), every message content received by the "SEE++ to Grid Bridge" is deserialized to C++ classes, which are processed for distribution to different servers and then serialized again before sending further in SOAP messages.

1.3 The Configuration of our Network for Using the Austrian Grid

Currently we have set up a grid client/portal machine in the office of Upper Austrian Research in Hagenberg. Since our network is protected by a firewall and we use DHCP and NAT (Network Address Translation), the machines of this local network are normally not reachable from outside. To configure the grid portal machine for the call back methods of Globus pre-WS architecture, we had to give a fixed IP address to this machine in order to apply port forwarding through our firewall to it. Then we configured our Globus installation on the portal machine to use the specific port range for callbacks (by the environment variable GLOBUS_TCP_PORT_RANGE).

Additionally, we had to adjust the primary domain name of the portal machine to the same fully qualified domain name as our network is visible from outside (in order that the Globus GRAM client module, GASS server module, etc. would be able to generate the corresponding callback strings).

2 The "SEE++ to Grid Bridge"


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.
>
```

Figure 3: Gaze Pattern Calculation via "seepp2grid" Bridge

The "SEE++ to Grid Bridge" acts as a SEE++ sever to the SEE++ clients and as a Globus client to the Austrian Grid.

2.1 How to Use "SEE++ to Grid Bridge"

Before running "SEE++ to Grid Bridge", the user has to create her own proxy certificate [Globus, 2004] by the command

```
grid-proxy-init.
```

Then, she is able to start the executable called "seepp2grid" with some parameters (see Figure 3). The only parameter that has to be given compulsory is the fully qualified domain name of the grid site on which "seepp2grid" program will start a/some SEE++ server(s):

```
seepp2grid [-help] [options…] GRIDSITE
```

Initially, server processes can be started on only one grid site. If the user does not give any other parameter, then the program assumes that the executable of the SEE++ server is located in the corresponding home directory on the given grid site and it starts only one process of it on the site. Then the "seepp2grid" starts to listen on the default port (8090) for client requests.

The following parameters can be used for the "seepp2grid":

-help	:	To display a sort description about the usage of the program.
-n <*number*>	:	The number of the SEE++ servers that are intended to start on the grid site can be specified by this parameter.
-g <*number*>	:	The maximum number of the (intended gaze pattern's) points whose coordinates are forwarded in a message to the same SEE++ server. This parameter is used to specify the granularity of the data parallelism (see Section 3). Its minimum value is 1. The default is 9 (which is the number of the points in the default gaze pattern used by SEE++, see Figure 2).
-path <*remote path*>	:	The location of the executable of the SEE++ server on the grid site can be specified by this parameter.
-port <*number*>	:	A port number can be specified on which the "seepp2grid" will wait for the connection requests of the SEE++ clients.

For instance, the command

```
seepp2grid -n 9 -g 1 -path /home/local/agrid/ag10022
altix1.jku.austriangrid.at
```

starts 9 processes of SEE++ server located in the directory "/home/local/agrid/ag10022"on the Austrian Grid site "altix1.jku.austriangrid.at".

During the execution of the "seepp2grid", the user can issue some commands additionally:

- By the command **start [-path <*remote path*> -n <*number*>] GRIDSITE**, some new SEE++ servers can be started on the same or on a different grid site similarly as before.
- By the command **exit,** the user is able to shut down all SEE++ servers (started on every grid site) and finish the execution of the "seepp2grid" program.

2.2 Starting Procedure of SEE++ Servers on the Grid

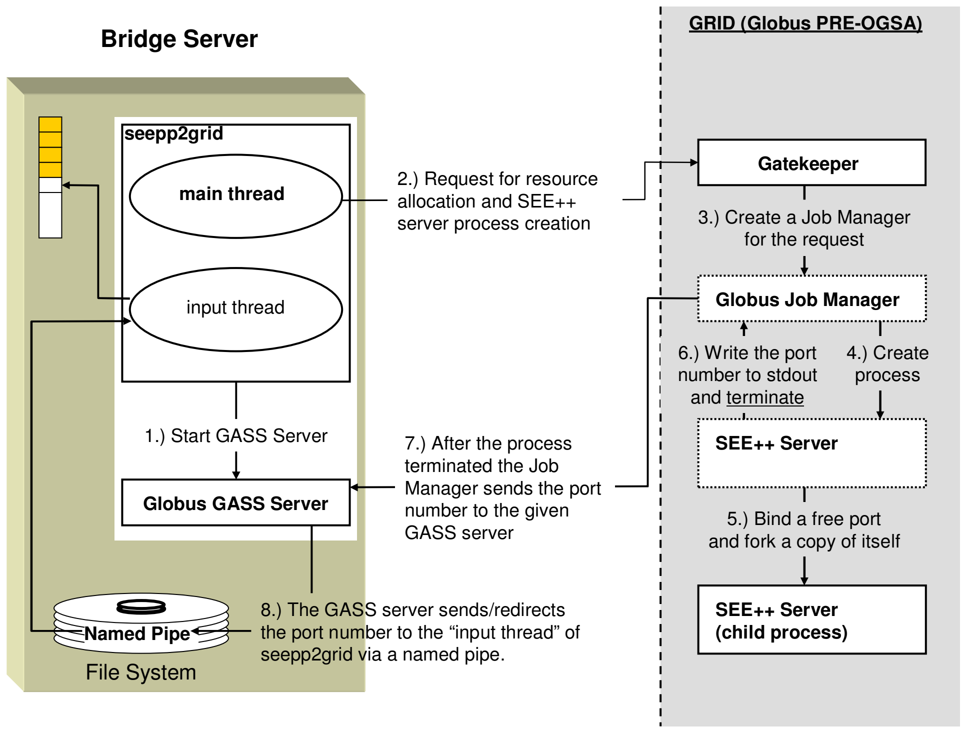

Bridge Server
GRID (Globus PRE-OGSA)
seepp2grid
main thread
input thread
2.) Request for resource allocation and SEE++ server process creation
Gatekeeper
3.) Create a Job Manager for the request
Globus Job Manager
6.) Write the port number to stdout and terminate
4.) Create process
SEE++ Server
1.) Start GASS Server
7.) After the process terminated the Job Manager sends the port number to the given GASS server
Globus GASS Server
5.) Bind a free port and fork a copy of itself
SEE++ Server (child process)
8.) The GASS server sends/redirects the port number to the "input thread" of seepp2grid via a named pipe.
Named Pipe
File System

Figure 4: Starting a SEE++ Server on Globus Site

In the first step, the "seepp2grid" starts a GASS server [Globus, 2004] on the local machine (see Figure 4). To this server, the standard output streams (and standards error streams) of the started SEE++ servers will be redirected by their Globus Job Managers. The GASS server in turn forwards this received information to the "seepp2grid" application via a predefined "named pipe". The Globus GASS server can be configured such that it prints out (to a determined target file) a received line at a time, so multiple senders can access to it safely.

The SEE++ servers can be started on the grid sites as normal grid jobs (see Figure 4). This means, the "seepp2grid" acts as a GRAM client and submits a job in a RSL (Resource Specification Language [Globus, 2004]) expression on a particular grid site through its Gatekeeper. The template of the RSL expression applied for starting a number of SEE++ server processes is the following:

&(executable="<*location_of_the_executable*>/grid-seeppserver")
(count=<*number_of_processes*>)
(stdout="<*contact_string_of_GASS*>/<*location_of_a_named_pipe*>")
(stderr="="<*contact_string_of_GASS*>/<*location_of_a_named_pipe*> ")

When a SEE++ server is started, it attempts to bind a free port in a predefined port range. Then it should send this information back to the "seepp2grid". Unfortunately this is not a trivial task, because it is not possible to communicate through the gatekeeper-connection in "real time". This connection is an https-stream and everything sent through it is placed in the GASS cache. But there is no possibility to force emptying of the GASS cache unless the submitted job terminates or one overflows the GASS cache by writing useless data to it. For solving this problem, we applied a technique that has already been implemented in the glogin [glogin, 2004] for a similar case. According to this, after the server process has bound a free port and sent the port number to the client trough the gatekeeper connection, it forks itself and then terminates. The Gatekeeper perceives the termination of the program and empties the GASS cache.

Fortunately, once the Gatekeeper has started a program on a grid site, then this program will take care of its communication itself. Therefore, the "seepp2grid" will be able to connect to the socket, which is still held active by the previously forked copy of the started SEE++ server.

3 Parallelised Gaze Pattern Calculation

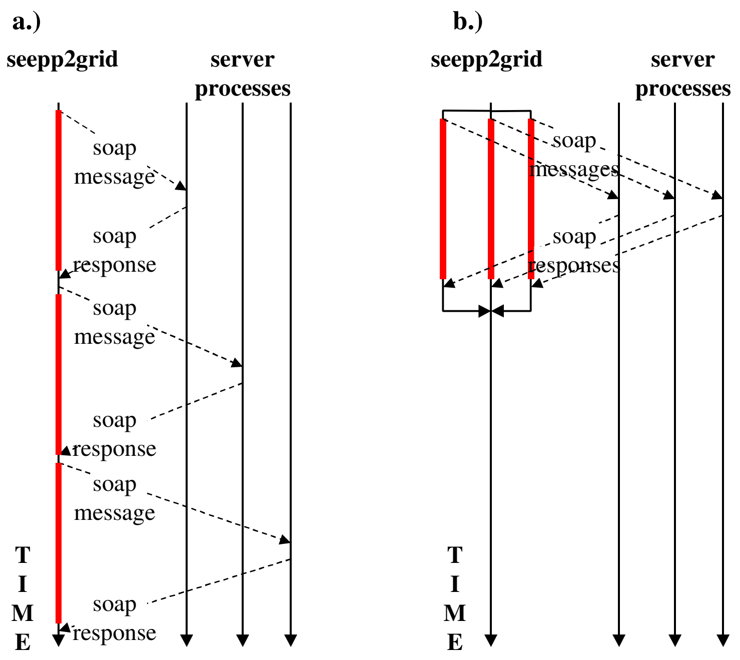


Figure 5: Communication between "seepp2grid" and SEE++ Servers with gSOAP
a.) without Threads **b.)** with Threads

After the SEE++ server(s) was/were started on a grid site and their port number(s) became known, the "seepp2grid" bridge waits for connection requests from SEE++ clients. When the user triggers a gaze pattern calculation on a SEE++ client, the following happens from the point of view of the "seepp2grid" program:

- A message **Calculate_Binocular_Test(*E1, E2, E, I*)** [SEE-GRID Design, 2004] arrives at the "seepp2grid" bridge from a client, where *E1, E2, E* define an eye model (fixing eye, following eye, reference eye) and *I* is a matrix. *I* determines/consists of the points of the current intended gaze pattern used on the client.

1. The "seepp2grid" creates a thread for processing the request of the client.

2. The program allocates a SEE++ server by a simple load balancing mechanism which always chooses either the first idle server or that one which has the least workload. The *Server_Id* of the allocated server is stored in an allocation table.

3. The program copies the data of the first/next *n* pieces of points to another matrix *I'*, where n is the granularity value of the data parallelism determined by the parameter "-g" of the "seepp2grid" (see Section 2.1).

4. The program sends a message **Calculate_Binocular_Test(*E1, E2, E, I'*)** to the allocated SEE++ server.

5. The "seepp2grid" repeats the steps 2 to 4 until each point contained by the matrix *I* is sent to a SEE++ server. The allocated servers for this request are linked one after the other into the allocation table.

   The usual two-way (request response) SOAP messages are performed in gSOAP as "Remote Procedure Call" (RPC). This means, a message cannot be sent after another one within a thread before the response for the first message arrives back (see Figure 5a). Of course, this may block the execution several times and may cause remarkable overhead in the communication. To avoid this, each gSOAP RPC of each kind of two-way message is executed on "seepp2grid" bridge in an independent thread (see Figure 5b).

6. When every such thread terminated, the *Session_Id*-s received as the responses for the **Calculate_Binocular_Test** messages are stored in the allocation table next to the corresponding *Server_Id-s*. Finally, a unique *Allocation_Id* is created from them and sent back to the client as a response for the original **Calculate_Binocular_Test** message (a *Session_Id* itself may be not unique any more, because we may use more than one server).

- A message **Poll_Status(*Allocation_Id*)** [SEE-GRID Design, 2004] arrives at the "seepp2grid" bridge from a client.

  1. The "seepp2grid" creates a thread for processing the request of the client.

  2. It takes all *Server_Id* - *Session_Id* pairs pointed by *Allocation_Id* in the allocation table and sends message **Poll_Status(*Session_Id*)** to each server identified by the corresponding *Server_Id* (in an independent thread).

  3. When every such thread terminated, an average is calculated from the status information *Percentage*-s received from the servers. This value is sent back to the client as a response for the original **Poll_Status** message.

  4. If the received status information indicate that a calculation is finished on a server, then the "seepp2grid" triggers the sending of a message **Poll_Result(*Session_Id*)** to the corresponding server in an independent thread.

The parameter *Session_Id* of this message is the same as the parameter of the **Poll_Status** message sent to the same server before (in step 2).

- A message **Poll_Result(*Allocation_id*)** [SEE-GRID Design, 2004] arrives at the "seepp2grid" bridge from a client.

    1. The "seepp2grid" creates a thread for processing the request of the client.

    2. It collects all calculation results arrived as responses for the messages **Poll_Result(*Session_Id*)** from the corresponding servers, compounds them into one, and sends it back to the client as the result of the requested gaze pattern calculation.

3.1 New Message in "seepp2grid"-server Protocol

For establishing the initial version of SEE-GRID, we added only one message to the communication protocol:

- **One-Way Message**
  Request: Terminate_Server

  By this message, the SEE++ servers running on a grid site can be shut down remotely from the "seepp2grid" bridge. Of course, if we change to Globus 4 WS architecture in the future, this kind of message will become unnecessary.

4 Initial Testing and Benchmarks

For testing the speedup of the current version of SEE-GRID, we compared the calculation time of some typical kinds of gaze patterns in some simple cases. We performed two kinds of tests: local tests within our own network and grid-based tests across the Internet. In the local tests, we used an AMD Dual Opteron 1.6Ghz. The grid-based tests were executed on the Austrian Grid site altix1.jku.austriangrid.at, which contains 64 pieces of Ithanium processors. In the second case, we investigated the effectiveness of the parallelism in different situations where 1, 3, 5 or 9 processes of the SEE++ server are started and the maximum number of the gaze pattern points which are sent together to one process (*granularity value*) is not limited, 3, 2 or 1.

The measured duration times presented below are *only preliminary results* of calculations that executed on the default eye model parameters. An exhaustive benchmark with realistic pathological cases will be performed later.

Each value located in the following tables is the median execution time of 5-7 executions. The actual intended gaze pattern was always calculated first. Then we varied the parameters of the eye model. We modified only the total strength (from 40% to 90%) of one or more eye muscles [SEE-KID, 2004]:

- In the second step, we changed the total strength of the muscle Medial Rectus (MR – internal straight eye muscle) on the following eye.

- In the third step, we changed the total strength of MR together with the total strength of the muscle Superior Rectus (SR –upper straight eye muscle) on the following eye.

- At the last step, we changed the total strength of MR and SR on both eyes.

For measuring, we installed the Ethereal network protocol analyzer [Ethereal, 2004] on the machine where the SEE++ client is executed. By this software, the network traffic between the local machine and the grid portal machine was filtered and each network package sent to or received from the port of "seepp2grid" was captured. After the execution of a test case, the duration time of the calculation can be determined from the recorded capture time of the first sent and of the last received message.

Machine Name	Dual Opteron	Altix 350 (altix1.jku.austriangrid.at)			
Server processes / Max. number of points sent together	1/all	1/all	3/3	5/2	9/1
Calculating intended Gaze Pattern.	3.8442s	2.0691s	1.1250s	0.9709s	0.7616 s
Changing the Total Strength of MR on the following Eye.	12.8413s	6.0777s	2.5515s	2.0007s	1.2263 s
Changing the Total Strengths of MR and SR on the following Eye.	13.1153s	6.2380s	2.5802s	2.0347s	1.2432 s
Changing the Total Strengths of MR and SR on both Eyes	13.5159s	6.4353s	2.6732s	2.1931s	1.3547 s

Table 1: Benchmark Results in case of the Calculation of the Default Gaze Patterns (with 9 points)

On Table 1, the benchmark results can be seen in the case of the calculation of the default gaze pattern (see Figure 2a). This kind of gaze pattern consists of only 9 points.

Machine Name	Dual Opteron	Altix 350 (altix1.jku.austriangrid.at)			
Server processes / Max. number of points sent together	1/all	1/all	3/3	5/2	9/1
Calculating intended Gaze Pattern.	7.3532s	3.6820s	1.8305s	1.4224s	0.9592s
Changing the Total Strength of MR on the following Eye.	26.5566s	12.0769s	7.3513s	4.7013s	2.5761s
Changing the Total Strengths of MR and SR on the following Eye.	28.2197s	12.9893s	8.8241s	5.7248s	3.4361s
Changing the Total Strengths of MR and SR on both Eyes	30.5519s	13.5362s	10.3562s	6.9384s	4.3489s

Table 2: Benchmark Results in case of the Calculation of the Orbit Gaze Patterns (with 21 points)

On Table 2, the benchmark results can be seen in the case of the calculation of the orbit gaze pattern (see Figure 2b). This kind of gaze pattern consists of 21 points.

Machine Name	Dual Opteron	Altix 350 (altix1.jku.austriangrid.at)			
Server processes / Max. number of points sent together	1/all	1/all	3/3	5/2	9/1
Calculating intended Gaze Pattern.	14.5279s	7.5880s	5.8263s	2.9230s	1.6462s
Changing the Total Strength of MR on the following Eye.	56.1237s	25.2703s	17.4387s	10.3963s	7.5788s
Changing the Total Strengths of MR and SR on the following Eye.	63.8593s	27.1793s	18.8115s	12.0023s	9.1101s
Changing the Total Strengths of MR and SR on both Eyes	74.6623s	28.6750s	20.0424s	12.9812s	9.7951s

Table 3: Benchmark Results in case of the Calculation of the Brainstem Gaze Patterns (with 45 points)

On Table 3, the benchmark results can be seen in the case of the calculation of the Brainstem gaze pattern (see Figure 2c). This kind of gaze pattern consists of 45 points.

From the measurement results it can be seen that the calculations became approximately twice fast by the usage of the more powerful hardware of the grid site (despite of the greater communication overhead). Furthermore, more speedup can be expected for the benchmarks in Table 3 if we use more computational processes and decrement the granularity value.

5 Conclusions and Next Steps

In the first phase of the project, we managed to achieve our main goals. We combined the SEE++ software and the software architecture of Austrian Grid and reached a remarkable speedup by applying of a simple data parallelism strategy (despite of the communication overhead).

In the next step, we will deal with the speculative parallelism as it is described in [SEE-GRID Design, 2004] Furthermore, we will change to the Web Service interface of Globus 4 and implement the SEE++ server as a grid service as soon as Globus 4 is deployed on the Austrian Grid resources.

Then we will start to investigate how the Pathology Fitting (and Surgery Simulation) proposed by [SEE-GRID Design, 2004] can be implemented efficiently in a grid environment (most probably by applying some combinatoric searching techniques).

In the future, we may also deal with the development of the automatic exploration of available/free "SEE++ grid services" on the whole grid infrastructure. For achieving this, we will intend to use the information provided by the information service of the Austrian Grid.

References

[AGRID S. Spec., 2004] *Austrian Grid Software Specification,* http://www.austriangrid.at/austriangrid/internal/deliverables/docs/WP_I-1/2004/AG-DI-1-2_v0.3.pdf

[Buchberger, 2004] 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

[Ethereal, 2004] Ethereal Network Protocol Analyzer. http://www.ethereal.com

[Globus, 2004] The Globus Tookit. http://www-unix.globus.org/toolkit/

[glogin, 2004] Herbert Rosmanith and Jens Volkert "glogin - Interactive Connectivity for the Grid" in: Z. Juhasz, P. Kacsuk, D. Kranzlmüller, "Distributed and Parallel Systems - Cluster and Grid Computing", Proc. of DAPSYS 2004, 5th Austrian-Hungarian Workshop on Distributed and Parallel Systems, Kluwer Academic Publishers, Budapest, Hungary, pp. 3-11 (Sept. 2004). http://www.gup.uni-linz.ac.at/glogin/

[gSOAP, 2005] gSOAP 2.7.0 User Guide, 2005. http://www.cs.fsu.edu/~engelen/soap.html

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

[SEE-GRID Design, 2004] Karoly Bosa, Wolfgang Schreiner, Rebhi Baraka, Michael Buchberger, Thomas Kaltofen, Daniel Mitterdorfer. *SEE-GRID Design Overview*. Austrian Grid Deliverable A1c-1, Research Institute for Symbolic Computation (RISC), Johannes Kepler University, Linz, November 2004.

[SEE-KID, 2004] SEE-KID home page, 2004. http://www.see-kid.at

[SOAP, 2003] SOAP Version 1.2 Part 1: Messaging Framework, W3C Recommendation, June 2003. http://www.w3.org/TR/2003/REC-soap12-part1-20030624