AUSTRIAN GRID

SEE-GRID Design Overview

Document Identifier:	**AG-DA1c-1_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	Wolfgang Schreiner	RISC	30.11.2004	
Verified by				
Approved by				

Document Log

Version	Date	Summary of changes	Author
1.0	2000-10-01	Initial Version	See cover on page 3

SEE-GRID Design Overview

Karoly Bosa
Wolfgang Schreiner
Rebhi Baraka

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

Michael Buchberger
Thomas Kaltofen
Daniel Mitterdorfer

Department for Medical Informatics
Upper Austrian Research (UAR)
{Michael.Buchberger,Thomas.Kaltofen}@uar.at
Daniel.Mitterdorfer@fh-hagenberg.at

December 10, 2004

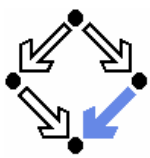

TABLE OF CONTENTS

1 Abstract

This document describes the initial design of the SEE-GRID software in the Austrian 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 Upper Austrian Research and the Upper Austria University of Applied Sciences; it consists of a client component for user interaction and of a server component that runs various computations (currently "Hess Diagram Calculation" and "Pathology Fitting", in the future also "Surgery Simulation"). The goal of SEE-GRID is to adapt and to extend SEE++ in several steps to make use of the grid:

1. The simulation component is transformed to a grid service.

2. "Hess Diagram Calculation" is parallelized and distributed over multiple grid nodes.

3. "Pathology Fitting" is parallelized and distributed over multiple grid nodes.

4. A grid database of "gaze patterns" is established with patterns derived from measurements and calculations. For querying this database for wanted patterns, many instances of the core problem of "pathology fitting" have to be performed; we will parallelize these queries and distribute them over multiple grid nodes.

Based on above results, SEE++/SEE-GRID may be improved in order to provide a future "surgery simulation" component that gives the doctor recommendations for how to perform a surgery that corrects a particular impaired eye.

2 SEE++ Fundamentals

SEE++ is a software for the biomechanical, interactive 3D simulation of the human eye, its orbita and its muscles, to simulate common surgical eye muscle operations (transposition, shortening, splitting, etc.) in a graphic interactive way that is familiar to an experienced surgeon. This software was developed in the frame of the SEE-KID project by Upper Austrian Research and the Upper Austria University of Applied Sciences [SEE-KID 2004, Buchberger 2004, Kaltofen 2002].

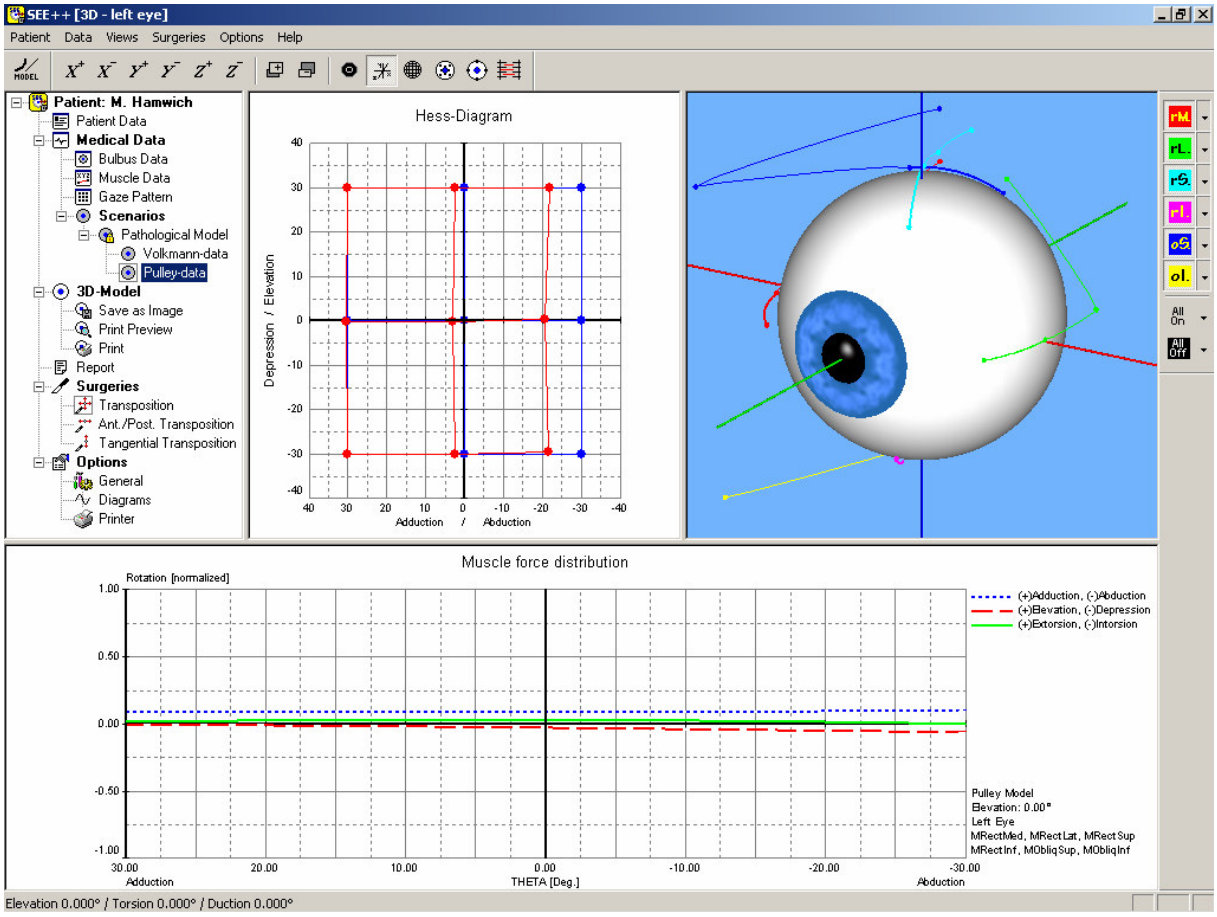

Figure 1: A SEE++ screenshot

For illustrating the functionality of SEE++ (see Figure 1 for a screenshot of the GUI) in a rather simplified way, we introduce the following notions:

- An *eye model* is a set of parameter values that describes the biomechanical structure of a particular human eye. We use two independent models, one model *L* for the simulation of the left eye and one model *R* for the simulation of the right eye. However, there are also influences that affect both eyes in common. These influences are still poorly understood and currently captured by a third model *E* of a hypothetical "reference eye". *E* serves as an additional parameter in the simulation of the left eye as

well as in the simulation of the right eye. We also call a triple (*L*, *R*, *E*) collectively "the eye model" (of a patient). The exact eye model of a living human cannot be computed by measurement, rather it has to be deduced by indirect methods.

- A *gaze pattern* (or *Hess diagram)* is an eye pair's image of a set of reference points in the plane. Seen by a healthy pair, these points form a regular pattern (the "intended gaze pattern" *I*, drawn blue in Figure 2). Seen by an impaired pair, the pattern is distorted (drawn red in Figure 2). Actually, there are two gaze patterns: one, if the left eye "fixes" the reference points and the right eye "follows" the left eye, and one if the right eye fixes the reference points and the left eye follows. Consequently, the gaze pattern depends on a model *E1* of the "fixing eye" and on a model *E2* of the "following eye" (which may be *L* or *R*) and on the model *E* of the "reference eye". The gaze pattern of a living human can be determined by measurement (the "measured gaze pattern" *M*) or by simulation from the eye model (the "calculated gaze pattern" *C*). The simulation is faithful, if *C* matches *M*.

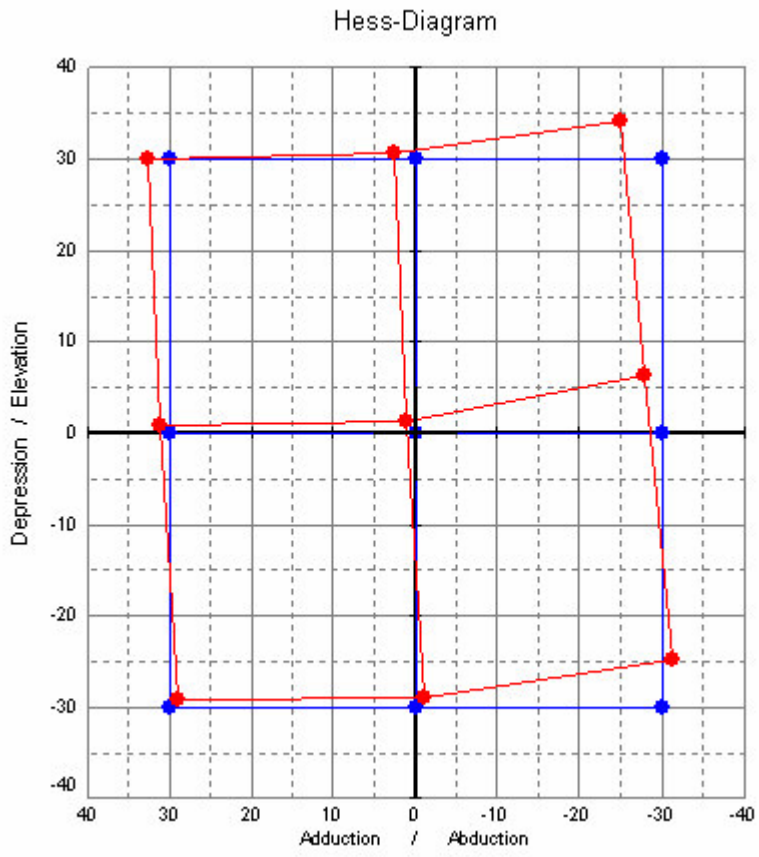

Figure 2: A Gaze Pattern

Basically, there are two core problems that SEE++ deals with:

- The calculation of a gaze pattern from an eye model (the "forward problem").
- The calculation of an eye model from a gaze pattern (the "inverse problem").

As usual, the inverse problem is the mathematically more difficult and the computationally more time-consuming one.

More specifically, SEE++ addresses the following computations:

1. Hess diagram calculation,
2. pathology fitting,
3. surgery simulation.

The first computation solves the forward problem, the two other computations solve (different instances of) the inverse problem. The first computation is fully implemented, there also exists a prototype implemention for the second computation. The third computation is not yet implemented but still under investigation. We are now going to describe the three problems solved by these three computations.

2.1 Hess Diagram Calculation

Input:

(E1,E2,E) … an eye model (fixing eye, following eye, reference eye)
I … the intended gaze pattern

Output:

C … the (calculated) gaze pattern of *(E1,E2,E)* for the points in *I.*

The typical use of this calculation is the following:

1. "Pathology Fitting": the doctor starts with a default eye model and modifies by "trial and error" the model parameters until he finds an eye model *(E1,E2,E)* where the calculated gaze pattern *C* (roughly) matches the measured gaze pattern *M* of a patient.

2. "Surgery Simulation": Further on, the doctor modifies again by trial and error the parameters of *(E1,E2,E)* until he finds by a "minimal set of parameter changes" an eye model *(E1',E2',E')*, such that the calculated gaze pattern *C* (roughly) matches the intended gaze pattern *I*.

The difference between *(E1,E2,E)* and *(E1',E2',E')* (i.e. the set of parameter changes) describes the surgery that has to be performed on the patient. If the simulation is faithful, after the surgery the newly measured gaze pattern *M'* of the patient will match *C* and thus *I*.

Each modification of the parameter set requires the re-calculation of the gaze pattern, i.e., gaze patterns are very frequently calculated. For a particular eye model, often both variants of the gaze patterns (both permutations of the fixing eye and of the following eye) are calculated.

Each calculation currently takes about 10 seconds or so, ideally a calculation would go on in "real time", since the doctor has to perform many calculations.

Currently, a gaze pattern consists of 9 points. With faster calculations, also gaze patterns with more points could be used.

2.2 Pathology Fitting

Input:

M … the measured gaze pattern (potentially only partial)
I … the intended gaze pattern
c … the "connection" between *M* and *I* (i.e., a mapping of points of *M* to points of *I*).
(D1,D2,D) … an eye model (fixing eye, following eye, reference eye)
S … a "parameter selection"

Output:

(E1,E2,E) … an eye model (fixing eye, following eye, reference eye) whose calculated gaze pattern (roughly) matches *M* (for the points of *I* identified by *c)* and that is identical to *(D1,D2,D)* except for the parameters selected by *S*.

The purpose of this calculation is to automate the first step of the process described in Section 1.1. Ideally, from the measured gaze pattern *M* alone, the model *(E1,E2,E)* of the patient could be constructed. However, there are several problems that make the inclusion of additional parameters necessary:

1. *M* may be only partial (i.e. some reference points could not be measured). Thus also *I* and an injective mapping *c* of the points of *M* to points of *I* are provided as input.

2. The "search space" is too large, i.e., the calculation would take much too long if it would take the space of all possible parameter values into account. Therefore, the doctor provides as an additional input a "standard model" *(D1,D2,D)* and a "selection" *S* of those parameters where he believes the "patient model" actually differs from the standard model. The standard model and the parameter selection can be based on literature, experience, or MRI data (images of the patient's eyes). The search space is thus deliberately restricted from more than 100 parameters to 30-40 parameters.

3. While an eye model uniquely determines a gaze pattern, the inverse does not hold, i.e. many eye models yield the same gaze pattern; furthermore, most of these eye models do actually *not* match the patient's real physiology. If the parameter space is sufficiently (and correctly) restricted as described above, the number of possible wrong results is reduced and the calculation is also to some extent "guided" (because the software gradually modifies the initially given parameter values to minimize the differences between the computed gaze pattern and the measured gaze pattern); still there exist ambiguity in the results and the returned result may not be the desired one.

The computation of an eye model that yields a given gaze pattern is a very time-consuming minimization problem which can be speed up by grid computing by parallelization of the optimization algorithm and by searching for multiple solutions rather than being content with the first one encountered (thus the doctor may take into account multiple results).

Furthermore, we may speed up the search and improve its results by using a database of known pairs *(E,P)* of eye model *E* and gaze pattern *P* (of previously computed patterns or patterns derived from patient measurements). If we find a pair where *P* is "similar" to the measured gaze pattern *M*, then the corresponding *M* may serve as a hint for how to automatically choose good inputs *(D1,D2,D)* and *S*. As for the problem of ambiguous results, such good inputs would make the selection of a "wrong" eye model more unlikely.

2.3 Surgery Simulation

Input:

I … the intended gaze pattern
(E1,E2,E) … an eye model (fixing eye, following eye, reference eye)

Output:

(E1',E2',E') … an eye model (fixing eye, following eye, reference eye) whose gaze pattern (roughly) matches *I* and that was constructed from *(E1,E2,E)* by a minimum amount of changes.

The purpose of this calculation is to automate the second step of the process described in Section 1.1. The input is the eye model of the patient (the result of either manual or automatic "pathology fitting"), the output is the desired eye model. The difference between both models describes the surgery to be performed on the patient.

The mathematical problem is essentially the same one as that of Section 1.2 (to determine an eye model from a gaze pattern), but with an additional side-constraint: since each modification of a parameter, describes a physical modification of the patient's eye, only a minimum amount of changes should be performed to *(E1,E2,E)* to yield *(E1',E2',E')*. The additional input *(E1,E2,E)* is therefore not only to improve the heuristics of search but also required by the problem domain itself: from all those eye models that yield *I*, the one is to be chosen that is closest to the patient's actual eye model.

This calculation would profit in the same way from grid computing (speeding up individual calculations, allowing to search for multiple results) and from a data set of know pairs of eye models and gaze patterns (derived from archives of surgery results).

3 The SEE++ Software

3.1 Architecture

SEE++ has been recently decomposed into 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 [SOAP, 2003]. It is assumed that the client may run behind a firewall that only allows outgoing connections, i.e., all connections to the server have to be initiated by the client.

As for the client component, there are actually two clients: the default is a Windows standalone client written in C++ (the "SEE++" client) which connects as a SOAP client to the SOAP server. However, there also exists a Java client applet that can be embedded into Web pages (the "JSeeppApplet" client). This applet cannot directly connect to the SOAP server, since the default Web browser security restrictions only allow an applet to connect to the machine from where the applet was loaded. Thus the applet connects via RMI to an "RMI2Soap" bridge server process on the Web server machine; this bridge server translates RMI requests into SOAP requests and acts as a SOAP client to the SOAP server.

The SOAP clients need not directly connect to a SOAP server but can also connect to a "SOAP message router" which distributes calculation requests among a cluster of SOAP servers and thus balances calculation load among multiple machines.

A diagram that sketches the overall system structure is shown on in Figure 3.

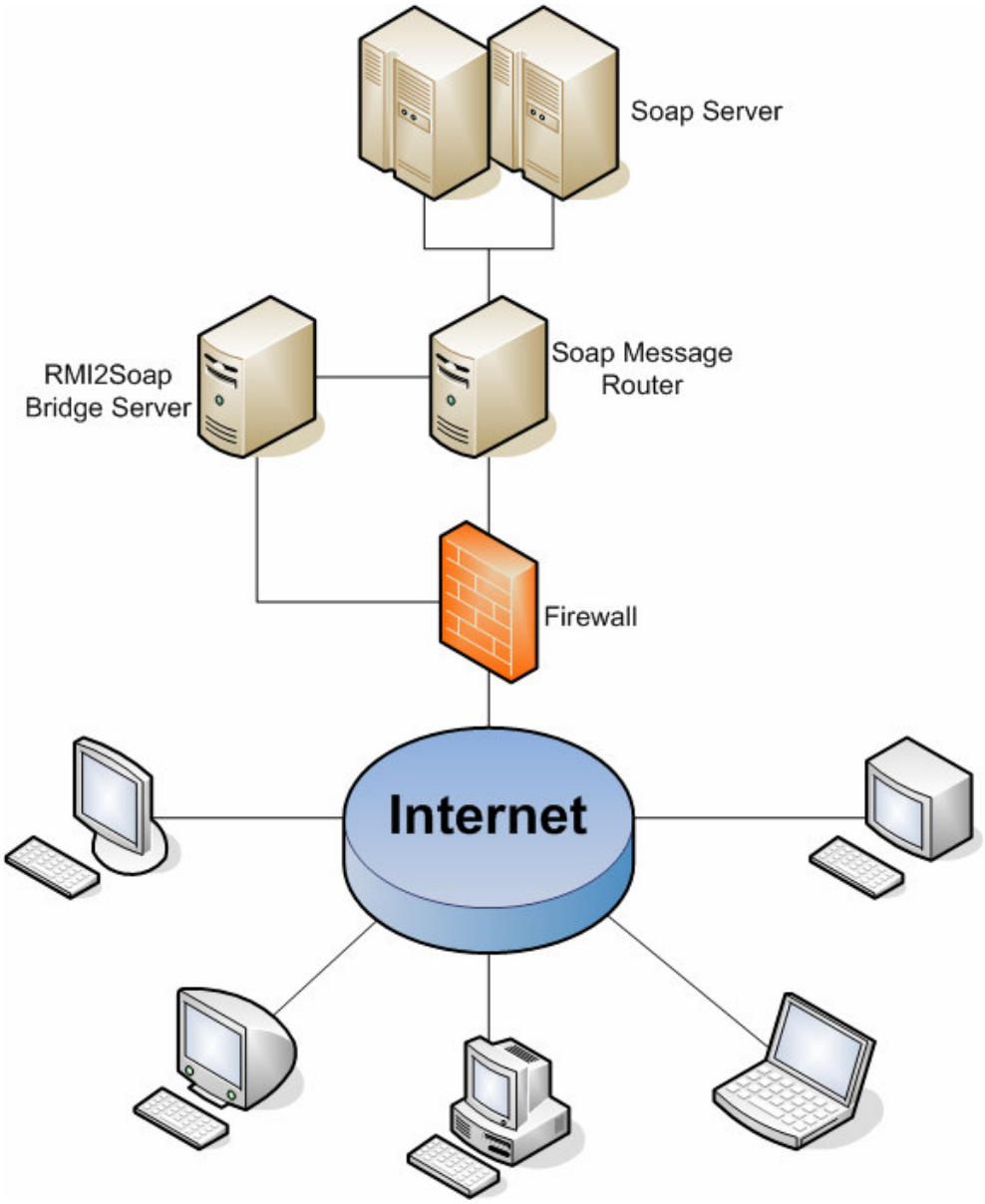

Figure 3: The SEE++ Software

3.2 Client-Server Protocol

After startup, the SEE++ server waits for connection requests from SEE++ clients. When the user triggers in a SEE++ client the calculation of a Hess diagram (see Section 1.1), the following dialogue takes place.

Two-Way Message

Request: Calculate_Binocular_Test(*E1, E2, E, I*)
Answer: *Session_Id*

The client sends the eye model and the intended gaze pattern to the server; the server starts a thread for the calculation of the gaze pattern from the eye model and returns an identifier for this calculation. The server is now ready to accept other calculation requests.

N≥1 Two-Way Messages

Request: Poll_Status(*Session_Id*)
Answer: (*Percentage*, *Terminated*)

The client polls the server for the progress of the calculation. The server returns a tuple of the percentage of the calculation completed (the ratio between the number of the reference points already calculated to the total number of points) and a flag that is set to true if the total gaze pattern has been completed.

Two-Way Message

Request: Poll_Result(*Session_Id*)
Answer: (*C, Visibility, Innervations*)

The client asks the server for the result of the calculation (this is only legal, if a previous status poll has signalled the termination of the calculation). The server responds with a triple of the calculated gaze pattern, the visibility status of each point in the gaze pattern (some points may have turned out impossible), and the innervations for each point of the gaze pattern (the innervations are the forces applied to each eye muscle to focus on the reference point). The server then releases the thread resources; the session identifier is thus not valid any more.

If a calculation result is not polled for a certain amount of time after the termination of the calculation (because the client has crashed or permanently lost connection), the thread resources are automatically released.

In addition, the following messages may be exchanged.

One-Way Message

Request: Terminate_Thread(*Session_Id*)

The client signals that it has lost its interest in a previously started calculation. The server is free to release its thread resources; the session identifier is thus not valid any more after this message.

Two-Way Message

Request: Get_Default_Data()
Answer: *Eye*

The client requests from the server a standard eye model (i.e. a set of standard parameter values) for further use; the server responds with such a model (for a single eye, not an eye triple). This message is typically issued when the client is started.

3.3 Algorithm for Hess Diagram Calculation

We sketch the algorithm for the calculation of a Hess diagram (see Section 1.1).

hessDiagram(E1, E2, E, I):

for each intended gaze position i∈I **do**
C[i] := position(E1 ,E2 ,E, i)

return C

position(E1, E2, E, i):

A: Compute from i and E the 3D gaze position i'.

B: Compute from i' and E1 the innervations I1 of the fixing eye.
C: Compute from I1 and E the intended gaze position i1 of the fixing eye.

D: Mirror i1 to yield the intended gaze position i2 of the following eye.

E: Compute from i2 and E the innervations I2 of the following eye.
F: Compute from E2 and I2 the gaze position c of the following eye.

return c

The computation of the 3D gaze position (step A) involves linear minimization, the computations of the innervations (step B and E) requires non-linear minimization; the later is the most time-consuming part.

In the actual software, both c and I2 are returned by the "position", and both the gaze pattern and the list of innervations is returned by "hessDiagram".

4 The SEE-GRID Architecture

In the following, we describe the architecture of the software to be developed in the frame of the SEE-GRID project. We will base our developments on the Globus tookit version 3.2 web service interface [Globus,2004] which will be also the basis for the upcoming Globus 4 toolkit. We will embrace this new version as soon as it becomes available and is deployed in the Austrian Grid.

4.1 Bridging SOAP and Globus

The SEE++ client (Windows application or Java applet) shall not be affected by the proposed grid extension. Therefore we introduce (in analogy to the RMI2Soap bridge described in Section 2.1) a SOAP to Globus bridge (the "JSoap2GridServer") which will act as a SOAP server to the SEE++ client and as a client to the Globus toolkit, see Figure 4.

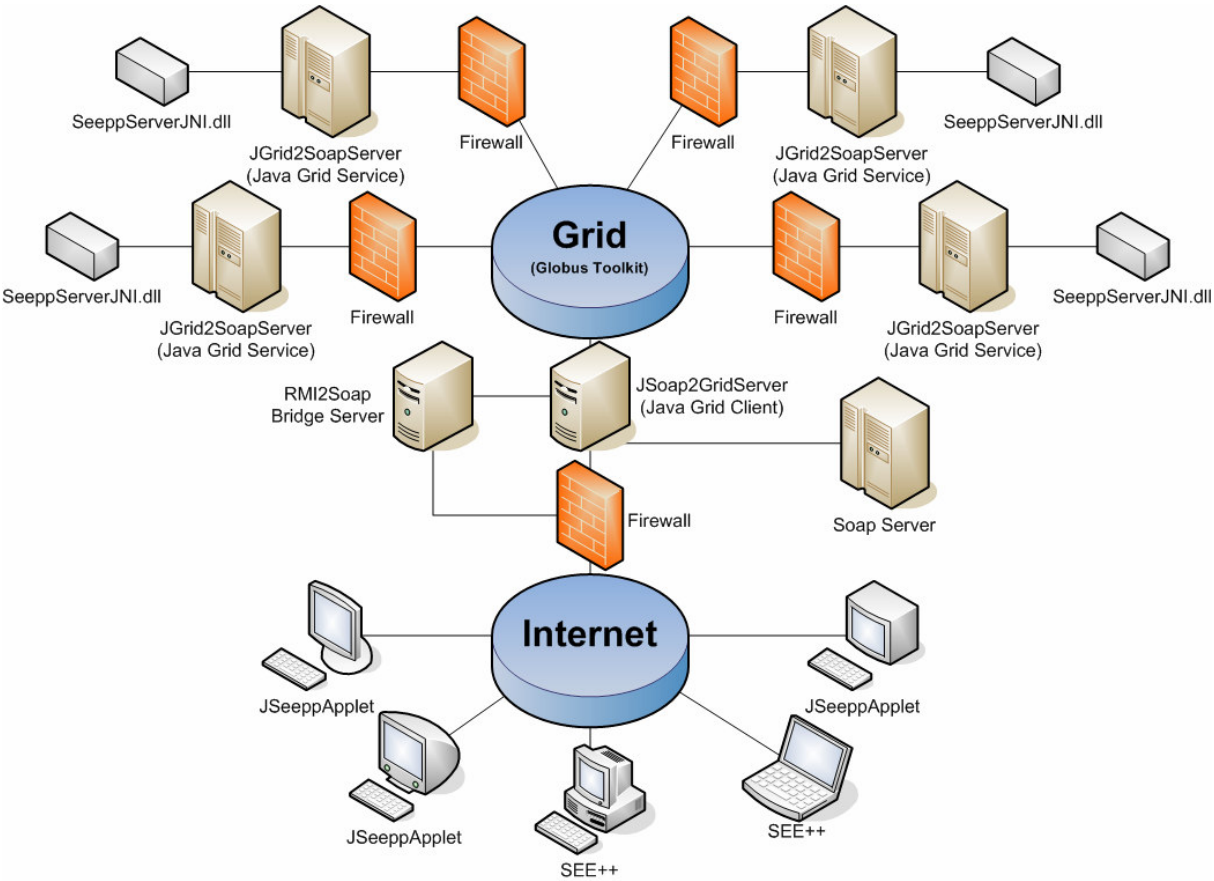

Figure 4: Bridging SOAP and Globus

A technical complication arises from the fact that SEE++ server is implemented in C++ and communicates with the client by exchanging C++ data structures in SOAP encoding. The

Web Service (WS) interface of the current Globus 3.2 toolkit only supports the implementation of services in Java; a mapping of C++ data structures to Java data structures would be difficult.

As a (hopefully temporary) solution, we thus implement a Java-based "JGrid2SoapServer" that uses a shared library "SeeppServerJNI" for interaction with the C++-based simulation code. In its simplest form, the JSoap2GridServer packs the XML-encoded SOAP request received from the SEE++ client into a character string and forwards this string to the JGrid2SoapServer who uses the "SeeppServerJNI" library to translate the string back into the SOAP request expected by the simulation software. In this way, the SOAP request is "tunnelled" through the grid layer without any changes to the client or to the server software.

Once the simulation server can be equipped with a true grid-interface (which should be possible with the upcoming Globus 4 toolkit), we can merge the "JGrid2SoapServer" with the simulation software and find a direct representation of the SOAP request as a grid request.

4.2 Grid-based Hess Diagram Calculation

As described in Sections 1.1, a core operation of SEE++ is to calculate the Hess diagram for a given eye model.

Actually, typically two diagrams are calculated reverting the role of the fixing eye and of the following eye. When using the "JSeeppApplet" client, always both diagrams are calculated and visualized side by side; in the "SEE++" client the calculation of the second diagram has to be explicitly triggered by the user. Thus there is high probability that a calculation request for one diagram is followed by a request for the calculation of the other diagram, a fact which can be utilized for speculative parallelism.

As described in Section 2.3, each diagram consists of multiple points that can be calculated independently from each other, which can be utilized for data-parallel execution.

A grid-variant of Hess Diagram calculation can make use of both forms of parallelism as expressed in the following pseudo-code:

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

    T := lookupCache(E1, E2, E, I)
    if  T <> null then
      C := gridWait(T)
      clearCache(E1, E2, E, I)
      return C

    T1 := gridStart(hessDiagramGridCore, E1, E2, E, I)
    T2 := gridStart(hessDiagramGridCore, E2, E1, E, I)

    C := gridWait(T1)
    storeCache(T2, E1, E2, E, I)
```

```
        return C


hessDiagramGridCore(E1, E2, E, I):

        for each intended gaze position i∈I do
          T[i] := gridStart(position, E1 ,E2 ,E, i)

        for each intended gaze position i∈I do
          C[i] := gridWait(T[i])

        return C
```

The speculative tasks (*gazeDiagramGrid*) are coarse-grained and can be easily placed on different grid nodes; the data-parallel tasks (*position*) are fine grained. These tasks can be placed on multiple processors within a grid node; also a block-wise distribution of gaze positions to grid nodes may be possible. A parallelization of the optimization performed by a position task (see Section 2.3) is probably too fine-grained for practical use.

As a technical point, a call to *gazeDiagramGrid* must actually not block but spawn a thread and immediately return a session_id (see the client-server protocol sketched in Section 2.2). The progress of the thread (in terms of the number of points calculated so far) can be polled and, when all points have been calculated, the result can be queried. This behaviour has to be modelled by an appropriate grid interface.

Since the doctor performs many such calculations, a diagram should be available as quickly as possible, at best without noticeable delay. This "soft real time" requirement can be quantified by the psychological law of "perceptual fusion": this law states that events that occur not more than 0.1s apart from each other are perceived as simultaneously. On a current PC, a single Hess diagram calculation takes 5-15 seconds.

4.3 Grid-based Pathology Fitting

As described in Section 1.2, Pathology Fitting takes an initial eye model and gradually "improves" it (by modifying individual parameters or combinations of parameters) until the gaze diagram calculated from the eye model matches an ideal diagram. For practical purposes, the user denotes by a "parameter selection" S those parameters that may be modified.

The SEE-KID software essentially reduces by proper encoding the pathology fitting problem to a numerical optimization problem to which standard solutions can be applied. At a very abstract point of view, the algorithm can be described as follows:

```
pathologyFitting(E, S, M)

        E1 := E
        C1 := gazePattern(E1)
        if C1 matches M return E1
```

```
loop
        p := nextParameterVariation(S)
        E2 := apply(p, E1)
        C2 := gazePattern(E2)
        if C2 fits M better than C1
            E1 := E2
            C1 := C2
            if C1 matches M return E1
```

In other words, from an *n*-dimensional parameter space (which is actually a subspace of the originally *(m+n)*-dimensional space), the algorithm starts with an initial solution vector in which it gradually modifies individual parameters to determine better solutions according to some real-valued optimization function.

However, there are several problems associated with this approach. One is to find an effective way to enumerate the sequence of parameter changes since they cannot be always individually changed but are bound together by various constraints (medical, anatomical, and geometrical ones). Another one is that there may exist actually multiple solutions most of which do not fit the physiology of the patient; the current implementation of SEE++ only returns the first encountered solution. Furthermore, since the implementation follows a "greedy" approach (going from one situation to a better one), it runs into the danger of getting stuck in a "local optimum", because the "global optimum" is not reachable from the initial model.

Each fitting process takes a considerable amount of computation time. Research on above issues (e.g. experimental investigation of parameter variation strategies) was severely hampered by the fact that systematic experiments require a large amount of computing resources as can be only provided by a grid.

The core problem of pathology fitting (in the current modelling framework that discretizes the originally continuous problem) can be seen as a subproblem of "discrete optimization", namely that of "combinatorial search" i.e. to find one or more optimal or suboptimal solutions in a discrete problem space. There is a vast literature on algorithms for solving this problem and on the parallelization of these algorithms. In particular, parallel and distributed variants of the "branch and bound" strategy have been devised, including such ones optimized for grid computing [Filho et al, 2003, Aida et al, 2003]. While there are some differences between the classical model combinatorial search and pathology fitting, we hope to adapt the basic ideas from this domain to our problem.

We thus plan, on the basis of the existing literature, develop a grid-based version of the pathology fitting software of SEE++. In analogy to existing grid implementations of "branch and bound", the software may e.g. make use of a hierarchical master-worker scheme where multiple multi-processors respectively clusters are connected by the grid to perform the calculation in parallel (see Figure 5 taken from [Aida et al, 2003]).

It should be noted that from a fast grid-based implementation for pathology fitting experimental data can be gathered that provide valuable feedback to the design of the fitting method itself, which can in turn be embedded into the grid software. Consequently, grid

implementation and algorithm design cannot be separated from each other but go together hand in hand.

Another approach to the solution of the problem could be a pre-calculation of pairs of a large representative data base of (eye model, gaze pattern) pairs from the overall parameter space such that the pathology fitting process starts by determining the "closest" pattern whose corresponding eye model serves as a good starting point for the optimization process. Such a database could be initialized with a small set that only coarsely covers the parameter space but is refined on demand to cover various "interesting" regions in more detail. This is one motivation for the establishment of the grid database described in the following section.

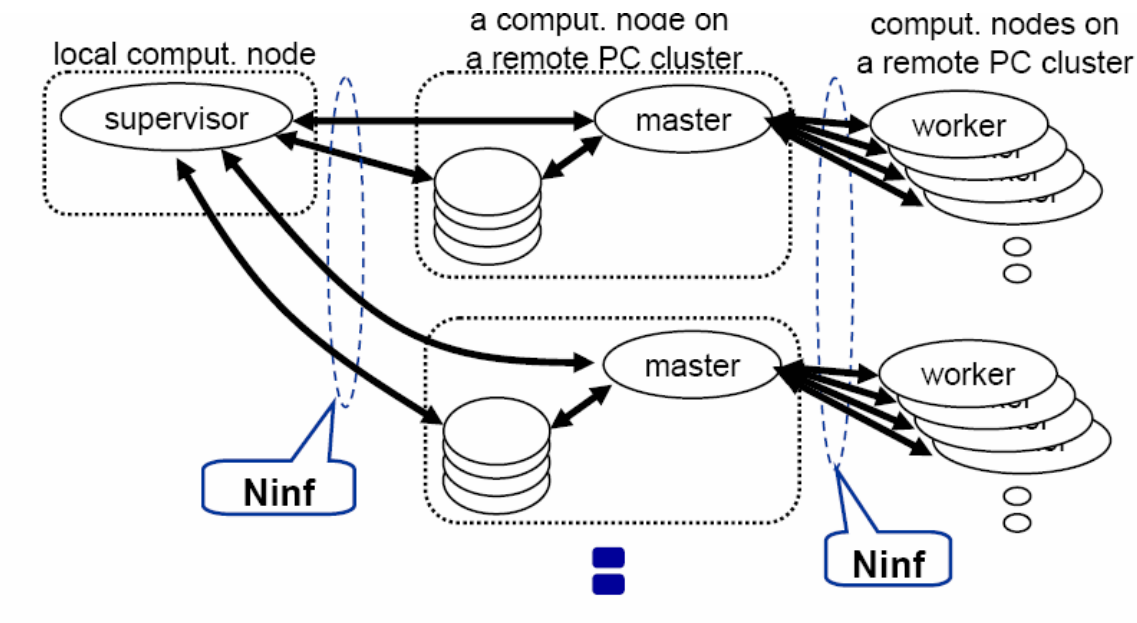

Figure 5: A Hierarchical Master/Worker Model [Aida et al, 2003]

4.4 A Medical Grid Database

As described in Sections 1.2, 1.3, and 3.3, the SEE-GRID software can profit in various ways from a grid-accessible database of relevant medical information, which is in line with the current trend on "evidence-based medicine". Apart from patient records and associated meta-data, this database will contain a large set of gaze patterns (measured and simulated ones) and corresponding eye models. This database can be then used in various ways:

1. Given a certain gaze pattern, we can search for data sets that "roughly match" the given pattern. In a database with *n* data sets, this essentially means to solve *n* instances of the core of the pathology fitting problem (the "pattern match").

2. Given a certain eye model, we can search for data sets that "roughly match" the given eye model. In a database with *n* data sets, this essentially means to compare *n* parameter vectors with a reference vector.

3. To perform statistical analysis on the database to detect certain patterns on the data set. This is essentially a data mining problem where data sets have to be compared with each other.

The database may retrieve its data sets by manual insertion of patient data (respectively by automatic transfer of data entered in local databases into the grid base) as well as by automatic insertion of the computed simulation data.

The database thus interfaces with the grid computations described in the previous sections in two ways:

- It is filled with data that are derived by grid computations (e.g. by storing the result of every pathology fitting computation),

- It is queried by using grid computations (e.g. by multiple applications of pathology fitting computations).

In the frame of the project, a corresponding model of the database with appropriate query mechanisms will be developed; this model will be implemented in such a way that it can be used by the grid computations described above and, for the purpose of querying, also make use of the grid computations. Thus the database itself becomes a grid service. Initially, we will use a centralized implementation of the database service, eventually we may also distribute the implementation over multiple grid nodes. Figure 6 illustrates the basic idea.

As for the grid implementation of the pathology fitting algorithm, we cannot separate the development of the grid database from the development of the database access methods that support the fitting process: by gathering more and more data and rerunning the calculations, more and more information becomes available that helps to construct better methods for matching data sets to given calculation problems.

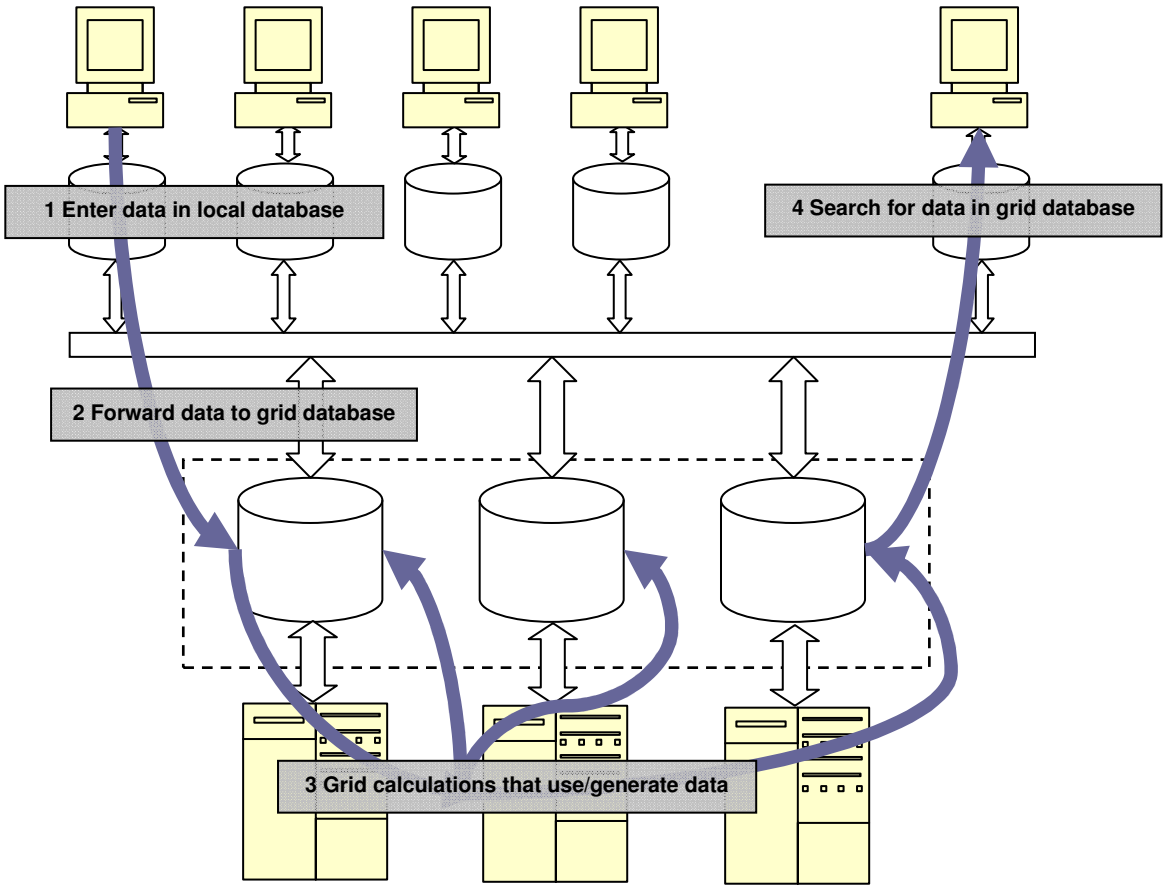

Figure 6: A Database for SEE-GRID

4.5 An Outlook on Grid-based Surgery Simulation

The grid mechanisms sketched in the preceding sections are also a basis for a future software component that solves the "surgery simulation" problem sketched in Section 1.3.

The core problem to be solved is identical to that of "pathology fitting" but with a much larger search space and with additional side-constraints (minimization of the parameter variations). This problem has in SEE++ not been touched yet, since it depends on a better understanding of how to effectively solve the "inverse problem" of computing eye models from gaze patterns, which is currently investigated in the (somewhat simpler) context of pathology fitting. By using SEE-GRID for running experiments (that would be too time-consuming in SEE++ alone) and analyzing the results, this understanding will mature such that eventually a component for surgery simulation can be developed. Since the computing resources for this component go beyond that of pathology fitting, SEE-GRID will also play an essential role here.

References

[Aida et al, 2003] Kento Aida, Wataru Natsume, Yoshiaki Futakata. *Distributed Computing with Hierarchical Master-worker Paradigm for Parallel Branch and Bound Algorithm*, 3rd International Symposium on Cluster Computing and the Grid, May 12 - 15, 2003, Tokyo, Japan. http://csdl.computer.org/comp/proceedings/ccgrid/2003/1919/00/ 19190156abs.htm

[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

[Filho et al, 2003] J. Viterbo Filho, L. M. A. Drummond, E. Uchoa e M. C. S. Castro. *Towards a Grid Enabled Branch-and-Bound Algorithm*. Report RT-05/03, Department of Computer Science -- Fluminense Federal University, 2003. http://www.ic.uff.br/PosGrad/RelatTec/reltec03.html

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

[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-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