SEE-GRID
A Grid-Based Medical Decision Support System for Eye Muscle Surgery*

Károly Bósa†, Wolfgang Schreiner†, Michael Buchberger‡ and Thomas Kaltofen‡

Abstract. *SEE-GRID is based on the SEE++ software system for the biomechanical simulation of the human eye. The goal of SEE-GRID is to extend 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. First, we have developed a grid-enabled version of the simulation of the Hess-Lancaster test, which is a medical examination by which the pathology of the patient can be estimated. Based on this, we work on a pathology fitting algorithm that attempts to give sufficiently close estimations for the pathological reasons of the disorder. Furthermore, we have started to develop a grid enabled distributed database where both real and simulated pathological cases can be collected, sorted and evaluated for improving both the later pathology fitting calculations and the future medical treatments.*

1. Introduction

SEE-GRID is based on the SEE++ [4, 10, 15] 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++ offers the possibility to use a client component for user interaction and visualization and a server component for running the actual calculations; the message protocol SOAP is used for communication between the two components.

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. 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

*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.

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

‡Department for Medical Informatics, Upper Austrian Research (UAR),
email: Michael.Buchberger@uar.at, Thomas.Kaltofen@uar.at

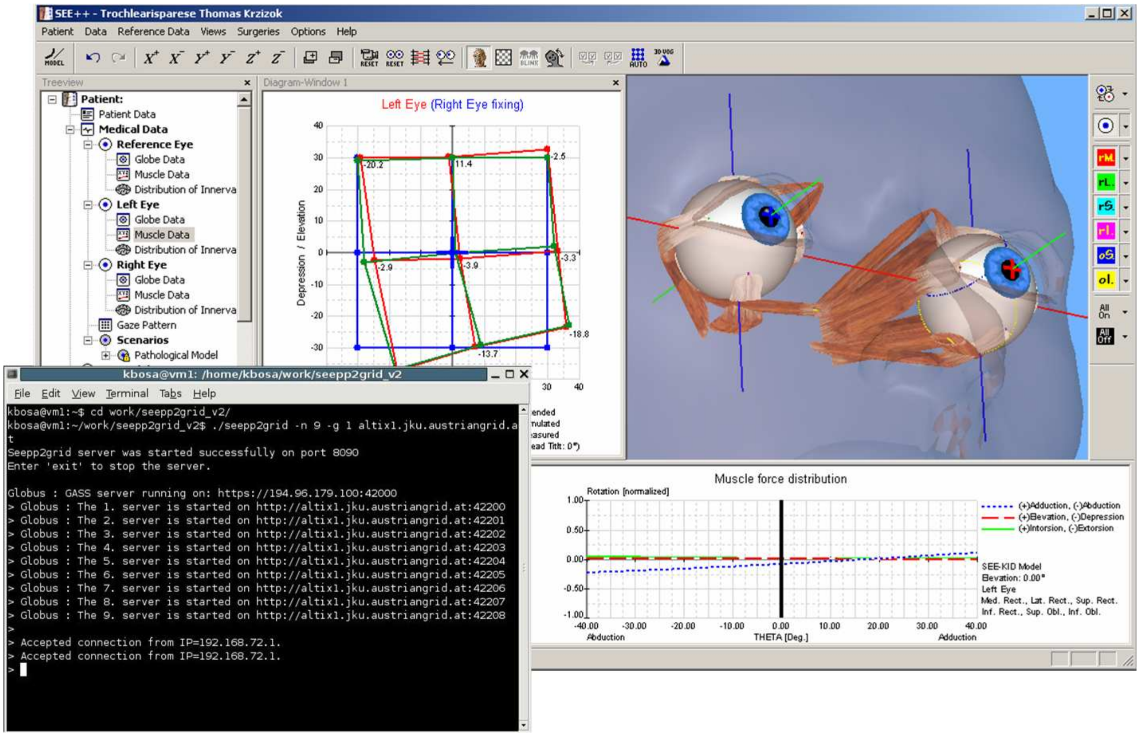

Figure 1. The Output of the "SEE++ to Grid Bridge" and the GUI of SEE++

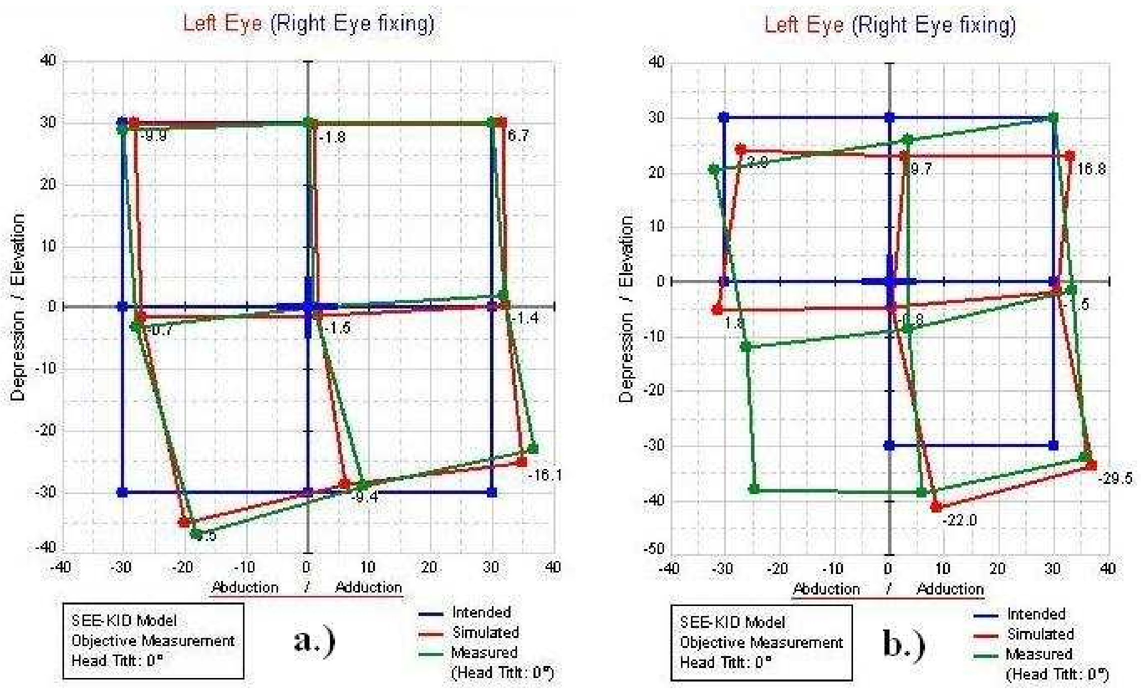

Figure 2. Gaze Patterns in SEE++: Intended (blue), Measured (green) and Simulated (red)

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 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.

The goal of SEE-GRID 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. We have approached this goal in three phases:

- We have implemented a parallel and grid-enabled version of the gaze pattern calculation, see Section 2.
- We are currently developing a grid based medical database for supporting medical treatments based on searching for similar cases, see Section 3.
- We are going to work on a parallel grid-enabled pathology fitter algorithm explained in Section 4. With this, the pathological reason from which a patient suffers may be estimated more precisely in a reasonable time.

The prototype of the SEE-GRID software is developed and executed in the infrastructure of the Austrian Grid [1].

2. Parallel Gaze Pattern Calculation

As a first step of our work, we combined the SEE++ software with the Globus (pre-Web Service) middleware [6]. Our goal was to demonstrate how a noticeable speedup can be reached in SEE++ by the exploitation of the computational power of the Grid.

2.1. The "SEE++ to Grid Bridge"

The initial component of SEE-GRID is the "SEE++ to Grid Bridge" [2], via which the unchanged SEE++ client can get access to the infrastructure of the Austrian Grid (see Figure 3).

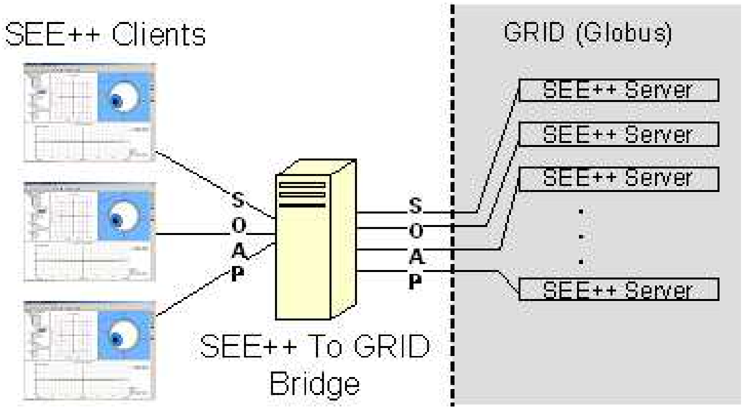

Figure 3. The Current Architecture of SEE-GRID

The “SEE++ to Grid Bridge” acts as a SEE++ server to the SEE++ clients and as a Globus client to the Grid. 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++ server processes are started on the grid sites as normal grid jobs via the Grid Resource Allocation Manager (GRAM). The SEE++ clients can communicate with these processes via the “SEE++ to Grid Bridge” by the same messages as in the original SEE++ software system; the usage of grid resources is completely transparent to them. Messages are encoded in the SOAP protocol which can be also used for communication within the grid.

2.2. Starting Server Processes in the Grid

The ”SEE++ to Grid Bridge” acts as a GRAM client and submits a job described by a Resource Specification Language (RSL) expression to a particular grid site through its gatekeeper.

When a SEE++ server is started, it attempts to bind itself to a free port in a predefined port range and to send this information back to the “SEE++ to Grid Bridge”. Unfortunately this is not a trivial task, because it is not possible to communicate through the GRAM-gatekeeper connection interactively. This connection is an https-stream and everything sent through it is placed in the cache of the “Global Access to Secondary Storage” (GASS, this service manages the I/O files of the applications in the Globus environment). But there is no possibility to force the emptying of the GASS cache unless the submitted job terminates or the GASS cache is filled by writing useless data to it.

For solving this problem, we have applied a technique that was already used in [14] in a similar situation. According to this, after the server process has bound itself to a free port and sent the port number to the client through 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, this program will be able to accept communication requests. Therefore, the “SEE++ to Grid Bridge” will be able to connect to the socket, which is still held active by the previously forked copy of the SEE++ server.

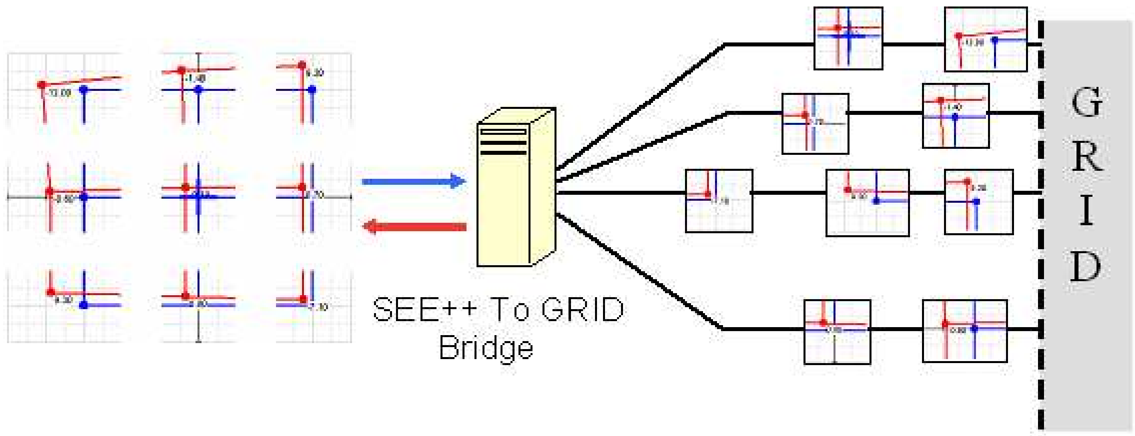

Figure 4. Parallelization of the Gaze Pattern Calculation

2.3. Data Parallelism

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), see Figure 4. Since the calculations of each gaze points are completely independent from each other, there is no communication among the server processes. The maximum number of server processes we used in a session was 45, because gaze patterns used in medical examinations can consist of at most 45 gaze points.

After the SEE++ servers have been started on a grid site and their port numbers have been disclosed, the "SEE++ to Grid Bridge" waits for connection requests from SEE++ clients. When the user triggers a gaze pattern calculation on a SEE++ client, the client sends the actual values of the eye model parameters (see Figure 5) and the information of the intended gaze pattern points in a SOAP message to the "SEE++ to Grid Bridge".

The "SEE++ to Grid Bridge" creates replicas of the received message with the eye model data and distributes the received gaze pattern points among the replicas according to a *granularity value*. This value is given by the user as a command line argument of the bridge and determines the maximum number of gaze pattern points whose data will be sent further together in the same SOAP message to a SEE++ server process.

The "SEE++ to Grid Bridge" distributes the newly created SOAP messages among the SEE++ server processes by a simple static load balancing mechanism. The bridge simply counts the number of the active computational tasks (which were sent in a message, but either their results still have not arrived back or they are not aborted) in the case of each started server process and it always send the next calculation request either an idle or the most slightly loaded SEE++ server process. Of course this solution is less efficient than a load balancing according to the real workload of the CPUs, but it is still sufficient for our current requirements.

As in SEE++, the client regularly sends another SOAP message to the "SEE++ to Grid Bridge" in order to check the status of the gaze pattern calculation. The bridge broadcasts this message to the corresponding SEE++ server processes to receive the status of the calculations of the corresponding gaze pattern points. When the servers returns with the percental status information about the requested

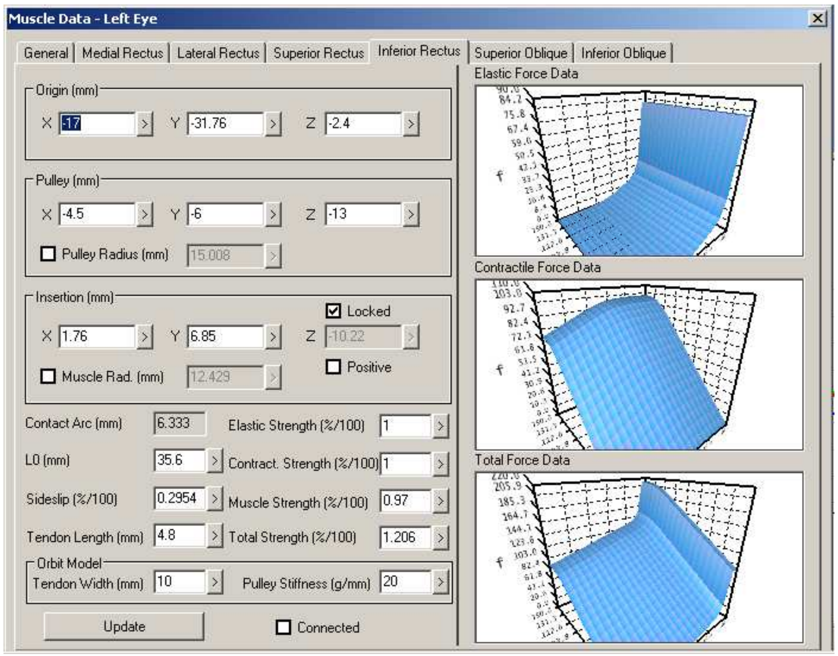

Figure 5. Adjusting the Parameters of the Eye Model in SEE++

computations, the “SEE++ to Grid Bridge” sums the received values and sends back a percental status information about the whole gaze pattern calculation.

The reason why this kind of pulling strategy is applied to receive the status information is that a server process (or a bridge) may be not able to initiate a call back connection to a client (or to a bridge) which resides behind a firewall. Furthermore, since a gaze pattern calculation may take some minutes, the usual two-way SOAP messages cannot be employed for receiving the results, because the sockets used by these messages may not remain open for such a long time. Therefore, the client has to trigger the returning of the results, when the computation is finished.

If the “SEE++ to Grid Bridge” receives the status information that one of the requested computations has been completed, it issues a third kind of message for pulling the results from the corresponding SEE++ server process. The bridge subsequently joins the calculated gaze points to one gaze pattern.

When the client eventually receives the status information that the gaze pattern calculation is done, it sends a SOAP message to the “SEE++ to Grid Bridge” for pulling the results. The bridge then returns the entire pattern.

2.4. Benchmarks

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). We also used three different gaze pattern sizes, namely 9, 21 and 45 points. The granularity value (the number of gaze pattern points sent together to a server processes) was 1 in every case, because our experience indicated that more speedup can be reached with a smaller granularity value than the greater ones (despite of the higher communication overheads).

Each value in Figure 6 depicts the average execution time of 5-7 computations of 45 gaze points.

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

Figure 6. Benchmark Results for Gaze Patterns with 45 points

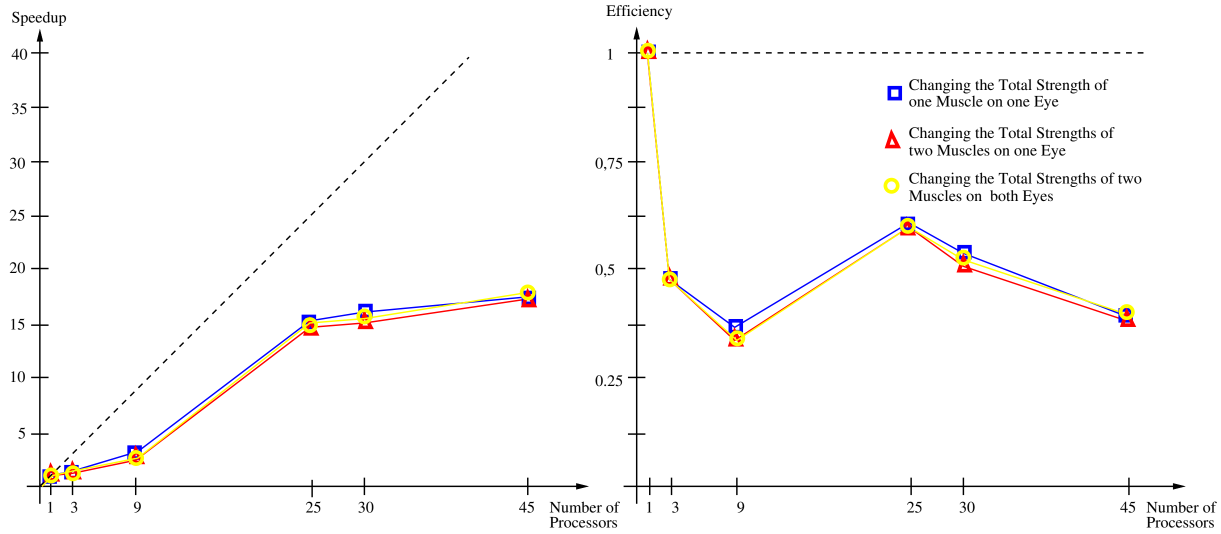

Figure 7. Speedup and Efficiency Diagrams for Gaze Patterns Calculation with 45 points executed on the grid site altix1.jku.austriangrid.at

The test cases 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 Linz.

In case of 25 or more processors, we also started some SEE++ servers (up to 10) on another (distant) grid site called altix1.uibk.ac.at in order to investigate the influence of higher communication latency for the measured values (see the benchmark results in the last three columns in Figure 6). This second grid site resides in Innsbruck and consists of 16 pieces of the same kind of Intel Itanium processors.

The testcases are artificial pathological situations, where in the first case only one eye model parameter was changed on one eye muscle, in the second case the same parameters are modified on two muscles in the same eye and in the last case the same parameters are modified on more muscles in both eyes. As one can see from the measured values, there are no considerable differences between

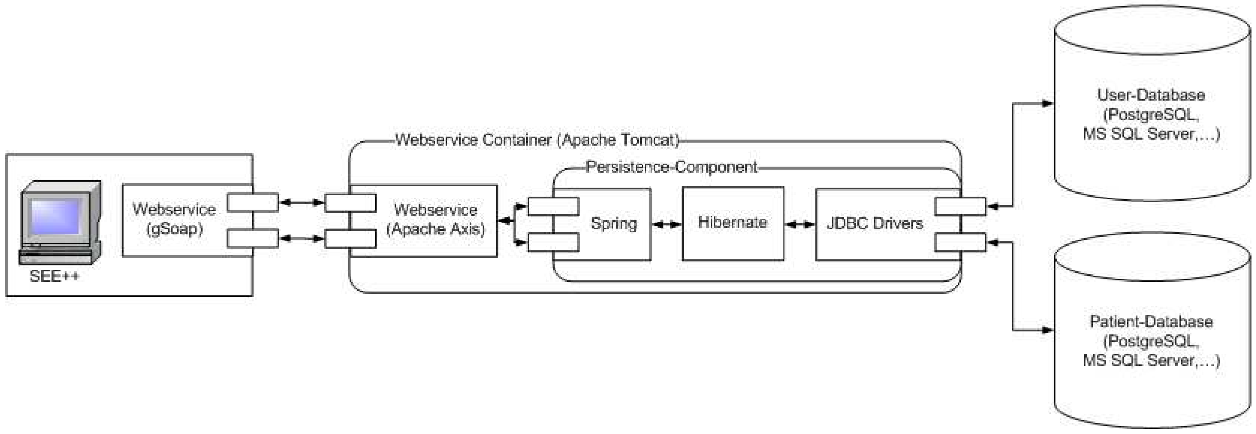

Figure 8. SEE-GRID Database Access Layer

the execution times of the 3 test cases, independently how many parameter values were changed.

In case of 25 or more processors (of the grid site in Linz), we speeded up the simulation of the Hess-Lancaster test by a factor of 14-17, see Figure 7. If we started and used some processes on the far grid site in Innsbruck as well, we got slightly worse results with the speedup limited to a factor 12-14.

3. A Grid Enabled Medical Database

In SEE-GRID, a distributed grid-based database is going to be used for storing and sorting patient data with gaze patterns and eye data. Thus a huge number of medical cases will be easily available for users/surgeons, and for the grid variant of the pathology fitting described in Section 4.

3.1. Prototype Implementation

As a starting point of our work, a database for SEE++ was designed [11] and prototyped as a web service application, see Figure 8. The SEE++ client interacts with the database via the SOAP protocol, the communication protocol of SEE++ was extended by additional SOAP messages used by the database application.

The medical data of SEE++ (patients' data, eye model parameters, measured gaze patterns, results of medical experiments, etc.) are stored in the patient database. The data model does not only support the needs of SEE++, it was designed for supporting general medical databases [11]. Hence, the data model is based on a metamodel, which consists of generally applicable data structures based on various design patterns [5, 13] instead of specific data types used by SEE++.

For exploiting this general structure and keeping the maximum flexibility of the application, we used an *Object/Relational* (O/R) mapping solution (namely Hibernate together with Spring) [8]. The O/R mapping technique ensures that the conversions between the object-oriented and the relational data structures are transparent to the source code.

Since the SEE-GRID database is designed for storing patient records, security is a very important aspect. The user database contains the user authentication and authorization information of the system. The security implementation ensures that every Web Service call is secured appropriately by checking the caller's identity. Furthermore, the persistence component employs many techniques to maximize security like:

- intercepting every Web Service method call and checking the authorization for each method separately;
- supporting certificate-based and username/password-based authentication;
- applying strong encryption of stored user passwords with a SHA-512 salted hash code.

The cryptographic algorithms used in the prototype are based on proven standards to maximize security [9, 16, 17]. Since the security component is not tied to the persistence component, it can be maintained separately and also used for other purposes.

Currently, the Web Service functionality on the database side is provided by the Apache Axis framework. We plan to supplement it by a grid-enabled database interface component in a later phase (see Section 3.2.).

3.2. Grid Enabled Version

Our next steps concentrate on developing a distributed grid-enabled database system that allows SEE-GRID to give efficient support to "Evidence Based Medicine". This grid-based database also has to be able to perform various *data mining* algorithms on its data sets.

According to our scenario, the doctors produce data for their local databases by the manual insertion of patient data. The data sets may be collected in a grid database by automatic transfer of medical data (like measured gaze patterns, eye model parameters, etc.) entered in local databases as well as by automatic insertion of the computed simulation data (the computed gaze patterns are never stored in databases, because these simulated patterns always depend on the biomechanical eye model used by the SEE-GRID software system; however they can be recalculated/recovered from the stored eye model parameters).

To establish this proposed 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 Seamless Data Access Middleware* (G-SDAM) [7] developed by the Institute for Applied Knowledge Processing (FAW) or on the *Web Service Resource Framework* (WSRF) 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.

G-SDAM is capable to add or remove data sources/resources at runtime. Its other core features are the possibility to distribute and launch a query on all participating distributed data source and then to collect and compose the query results received from various data sources into one query result. To a client the query seems to be processed in a central database. The implementation of the G-SDAM architecture is not yet complete, such that we may only use a simplified version of it. This will yield the previously mentioned advantages without the support of nodes containing heterogeneous data structures.

The other possibility is to deploy data (re)sources as grid services in Globus 4. In this case, we must develop a parallel/distributed search algorithm so that we will be able to access and collect the desired information from the distributed database. For achieving this, we plan to utilize some of the features of WSRF, like

- *Resource Properties* for discovering the available data source nodes,
- *Stateful Services* for performing queries and for submitting some computational jobs (see Section 4.2.),
- *Notification* either for reporting the changes in the database triggered by different operations (e.g.: insert, update, delete) or for informing the client about the status of the ongoing/finished computations (see Section 4.2.) and
- *BaseFault* for handling the reported faults during the Web Service invocations.

Since both the G-SDAM and the Web Service Architecture in Globus 4 are able to communicate via the SOAP protocol with other grid-based applications, our database implementation is flexible enough for later adaptation.

4. Grid-Based Pathology Fitting

Pathology fitting is an automatic modification of a subset of the patient's eye model parameters (see Figure 5) until the calculated gaze pattern matches the measured one. Unfortunately, a gaze pattern does not uniquely determine the values of eye model parameters. Further difficulties are that the gaze pattern 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).

Therefore, the pathology fitting either has to produce all possible solutions and present them to the doctors (who can choose the correct one), or the doctors must give additional input data which are based on their own experiences and/or which are derived from some other medical investigations (e.g. choosing the affected eye(s), assuming the syndrome, measuring the eye muscle forces, etc.).

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 gives an approximation of the correct solution. This heuristic consists of a list of particular eye model parameters in a specific order representing the weight by which the parameters affect the presumed strabismus syndrome. This list specifies a strict sequential modification order for the pathology fitting; modifying the parameters in this order excludes most incorrect solutions. The heuristic is determined by the user/doctor from her own experiences and from other medical examinations.

Pathology fitting is essentially a non-linear optimization problem in a multidimensional parameter space. However, the desired solution may not always be the global minimum, for instance the previously described heuristic sometimes leads the algorithm to some desired local minimum.

```
/*
Eye  : Initial Eye Model
M    : Measured Gaze Pattern
H    : Heuristic
*/
pathologyFitting(Eye, M, H)
  E1 := Eye
  C1 := gazePattern(E1)
  if C1 matches M  return E1

  loop
    p := nextParameterVariation(H)
    if p is equal to NULL  return E1
    E2 := optimization(p, E1, M)
    C2 := gazePattern(E2)
    if C2 fits M better than C1
      E1 := E2
      C1 := C2
      if C1 matches M  return E1
```

```
/*
p    : a Type of Eye Model Parameter
E1   : Eye Model
M    : Measured Gaze Pattern
*/
optimization(p, E1, M)
  v = vector of the p parameter values
          on the six eye muscles in E1

  loop
    C2 := gazePattern(E1, v)
    if |M−C2| < ε   return  E1
    v2:= OptimizationStep(v, |M−C2|) /* with Jakobian
          and Hessian matrix calculations */
    Update parameter p in E1 by v2
    v := v2
```

Figure 9. A Draft of the Pathology Fitting Algorithm

The pathology fitting process receives as its input an initial eye model, the measured gaze pattern and the heuristic. Roughly, the pathology fitting works in the following way (see Figure 9):

- On the highest level, the algorithm selects the different kind of eye model parameters contained by the heuristic one by one. The eye model is updated based on the calculated one if and only if the fitting of the currently modified parameter yields any improvement (its calculated gaze pattern matches the measured gaze pattern better).
- On the lower level, a non-linear optimization algorithm modifies the model parameters selected by the heuristic in several iterative optimization steps.
- On the level of the optimization steps, the algorithm performs some computations by which it tries to determinate the next improvement. At the end of each optimization step, a gaze pattern is calculated with the modified eye model data for the evaluation of the improvement.

4.1. Initial Studies

We have already extended the pathology fitting component of SEE++ by parallel gaze pattern calculation [3], 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.

Based on this initial result, we have further evaluated how to parallelize the pathology fitting algorithm itself on various levels of the algorithm:

1. On the highest level, the different kinds of eye model parameters have to be modified in the specific order given by the heuristic. Hence, the only possibility for parallelizing the algorithm on this level is to find a non-sequential heuristic instead of the current one.

2. At the medium level, the applied non-linear optimization algorithm (Levenberg-Marquard) itself is an iterative algorithm where in each step a (Jacobian and Hessian) matrix is computed which can be speeded up by parallel algorithms that apply domain decomposition methods [12]. However in our case, the pathology fitting uses matrices that are too small to profit from this strategy (each matrix computation takes less than 1 second).

3. Another approach would be the separate optimization of the eye muscles which can be performed concurrently on some processors. In case of a healthy/ideal eye, the 6 eye muscles are working in pair [4] (the pairs are independent from each other). But in a diseased eye, it cannot be predicted whether an eye muscle is independent from another one. Therefore, the optimization algorithm has to modify the eye model parameters of the six eye muscles of one eye together.

Since from these investigations a straight-forward parallelization of the current optimization algorithm does not seem very promising, we have started to investigate radically different optimization approaches (like simulated annealing) that may offer more potential.

4.2. The Proposed Grid-Based Pathology Fitting

Rather than for speeding up a single pathology fitting process, we may use the grid to perform *multiple* pathology fitting processes and thus improve the existing algorithm as follows: 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. In the SEE-GRID database, a huge number of medical cases will be available for users/surgeons, which can also be utilized for the purpose of the proposed parallel pathology fitter:

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).

This strategy requires the computational power of the grid, since numerous similar cases may exist in the database. As a consequence:

- we may get better solutions than in the case of the existing algorithm;
- we may get more than one solution relevant to the actual pathological situation of the patient;
- the execution of the solutions may take less time, since we have good estimations at the very beginning.

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

5. Conclusions

By our developments, SEE-GRID may become an efficient tool for supporting and improving the medical treatment of strabismus. Based on the results and investigations described in this paper, our ongoing research work has three main directions:

Implementation of the grid-based database We currently work on the adaptation of the existing prototype implementation of the SEE-GRID database to the Grid.

Algorithmic improvements We have started to investigate different strategies and possibilities in order to achieve a better understanding of the fundamental pathology fitting algorithm and thus to devise new, faster sequential or parallel algorithms.

Development of a grid-based pathology fitting method We are going to implement a variant of pathology fitting that applies a grid-based parallel search technique to find cases in the SEE-GRID database that are similar to measured patient data; using these cases as starting points, the method executes multiple independent pathology fitting processes on the grid.

Later we plan to extend the SEE-GRID medical decision support system by a surgery fitting algorithm that is able to give suggestions to the doctors for the optimal treatment of patients.

6. Acknowledgements

The G-SDAM framework mentioned in Section 3.2. is developed by the Institute for Applied Knowledge Processing (Institut Für Anwendungsorientierte Wissensverarbeitung — FAW) as a partner of the SEE-GRID project.

References

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

[2] Karoly Bosa, Wolfgang Schreiner, Michael Buchberger, Thomas Kaltofen, *The Initial Version of SEE-GRID*, Austrian Grid Deliverable A1c-1-2005, Research Institute for Symbolic Computation (RISC), Johannes Kepler University, Linz, March 2005.

[3] Karoly Bosa, Wolfgang Schreiner, Michael Buchberger, Thomas Kaltofen, *A Prototype of the SEE-GRID Pathology Fitter*, Austrian Grid Deliverable A1c-3-2005, Research Institute for Symbolic Computation (RISC), Johannes Kepler University, Linz, July 2005.

[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] Martin Fowler, *Analysis Patterns: Reusable Object Models*, Addison-Wesley, 2004.

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

[7] *A Report on a Unified Grid-aware Access Layer for SEE-GRID Data Sets*, Austrian Grid Deliverable M-4aA-1c, FAW Institute and RISC Institute, Johannes Kepler University, Linz, August 2005. `http://www.faw.uni-linz.ac.at`

[8] *Hibernate* home page, 2005. `http://www.hibernate.org/`

[9] Michael Howard and David LeBlanc, *Writing Secure Code*, Microsoft Press, 2nd edition, 2003.

[10] 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`

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

[12] N. N. R. Ranga Suri, Dipti Deodhare, P. Nagabhushan, *Parallel Levenberg-Marquardt-Based Neural Network Training on Linux Clusters*, ICVGIP 2002, 3rd Indian Conference on Computer Vision, Graphics and Image Processing, Ahmadabad, India.

[13] *NHS Healthcare Modelling Programme*, 1995,
`http://www.standards.nhsia.nhs.uk/hcm/index.htm`.

[14] Herbert Rosmanith and Jens Volkert *glogin - Interactive Connectivity for the Grid*, 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/`

[15] SEE-KID home page, 2005. `http://www.see-kid.at`

[16] Douglas Schmidt, Michael Stal, Hans Rohnert, Frank Buschmann, *Pattern-Oriented Architecture*, Vol. 2: Patterns for Concurent and Networked Objects, John Wiley and Sons Ltd, 2000.

[17] Bruce Schneier, *Cryptanalysis of SHA-1*, 2005.
`http://www.schneier.com/blog/archieves2005/02/`