Grid-aware Database Support for Medical Software

Master's Thesis

for obtaining the academic title

Master of Science

in

INTERNATIONALER UNIVERSITÄTSLEHRGANG
INFORMATICS: ENGINEERING & MANAGEMENT

composed at ISI-Hagenberg

Handed in by:
Imre Zoltán Matkó

Finished on:
July 2, 2008

Scientific Management:
A.Univ.-Prof. Dipl.-Ing. Dr. Wolfgang Schreiner

Grade:

Name of the Grader:
A.Univ.-Prof. Dipl.-Ing. Dr. Wolfgang Schreiner

Hagenberg, July, 2008

Acknowledgements

I deeply thank all the people who, during the several months of this master's study, have contributed in various ways to my work.

First I'd like to express my gratitude for Bruno Buchberger for founding the International School for Informatics and helping me to develop not just as a computer scientist but also as a person. I'd like to thank all those who were actively involved in setting up and coordinating the activities of ISI, specially Betina Curtis, making possible in this way for me to become one of the “pioneers” of this special master's program.

I would like to thank Judit Robu, my former diploma work adviser, who recommended the ISI master's program and was also actively involved in guiding my way to continue my studies in Hagenberg, “the best place for research” as she said.

I am very grateful to my adviser, Wolfgang Schreiner, for coordinating my work, for helping me to adapt in every difficult situation and for carefully guiding the composition of this thesis document.

I would like to thank the Austrian Grid Development Center (AGEZ) for offering me the chance to contribute to the SEE-GRID project and for supporting my studies and stay in Austria.

As the work was carried out in a close collaboration with the Research Unit for Medical Informatics from RISC Software GmbH, I've spent most of my work time in their offices. I would like to thank all their support and helpful attitude what I've experienced on a daily basis. I need to thank the members of the SEE-KID team, specially Thomas Kaltofen, for helping me to understand all the details I needed about SEE++, as well as the developers from the SEE-GRID team, specially Károly Bósa from the RISC Institute, for all the valuable discussions and advices.

I say “thank you” to all my colleagues from the International School for Informatics as well as to other students from Hagenberg, who were good buddies in our everyday life and also contributed to my work in different ways.

I gratefully thank and appreciate very much the efforts of my Parents, who always supported me in my choices making it possible for me to follow my own way. I need to remember this every day.

Imre

Abstract

SEE-GRID is a research project based on the SEE++ software system for the biomechanical simulation of the human eye. The project aims to develop a Grid-enabled version of SEE++ to support ophthalmologists in the treatment of common eye motility disorders.

Currently there exist Grid-based implementations of computation and calculation services required by SEE++ and a prototype of a metamodel based medical database. The current database implementation has some restrictions, performance problems and its Grid integration is unfinished. The goal of this thesis is to analyze the software and find performance bottlenecks and to extend the design with a Grid interface and provide a prototype implementation.

Based on the benchmarking and profiling of the SEE-GRID database components we realized that the current metamodel is too complex, causing to spend most of the runtime with database queries and data transformations. Some feasible solutions we proposed are the simplification of the metamodel or the usage of fast, XML based back-ends.

We have evaluated state of the art Grid data-resource management tools and found OGSA-DAI and AMGA two promising solutions for our project. We have extended the design of the SEE-GRID database with a Globus "*Web Service Resource Framework*" based interface and with the services of OGSA-DAI. To prove the applicability of the design we prototyped the Grid interface and evaluated in practice OGSA-DAI with respect to the needs of SEE-GRID.

Contents

List of Figures

1. Introduction

„Modice fidei quare dubitasti?"

The new term of Grid-computing was coined in the mid-late '90s by Ian Foster and Carl Kesselman [FosterKesselman1997]. They suggested that a new, efficient and secure solution would be needed to share computing resources, as easy for the users as the usage of the power-grid is. Since then many Grid projects have evolved and had a remarkable effect on science, offering a secure service to access resources which until now could be offered only by supercomputers. Nowadays in Europe nuclear physics experiments are the biggest Grid-resource consumers [LHC2008] immediately followed by applications from life sciences and biomedical research teams [EGEE2008].

A medical application which aims to exploit Grid technology is SEE++, a software system for the biomechanical simulation of the human eye and common eye surgery techniques. SEE++ is being developed in the frame of the SEE-KID project at the Research Unit for Medical Informatics of RISC Software GmbH [SEE-KID2008]. The software provides medical decision support for surgeons, focusing on the treatment of strabismus, a common eye-motility disorder. The main features of SEE++ are the following: the interactive 3D simulation of the human eye and its muscles; the simulation of common eye surgery operations; the simulation of the Hess-Lancaster test, a consacrated method in the diagnosis of strabismus; automatic pathology fitting to estimate the pathology of a patient based on measured or simulated medical data [Bosa2007].

SEE++ stores all the medical data in binary files, what was satisfactory in the beginning. However the current needs of the software could make use of a distributed database. This could collect data from many sources and offer an efficient way for their access and management. We also have to note that some calculations and simulation processes in SEE++ can be very time consuming. For efficiency and good performance these may even require the doctors to adapt some parameters manually [Mitterdorfer2005] thus are not fully automatic.

The SEE-GRID research project provides solutions for the above mentioned problems with the development of faster and fully automated algorithms and with a distributed database for SEE++. At the moment there exist Grid-based implementations for the parallel computation of simulation predictions and the distributed calculation of simulation parameters as well as a prototype implementation of a metamodel-based medical database [SEE-KID2008]. The database stores both real and simulated pathological cases and patient data. It is planned to gradually develop a distributed Grid database. This should support evidence based medicine, so

doctors could consult medical evidence data from a large database instead of books. Security is also an important aspect and the Grid offers the necessary technology to ensure the privacy of patients. The prototype database implementation now already uses a web service interface and offers basic store, load and search operations for SEE++ clients.

The current database implementation has some restrictions, performance problems and its Grid integration is not finished. The goal of this thesis is to investigate the main performance bottlenecks and formulate some recommendations for improvement as well as to extend the design of the Grid interface and provide a prototype implementation.

We did a benchmarking of the database operations performed by the SEE++ client. We also profiled the modules of the persistence component to obtain details about the instantiated objects, memory usage and the runtime spent in each module. Finally the gathered informations were completed with data extracted from the database management back-end (MySQL) about the network traffic, the number and type of queries and other details. The results made clear that the current metamodel is too complex, being the main cause of the poor performance of the persistence component. This overengineered metamodel causes to spend more than 54% of the average runtime of operations with database back-end operations and 10% of the runtime with data transformations. Some feasible solutions can be the simplification of the metamodel or the usage of fast, XML based back-ends.

To analyze the possibilities for the integration of the persistence component into the Grid, we have provided an overview on state of the art Grid data-resource management solutions. Two of these solutions which could be applied in the SEE-GRID project are OGSA-DAI for the Globus Toolkit and AMGA for the gLite Grid middleware. We have made a detailed evaluation of these latter two solutions and a short comparison of their features.

We extended the design of the SEE-GRID database with a web service interface based on the Globus "*Web Service Resource Framework*" and we also provided a prototype implementation to prove the designs applicability. We also included in the design the high-level services provided by OGSA-DAI. Our practical experience by prototyping the Grid interface and experimenting with OGSA-DAI showed that it is possible to further develop SEE-GRID based on these high-level Grid services. The further development in this way could complete the integration of the medical database into distributed Grid environments.

The structure of the thesis is as follows: in Chapter 2 we give an introduction to the field of Grid computing and an overview with special consideration to database management solutions. In Chapter 3 we offer an introduction to the medical background of the work, we describe the status of the projects involved, respectively the SEE-KID and SEE-GRID projects, and finally

we give a description about the limitations of the current database implementation. Chapter 4 presents our work on the performance evaluation of the SEE-GRID database. We present some important aspects about the architecture of the persistence component, the design of our benchmarking and profiling work and finally the results and conclusions are provided about the performance analysis. The evaluation of the two Grid data-resource management solutions, OGSA-DAI and AMGA, which are promising for the future development of SEE-GRID, is provided in Chapter 5 as well as a short comparison of them. In Chapter 6 we present an extended design of the persistence component with a new Globus Java web service interface and the OGSA-DAI Grid data-resource management system; in this chapter we also present our results about the implementation of a prototype of the mentioned interface and about our experiences with OGSA-DAI and give a detailed overview of the design. Chapter 7 includes the presentation of results and conclusions of this thesis work and an outline of the future development of the SEE-GRID database system.

2. State of The Art

Our work on Grid-aware database support deals mainly with the new but already quite elaborate technology of Grid computing and with its support for databases and meta-data handling. There are many Grid projects around the world but there are just a few software packages which are being used in production environments and just a couple of tools dealing with Grid-based database support. In this chapter we present the basics about today's Grid and the current state and capabilities of some data management software tools.

2.1. Grid Computing

A definition of the Grid is the following: “the Grid is a service for sharing computer power and data storage capacity over the Internet. The Grid goes well beyond simple communication between computers, and aims ultimately to turn the global network of computers into one vast computational resource” [GridCafe2008].

Its history dates back to the 80's, but it was mentioned under different names, such as meta computing, distributed resource management, resource co-allocation [FosterKesselman1999, Foster1999] or virtual computing [Grimshaw1994].

The origin of the term “*Grid*” is known to be the work of Ian Foster and Carl Kesselman's: "The Grid: Blueprint for a new computing infrastructure", where it is metaphorically compared to the power grid suggesting that it should be as easily accessible as the electronic power grid [FosterKesselman1997].

The most common approach to develop grid software is based on the middleware which lies between the OS and the applications. The middleware runs on each component of a distributed computer system (www.objectweb.org).

The Grid aims to offer a solution for solving problems with high computational and storage requirements. Up to now a possible solution was the usage of supercomputers, but their high purchase, operation and maintenance costs significantly reduce their applicability range. The Grid approach tries to overcome this with the integration of a large number of geographically distributed, heterogeneous resources into a virtual computer architecture. In many applications this can not replace the use of expensive supercomputers but it still offers a plausible solution for many other problems of science and business [Kesselman1998]. Actually Grid computing is seen rather to increase the need for supercomputers, because users from remote locations will require access to resources with low latencies and high communication bandwidths. Such top performance currently can be achieved only by supercomputers [Foster2001].

The Grid architecture, in a very general view, is built up on four layers. The lowest is the **network layer** which is responsible for the interconnection of the Grid resources. Next is the **resource layer** which can be represented by the computational resources, storage and database resources as well as by other types of resources. The **middleware layer** is composed of the software laying between the operating system of the actual Grid nodes and the high level Grid applications. This layer is essentially the software what enables the elements of the Grid to cooperate. On the top of the architecture is the **application layer**. This layer is composed of various applications for science, engineering, business or support systems for different applications. The actual users normally have to interact only with this layer. To this layer a so-called serviceware is bound which performs general management functions such as monitoring, logging and controlling the activities of the users [GridCafe2008].

The basic structural requirements of a Grid can be found mostly in the works of Foster, Kesselman and Tuecke [Foster2001, Foster2005, Banks2005, Tuecke2003, Andrieux2007]. They have contributed much to the standardization efforts in this domain of the Open Grid Forum, formerly known as the Global Grid Forum (www.ogf.org). Their multi-layered grid system, The Globus Toolkit, developed in the frame of the Globus project [Kesselman1998, FosterKesselman1999], became a de facto standard in the industry. Globus itself have also been influenced by other projects and have collaborated with other organizations and initiatives in areas as web services, virtual machines and peer-to-peer architectures [Foster2006].

In today's Grid systems the users are members of entities called Virtual Organizations (VO). A VO can correspond to a physical organization, workgroup or other group of people who can be in geographically remote places but are somehow related. One user can be the member of multiple VOs in the same time [Foster2001]. Generally VOs contribute to the Grid by sharing some resources and have access to other VOs' resources. This is a regulated process, each VO can define its own policy of access to its own resources and VOs can negotiate among them different levels of access to different resources.

The authentication of actual users on the Grid is handled by a certificate based public key security infrastructure [Foster2001]. This infrastructure provides a single sign on feature, so the users have to authenticate themselves just once and they get a uniform access and authorization to Grid resources corresponding to the rights of their VO.

There are hundreds of Grid projects around the globe, one of the largest projects is the European EGEE (Enabling Grids for E-sciencE – www.eu-egee.org). This project is formed by a cooperation of more then 240 institutions from 45 countries from all around the world. It provides an advanced Grid infrastructure for the scientists, based on a next generation

middleware called gLite (http://glite.web.cern.ch/glite/), which was developed in the frame of the EGEE project. The infrastructure of this Grid consists of more than 41.000 CPUs, provides around 5 Petabytes of storage and maintains 100.000 concurrent jobs. With the help of such resources the scientists can perform large scale calculations, simulations and modeling and are also able to share their results in a controlled and secure way.

2.2. The Austrian Grid

Austrian Grid [Agrid2008] is a project funded by the Federal Ministry for Education, Science and Culture (BMB:WK) after recommendation by the Austrian Council for Research and Technology Development. In its first phase (2004-2007) the project has strengthened the role of Grid computing in Austria and by time set up the necessary infrastructure for a prototype-testbed. The project has promoted grid computing among scientists and researchers and also contributed to international projects, such as EGEE.

The main hardware resources of the Austrian Grid currently are formed of several sites with Altix 350 Multiprocessor Systems as well as of various types of clusters. The software architecture is based on the Globus Toolkit 4, it offers the possibility to use several custom services as well as to submit MPI or OpenMP based jobs.

In its second phase (2007-2009) the project aims to continue the expansion of expertise of scientists, move to a sustainable service-oriented Grid architecture and to extend and develop various Grid applications. For a long-term sustainable and efficient development the Austrian Grid seeks to involve more and more the business and industrial sectors.

2.3. Grid Data Management

As much the Grid is about sharing and accessing computing power it is also about the handling of data. The results of different calculations or simulations may require to access some data during their operation and also they might produce large amounts of output. For an example the LHC Computing Grid [LHC2008] handles huge amounts of data and it has over two hundreds of nodes, geographically distributed all around the world. In full capacity operation it will have to distribute, process and store roughly 15 Petabytes of data annually, having thousands of concurrent jobs processing it [LHC2008]. If we think about other production Grids, such as the Austrian Grid (www.austriangrid.at), the int.eu.grid (www.interactive-grid.eu), etc., these also have to realize the efficient management of data. This requires scalability, performance and fault tolerance. It is clear that centralized services can't fulfill such requirements.

To fulfill the requirements for data handling on a Grid, technologies have to be provided for

the storage, movement and replication of data and the access and integration of data for applications and services [Antonioletti2007]. Nowadays we have quite elaborate Grid solutions for the handling of file-based data. A widely known solution is the GridFTP protocol [GFD.47] developed and implemented by the Globus Alliance.

A web service based solution is the Reliable File Transfer (RFT) Service [RFT2008], providing a "job scheduler"-like functionality. It enables the clients to transfer files or directories and to monitor the transfer status or to subscribe to receive notifications if the transfer status changes.

Higher level services of a Grid should realize data replication and the management of the different copies or replicas. We can find such type of data management in the Globus Toolkit, implemented by the Data Replication Service (DRS) [DRS2008]. This high-level data management service ensures the management of files distributed and replicated over storage sites by locating them via a Replica Location Service and executing transfers by RFT.

2.4. Grid Database Resources

Many applications, now also on the Grid, require the integration of database technologies and many projects, such as the AstroGrid or DGEMap, utilize large databases of astronomical and biological data [Antonioletti2007]. The Grid-enabled version of SEE++ (see Section 3.3) already uses databases for managing user information and medical data. However this aspect is not yet fully integrated with Grid middleware [Mitterdorfer2005].

There are a couple of solutions which deal with distributed data and database handling. Some of them were developed specifically for Grid middleware while others can be deployed in a standalone manner but have support for the integration in Grid environments. Here we present a few products and give a brief overview on their current state and capabilities.

- **The OGSA-DAI Project**

The OGSA-DAI Project [OGSA-DAI2008] provides middleware software to support heterogeneous Grid database resources. The project has started in 2002 and by now it has undergone three main phases. It is developing two main software products, the OGSA-DAI and the OGSA-DQP.

OGSA-DAI is a product which is able to expose for querying and updating a large variety of data sources through a web service based presentation layer. Its web services are WSRF compliant [OGSA-DAI2008, WSRF2008] so it can be deployed in different environments supporting this framework definition. The requests to the web services are uniform regardless of the data sources which can be relational, XML or even file-based. The clients can define and use

the different data sources in a grouped or federated manner so there is no need to create separate new connections to access multiple data sources from the same client [OGSA-DAI2008].

The OGSA-DQP framework is a service based Distributed Query Processing system. It is an extension of OGSA-DAI for processing distributed queries over potentially remote collections of relational data services. The framework basically consists of a coordinator and an evaluator service. The coordinator is the Grid Distributed Query Service which keeps track of and manages the metadata and computational resource information. The Query Evaluation Service is used by the coordinator to execute different query plans. The execution is partitioned by the coordinator, so actually a set of evaluators are performing the actions, which are organized in a tree structure [OGSA-DQP2008].

There is already a large number of projects using OGSA-DAI and OGSA-DQP, many of them being from the field of biomedical sciences, such as the Biogrid, BRIDGES, ViroLab, etc. A wider list of projects using this software is maintained on the official website [OGSA-DAI2008].

We have to note that in the last times the OGSA-DAI system has suffered major changes, the latest release, version 3.0, is a complete redesign and reimplementation of the product. This was necessary in order to increase performance and reliability and to put the basis of a sustainable development for the future, taking into account new standards and user requirements [Antonioletti2007]. The latest version of OGSA-DQP, version 3.1, is still in a prototype phase and can be used only with the older OGSA-DAI 2.2. A new release is planned for the future which will support the new architecture of OGSA-DAI 3.0 [OGSA-DAI2008].

- **WebSphere Information Integrator**

An enterprise solution from IBM for the management of distributed data sources is the WebSphere Information Integrator (WSII) [WSII2008]. It is a fully J2EE compliant, web services compatible implementation. It allows to access in a uniform way the data stored in different repositories which can be integrated via a library of pre-built connectors. The system supports NTFS filesystem based resources, FileNet services, Lotus and DB2 repositories but it has only a read-only access support for DB2, Oracle or other relational database systems accessed through the WSII federated data server. By data federation the WSII can offer combined content views, federated search capabilities and automatic notifications for events. It supports a large variety of clients thanks to its SOAP based web services interface architecture, it has a built in internationalization support, data distribution management and load balancing [WSII2008].

Drawbacks of the software from the point of view of Grid computing are that it is a commercial product that was not designed specifically for Grid environments. However, thanks to its web services support and OGSA-DAI wrapper, it can be integrated into grids [Sinnott2005].

- **The AMGA Metadata Catalog**

AMGA is a software package developed by the ARDA Metadata Catalogue Project [AMGA2008]. The projects aims to study the needs on metadata catalogues in Grid environments and to provide an interface for metadata access on the Grid. The project was started by the ARDA group and has evolved hand in hand with the gLite Data Management team. It has become the official EGEE metadata interface, being included in the latest gLite releases [Santos2005].

Metadata Catalogs are such services on the Grid which allow clients to search, locate, query and eventually update data among multiple data storage resources [Santos2006]. This offers an abstraction layer which separates the actual data resources from the way of interacting with those providing in this way support for various data resources.

The implementation by default provides a streaming and an alternative, SOAP based, front-end for client-server communication. The AMGA server has support for interactive (command line) clients for the streaming front-end and provides client APIs for C++, Java, Python, Perl, PHP and even for Web based streaming clients. There is also an option to group the different data sources in data federations [AMGA2008].

One of its important features is the database independent replication built into AMGA, supporting partial replication and federation of catalogs also.

- **Storage Resource Broker**

The Storage Resource Broker (SRB) [SRB2008] is a software developed at San Diego Supercomputer Center. It supports distributed shared data collections on various storage systems. The primary motivation of the project was to develop a software infrastructure for distributed file systems, distributed logical file systems and digital libraries. It can be also applied as a Data Grid Management System and it offers support for the security infrastructure of Globus. Its metadata catalog service (MCAT) supports federations of data sources and replication [Rajasekar2004]. It is a quite elaborate and robust product but its services are not OGSA compliant, its structure does not allow a modular reuse of parts and currently it isn't open source [SRB2008].

The project is planned to be stopped and replaced by the new iRods system, developed by the

same team, however it will still have support in the future [SRB2008].

- **iRods**

The SDSC Storage Resource Broker team has recently started the development of a new open source data grid software system called iRods (Rule-Oriented Data management System) [iRods2008]. On a long term the project aims to replace SRB with iRods and to support the transition between the two systems.

The software is based upon the experiences with SRB providing similar functionality extended with a "Rule Engine" in the iRods core which has to interpret the requests based on rules and actual conditions. This feature enables the use of a simple mechanism for application specific processing by custom defined rules and workflows [Hedges2007].

For the future, it is planned to implement a complex preservation mechanism of digital content, however this is more related to binary, file-based data. Preservation implies the storage of digital data in some standard format corresponding to its type. This implies that the abstraction layer of the data resource manager shall provide for the clients data converted into the specific formats requested by the clients [Hedges2007].

The project is active and iRods has reached version 1.0 to be released in 2008. It is designed to be used with Unix and Linux systems but its full integration to Grid middleware is still missing.

- **Mobius**

The Mobius project [Mobius2008] of the Biomedical Informatics Department of Ohio State University aims to develop tools and middleware components to share and manage data and metadata in Grid systems or other distributed environments. Its approach is based on XML, every data source has to be described by an XML schema. These schemes are then managed with the DNS-like Global Model Exchange service and are exposed using XPath. Its module for data instance management, called Mako, implements the actual query, update and delete operations on different data sources which can be relational databases, XML databases, file systems or of any other type having appropriate connectivity interfaces. It supports the federated management of different data sources. Another feature is the Data Translation Service (DTS) for translation between data models having similar content but a different structure [Langella2004].

The project web page [Mobius2008] describes various plans for future development but there was no new release of the product since version 1.0, appeared in December 2004, and no major change since 2005.

3. Problem Statement

In conventional medicine all the processes of a treatment are carried out directly on the patient. This is especially valid in treatments where surgeries have to be performed. In such cases the doctors have to rely on the advices of the most experimented surgeons and to follow strict guidelines. In ophthalmology, for an example, in the case of complicated pathologies even these specialists have to rely on documented empirical values and have to perform multiple surgeries for satisfying results [Buchberger2008]. For such cases new computerized medical decision support systems could offer major help, which can offer accurate bio-mechanical modeling and simulate surgery results. However these systems require adequate technical background and the development of new technologies. A project to develop such a system for eye-motility disorders is the SEE-KID (see section 3.3).

Technological development always had a beneficial influence on medicine and medical sciences. It has opened new perspectives for doctors from research to everyday health care. Nowadays almost every complex medical examination is computer based or computer assisted. The tendency in developed countries is to move towards an IT-supported health care system with the electronic administration of patient data, examination results and medical-decision support [LenzReichert2007].

Nowadays supercomputers make possible to perform several simulations for the research on fundamental problems of biomedicine, biomechanics and molecular biology. This involves problems such as the analysis of DNA sequences, genome annotation, evolutionary population simulations, protein structure prediction, biomedical image analysis etc. [Nair2007].

In medical diagnosis and decision-support todays high performance computers can provide a reasonable support in the analysis of examinations data, image processing and 3D visualization. This requires not only high performance but also reliability, authentic representation and secure data handling taking into account the patients privacy [Buchberger2004].

We can see that the keys of the problem are high performance and reliability in the calculations and efficient and secure handling of the medical data, respecting the corresponding laws.

3.1. The Treatment of Strabismus

A field of medicine which could use effectively medical-decision support provided by modern computer based technologies is ophthalmology. Here we briefly present a common eye-motility disease called strabismus and some aspects of its surgical treatment.

Strabismus, also known as squinting, is an eye alignment disorder when one or both eyes of the patient are deviated out of alignment. Nowadays about 6% of the population suffer of strabismus. These persons can suffer of double vision or in other cases of severe weakening of the eye which fails to accurately fixate on objects because of its misalignment. The limited binocular vision caused by this type of disease also affects the three-dimensional spatial vision [Buchberger2004]. This type of disorder very often can be sufficiently corrected with eye muscle surgeries. This surgery deals with six extraocular muscles which form a closed system. This makes the surgery complicated because changes to a muscle affect all the others so it is hard to predict the exact outcome [Kaltofen2002]. In this field of ophthalmology doctors still have very little computer-based support. They have to follow guidelines based on assumptions and have to lean on their experience in the surgical treatment of strabismus [Buchberger2004].

In the traditional surgical treatment of strabismus experience shows that in many cases more than one intervention is required because the estimation of the outcome is not precise enough. With detailed graphical visualization, simulation of the disease and interactive "virtual surgery" plan and check, an ophthalmologist doctor can correct the process of a surgical procedure to achieve the best result [Buchberger2004].

3.2. Existing Solutions

The first simple geometrical eye models have been created in the 19th century. These formed the basis of the so-called "string model" of Krewson, from 1950, and the more sophisticated "tape model" of Robinson [Mitterdorfer2005]. These models can describe the geometry of the human eye but lack the simulation of muscle forces and statics and thus are considered "historic" modeling approaches.

Based on the research work of Miller and Robinson, from the 80's [Haslwanter2005], a more sophisticated modeling became possible, which could take into account the effects of changes to the oblique eye muscles. Later, in 1995, as a result of the research made by Miller and Demer, the "active pulley model" was presented [Miller1999a], being still one of the most accurate and sophisticated approach.

The most notable computer based solution of the 90's was the Orbit™ Gaze Mechanics Simulation software system (www.eidactics.com). It was designed for educational and research needs being able to simulate strabismus syndromes by a sophisticated bio-mechanical model [Miller1999].

According to Eidactics (www.eidactics.com) the latest version of the software, Orbit™ 1.8, appeared in 1999 and since then there was no update. Orbit™ fully supports only the Macintosh

OS9, its usage is limited due to the Classic mode of OS-X under newer versions, the “Parameter Fitting” function does not work in this case, and the program can not be run at all on Intel Macs.

There also exist some virtual eye surgery systems which are able to model the mechanics of the human eye, such as the Eye Surgery Simulator of the William H. Havener Eye Institute of Ohio State University (www.eye.osu.edu). These are serving just for training purposes and do not include pathology fitting or medical decision-support features.

Regarding to the diagnostics of squinting, the Hess-Lancaster test is a well known and accepted method [Buchberger2004]. This test results in two diagrams which represent the state of the two eyes of the patient. The diagrams contain the so-called “intended gaze pattern”, representing what the patient should see, and the “measured gaze pattern”, showing what the patient actually sees.

To appropriately handle this type of medical data, both the client's personal data and the examination results, we need special data models because we have to handle very complex structures and store them in an organized way. There are several general and special metamodels. These kind of models were developed in the frame of the Healthcare Modeling Programme of the British National Health Service at the beginning of the 90's. As a result of this effort the Cosmos Clinical Process Model of Martin Fowler appeared [Fowler2004]. An adaptation of this model for ophthalmology was applied in the SEE-KID project for the web service and the Grid-based data base support [Mitterdorfer2005].

3.3. The SEE-KID Project

The SEE-KID Project (www.see-kid.at) was originally pursued at the Department for Medical-Informatics (UAR/MI) of the company Upper Austrian Research GmbH (UAR – www.uar.at), which is a non-profit institution, focusing on fundamental and applied research and the knowledge transfer between science and business. Recently the department has moved to RISC Software GmbH. SEE-KID currently is in an advanced phase, focusing on the development and research for new features and upgrades of the SEE++ software system. For the simulation of eye surgery, SEE++ is able to provide similar functionality as the Orbit™. However compared to that it has many extensions and new features and it exploits newer techniques of software and hardware environments. It also takes into account newer medical research results about eye model types in implementing its own SEE-KID model and Active Pulley model [Kaltofen2002] of the human oculomotor system.

The system is designed to run on Microsoft Windows operating systems. In 2004 the development of a client/server approach was started which moves the calculations to a

Calculation Server. This implies the possibility to set up a separate SEE++ calculation server and use it with different clients. This by now works well on different Windows, Linux and Mac OS X platforms. The connection between the clients and the server is done via a web service interface which allows the implementation of a large variety of clients which can use a LAN connection or even an Internet network access. There are many different client implementations, such as the original C++ version for Windows, the Java based interface, or the Matlab interface. The server can accept requests via its webservice interface while there also exists a RMI to SOAP bridge for Java-RMI applications.

A new exciting feature, which is currently under development, is the VOG or Video-Oculography mode. This technology enables the recording of eye movements which can support a clinically applicable, automated binocular strabismus-test.

3.4. The SEE-GRID Project

The SEE-GRID project (www.see-kid.at) aims to provide near real-time performance for the simulation and pathology-fitting calculations by developing a Grid-enabled variant of the SEE++ system. The development takes places in close collaboration between the Austrian Grid Development Center (AGEZ), the RISC Institute and RISC Software. The use of Grid environments can be the solution for the high computational needs of SEE++, but currently the implementation and adaptation of the software is in an alpha stage. SEE++ currently is using binary files to store patient data. This is sufficient and fast in a standalone or a simple client-server deployment. However with new requirements it appears the need for the application to support various database systems to provide the portability, efficient search, version control and

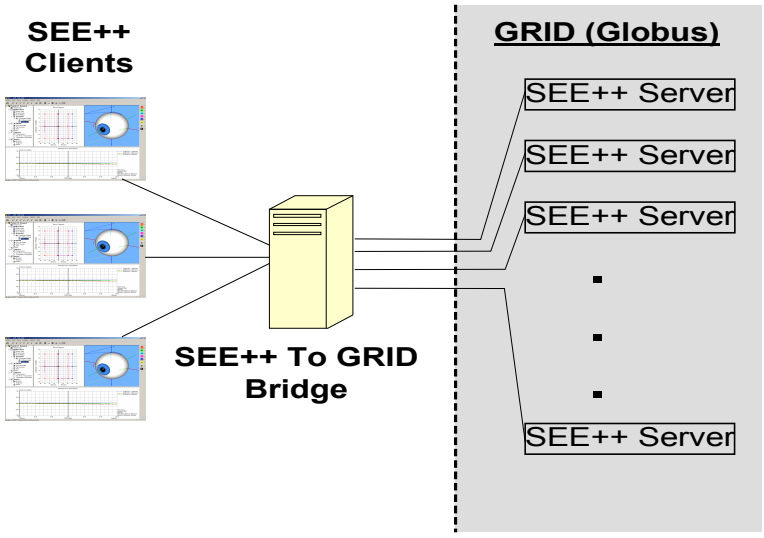

Figure 1: Sketch of the SEE-GRID architecture for calculation requests (from [Bosa2005])

sharing of patient data and the management of user data. For these reasons a prototype of a Grid-aware database for SEE++ has been developed.

3.5. The Persistence Component

The database system of SEE++ is currently a metamodel-based [Fowler2004, NHS2005] implementation which uses web service technology for communication [Mitterdorfer2005]. This implies the development of a so-called "persistence component" which realizes the Transport Model – Metamodel Mapping. This is a quite complex task and in the same time a key factor for the integration of the database system in Grid environments. Currently the SEE-GRID project is working with the Globus [FosterKesselman1999] and gLite [gLite2006] Grid middleware solutions.

3.6. Grid-Aware Persistence

The Grid-aware persistence component realizes the integration of the database system into Grid environments. Because its design is essentially web service based, its integration into grid environments such as Globus or gLite is not very complicated. It requires the extension of the authentication and authorization provider in such a way to support the proxy-certificate based Grid security infrastructure. This is realized by the Security Base module of the Persistence component. In this way the current implementation deals with two authentication procedures because the users of the client and the grid user handling form two separate security layers.

The latest version of grid middlewares also contain higher level database support. The Globus toolkit supports the OGSA-DAI implementation (www.ogsdai.org.uk) which is a purely Java based WSRF compliant web service being able to use as database resources a large variety of relational, non-relational databases or even filesystem based persistence [OGSA-DAI-DevGuide]. For the gLite middleware the AMGA Metadata service was developed which has a promising fast streaming communication protocol, supports mysql, Oracle, Postgress and SQLite backends and can realize basic master-slave replication [Koblitz2008].

3.7. Limitations of The Current Implementation

The persistence component implementation at the moment is quite limited so it requires serious optimization and extension work. This limited functionality consists in: a poor performance while we would require real-time or near-real-time operations; it supports only one data source thus the data can be stored only on a single node; it is over-engineered, has a too general structure for the metamodel.

The current implementation uses JDBC drivers for the database connection, thus it uses a

direct, point-to-point connection, not taking into account the Grid infrastructure and limiting its usage to a single database server node.

The development team also encountered some limitations from the side of the Globus Toolkit middleware while extending the software to use the "Web Service Resource Framework" architecture which makes impossible the full integration with this middleware [Bosa2007, Woess2006]. Now there exists another approach for the development using the gLite Grid middleware [gLite2008] and there is an intention to further extend the software on the basis of higher services of gLite [Bosa2007]. In the next phase for the development regarding the database system we need to evaluate the capabilities and database support of the Globus and gLite middleware as well as to improve and extend the current implementation.

4. Performance Analysis of The Current Database System

As we have presented in the previous chapter, the performance of the current SEE-GRID database implementation is not adequate for "real-world" deployment thus it requires optimization. In order to complete this, we need an analysis of the persistence component and the identification of possible performance bottlenecks. This requires a benchmarking of the relevant features of the software and then a detailed profiling of the most performance-critical parts. Based on these results, we will evaluate the current status and plan the optimization work.

4.1. Persistence Component Architecture

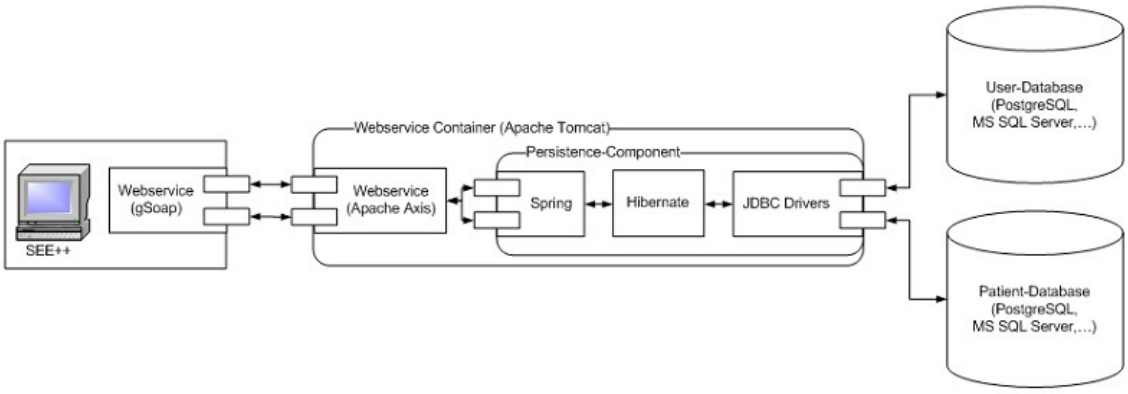

Figure 2: SEE++ Database Access Layer (from [Bosa2005])

The SEE-GRID project aims to gradually extend SEE++ to develop a grid-based medical decision support system. For the persistence component, in a first stage, a database system was designed and prototyped as a web service application (see Figure 2.) [Mitterdorfer2005, Bosa2005].

The SEE++ clients require from the persistence component to create, save, update, load and delete medical and patient data. These data consist of patients' information, data about measured and simulated gaze patterns, results of medical experiments, etc. The communication between the clients and persistence component is realized via SOAP messages. The web service interface is provided by Apache Axis with which it is loosely coupled; it acts as a facade to the underlying components. The server side stubs, a remote Proxy class and a corresponding interface, as a facade, are generated during build time with the help of the **FacadeGenerator** module (see Figure 3); more information is provided in chapter 5.4.1. of [Mitterdorfer2005]. In the future this web service interface is planned to be substituted by a grid-enabled component [Bosa2005a].

Security is very important since the application is operating on confidential medical records

and patient information. The **SecurityBase** module implements different techniques to ensure security; it is connected only to the web service module as presented in Figure 3. It intercepts every web service method call and performs authorization based on user name and a password encrypted with a SHA-512 salted hash. For grid deployment it is also possible to extend the security module with certificate-based authorization support. The design ensures that users can not use any operation unless access is granted based on their credentials. The users can be members of one or more different groups. Access rights can be granted only to these groups but not to individual users. The network communication also has to be secured, which can be realized by Stunnel [stunnel] via encrypted HTTPS or SSL. The network transfer encryption is eliminated during development to enable easier network traffic debugging.

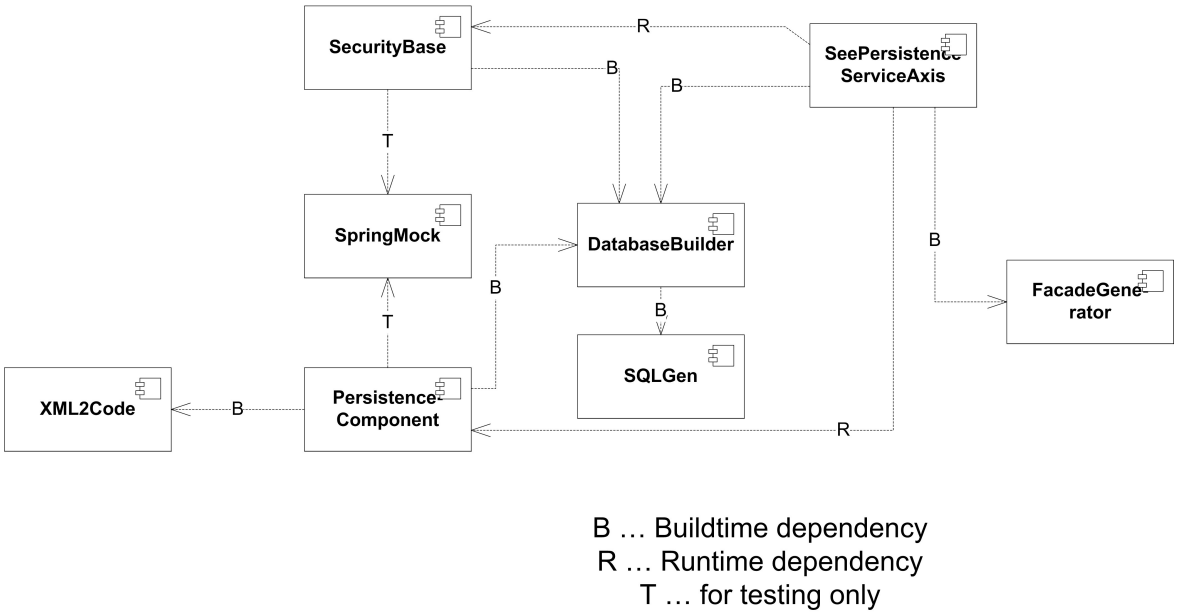

Figure 3: SEE-GRID Modules Dependency Chart (from D. Mitterdorfer)

The core component is represented as the **PersistenceComponent** in Figures 2 and 3. It is based on the Spring framework [Spring] component manager. Spring allows configuration support via XML files, facilitates unit testing and fosters the implementation against interfaces. The SEE++ application uses object-oriented data structures which are mapped to a transport object model. This model was developed for the client-server architecture SEE++ and was re-used for the persistence component to keep the clients simpler. The persistence component has a generic metamodel based design supporting general medical databases. This metamodel is instantiated into a concrete model when it is applied for a specific data model. When the data model of an application changes or we want to apply the persistence component for a different application, it is not needed to change the persistence component or the metamodel at all. In such

cases, only the instantiation of the metamodel has to be changed.

In SEE++ the actual transport object model does not match the metamodel, because the latter is more sophisticated, thus marshaling and unmarshaling require more time for such complex object graphs. The persistence component has to communicate with all the clients via the transport object model but internally it operates only with the metamodel. Because of this, an additional transport model – metamodel mapping layer was introduced. This layer is based on transformation rules for both ways between the two different representations. There are also operations implemented to perform the comparison of two object-graphs, regardless of their actual representation; furthermore two such graphs can also be made equal.

The medical data finally can be stored in any relational database supported by the JDBC drivers. This is done with the help of the Spring framework which has predefined Object/Relational (O/R) mapping functionality implemented by an open source tool called Hibernate [hibernate]. This tool ensures a transparent conversion from the object-oriented model to the relational data structures and vice-versa. Transparency means that the tool can be applied to arbitrary objects, because those don't need to implement any interfaces or to extend specific classes. The database access layer is automatically generated by Hibernate from the existing database tables. This feature enables the independence of the underlying database system. More details are presented in Chapters 4 and 5 of [Mitterdorfer2005].

4.2. Benchmark and Profiling Design

For the evaluation of the current state of the persistence component we need to analyze the performance of the product. We need to identify the possible performance bottlenecks and analyze their causes. Such a task can be carried out by the help of different runtime analysis operations. In the following we present our approach.

Benchmarking is designed in such a way that it will measure the performance of the different operations provided by the SEE++ clients. By performance measurement, we mean the overall runtime and maybe memory usage monitoring. The monitoring should not affect the software's performance, therefore the tools or techniques selected should not introduce overhead to the operations, at most they may introduce monitoring actions with a minimal overhead. It is also an important aspect to choose an appropriate test data-set as it influences the processing time of the operations.

By profiling we aim to identify and analyze in detail those modules of the persistence component which are causing the main performance bottlenecks. This involves an evaluation of execution times on package, class and method level and memory usage information of modules

and individual objects. For this task the benchmarking can give some additional hints on which functionality of the software we should concentrate and what kind of sample data should be used.

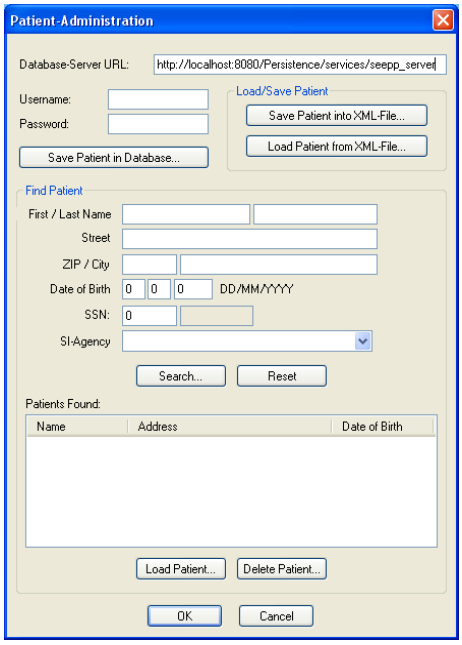

Figure 4: SEE++ Client Dialog

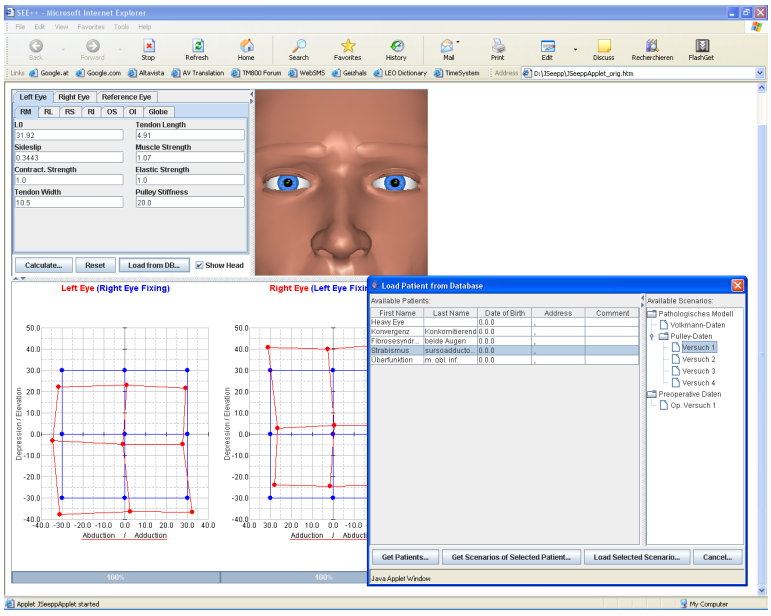

Figure 5: SEE++ Java Applet Client

The benchmarking and profiling scenarios are designed to take into account our experience with the usage of the SEE++ clients. In Figure 4, we can see that the SEE++ rich client can perform different operations on the database with the medical data. These operations can be categorized in the following groups:

- **Create** – New patient and medical data records are created in the database with the "Save Patient in Database" function of the client if the corresponding patient has no previous records; if new medical data is introduced for an already saved patient, new medical scenarios are created (examination, simulation results etc.).
- **Update** – An update operation is performed in the database if the actual patient and its medical data were already saved and just some information, values were changed and those are updated with the "Save Patient in Database" function.
- **Retrieve** – The "Search" functionality is able to search for patients, based on various constraints (see Figure 4). This involves the retrieval of the personal data for one or more patients if the search is successful but no medical data is retrieved. Based on the search results, the "Load Patient" function retrieves all the medical data of a selected patient.
- **Delete** – The "Delete Patient" function deletes from the database all the personal and medical information of a selected patient, however this is just a "logical" delete.

For the usage of the persistence component the user name and password of a SEE-GRID user

has to be introduced. There is no login/logout functionality since the communication is stateless and each call is secured separately.

In Figure 5 we can observe that the SEE++ thin client, a Java Applet based client, has a limited functionality compared to the rich client. The thin client is only capable to perform read-only operations and it has no advanced search functionality because it was intended to be used only for demonstration purposes on a sample database. Because of its demo purpose it does not require user information, it automatically connects to the persistence component with a limited guest user authentication.

The test cases for benchmarking are based on the above enumerated groups of functionality. Each of the test scenarios will be performed on a dataset of different types of sample data: where patients have a minimal amount, average and large amount of medical data stored in the database, respectively written to the database for save/update operations.

To set up a test environment we have several choices. Benchmarking directly with the usage of the SEE++ client by human intervention is not a good option because it can be very error prone and the individual scenarios can not be reproduced, repeated. A possible solution is to implement a modified version of the client software which can perform automatically certain predefined operations for benchmarking and profiling. This requires an additional extension of the existing code and eventually to hard-code some sample data which can not be loaded simply from files. An alternative, yet plausible solution, is to automate the behavior of the client GUIs by the simulation of user intervention. For such purposes there are available different tools based on simple script languages. We have chosen the freeware AutoIT [autoit] software tool. AutoIT has an advanced scripting language for Windows GUI automation by simulated keystrokes, mouse movement and window/control manipulation. Scripts can be compiled into standalone executables so each scenario can be repeated whenever it is necessary and they are even portable between different computers. It is also possible to create complex GUIs and interactive scripts so individual runs can even be parameterized.

For profiling the most simple, but powerful solution is the Java™ Virtual Machine Tool Interface (JVM TI) [jvmti]. JVM TI provides low level support for time and memory usage profiling and monitoring however the application of such tools in such a complex environment as the SEE-GRID would require serious efforts. Thus we've considered to use an automated profiler tool which can perform Java application profiling by instrumenting the byte-code during runtime, so the source code does not need to be modified at all. An appropriate profiler for such purposes is the Test & Performance Tools Platform (TPTP) open platform [tptp]. It is an advanced framework for application monitoring, testing, tracing, profiling and log analysis. We

have chosen this tool because it has support for web application profiling so we can do the performance evaluation of web-services deployed in a container. TPTP profiling offers on package, class and method levels analysis for execution time, memory usage, method code coverage and probe insertion.

4.3. Benchmark Technicalities

All the benchmarking and profiling was realized on a 64 bit AMD Dual Opteron 1.6GHz system with 1GB RAM, running Windows XP Professional, x64 edition, Version 2003, SP1 operating system. For the deployment and testing of the persistence component, the following tools were used:

- Java Runtime Environment 1.5.0.15
- Eclipse Version: 3.3.2, Build M20080221-1800
- Test & Performance Tools Platform (TPTP) 4.4.1
- AutoIt Version 3.2.10.0
- MySQL 5.0.45, database management system, community version
- Apache Tomcat 5.5.26 web server
- SEE++ 7.3

4.4. Benchmark Results

By the automation of the SEE++ client we have performed the time measurement benchmark of save, load, update and delete operations. The sample data set is composed of patient data and corresponding medical scenario data with 2, 5 and 13 scenarios. These samples reflect scenarios with minimal, normal and large data sets. One benchmark scenario, for such a sample data, realizes the following operations on the persistence component: two search operations, one save, one update (save the data after it is already stored), load and delete operations. A complete iteration performs these steps once for each of the three sample data sets. The time measurements are based on the average of 10 such iterations. The AutoIT scripts implement both the automated test cases for the SEE++ client and the time measurement of the individual operations. The time measurement on the client side is also meaningful for the persistence component, because all the software components, server and client both, were running on the same system for the benchmark. The SEE++ client's, the persistence component's and the database manager's (MySQL) interconnection is realized via the local host's loopback network adapter, thus the network overhead is constant and not significant.

	2 scenarios	5 scenarios	13 scenarios
Traffic Received (KB)	3025	12288	35840
Traffic Sent (KB)	9633	36864	107520
Total Traffic (KB)	12658	49152	143360
Connections (number of)	61	61	61
XML test data size (KB)	364	915	2396

Table 1: Sizes of sample data and the network traffic, between the persistence component and the database system, for the three test scenarios

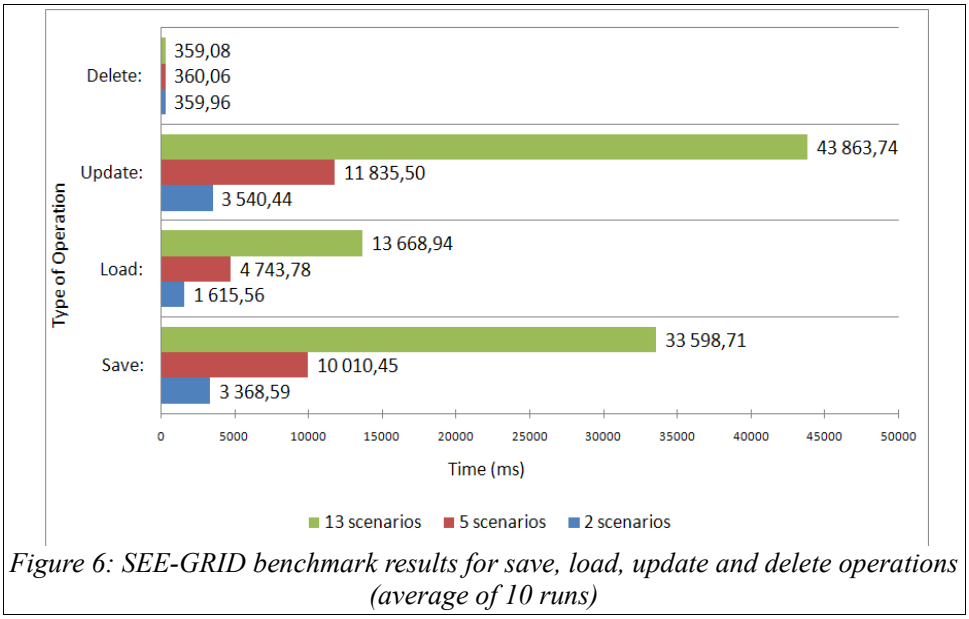

Figure 6: SEE-GRID benchmark results for save, load, update and delete operations (average of 10 runs)

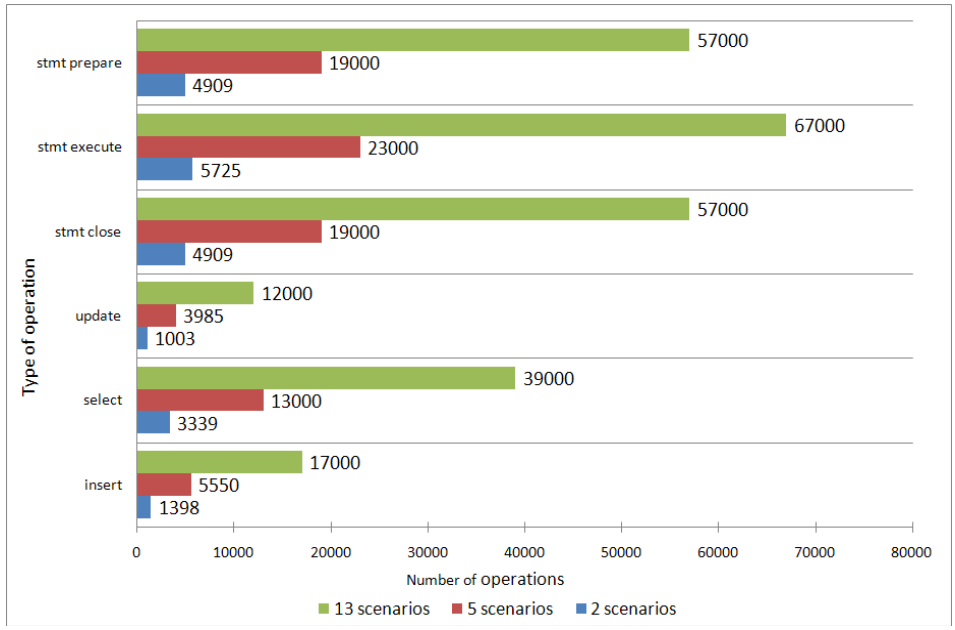

Figure 7: The number of the most important operations performed on the MySQL database for the SEE-GRID benchmark

The SEE++ client loads all the sample data from XML files. The sizes of the data sets can be seen and compared in Table 1, as well as the amount of network traffic between the persistence component and the database manager. These data are transferred to the persistence component in XML format, in SOAP messages. In Table 1 we can observe that the data sizes show an exact linear growth: 5 and 13 scenarios meaning an increase of 2.5 and respectively 6.5 times, compared to the first case, having 2 scenarios. These growth factors exactly correspond to the growth of the data size for the XML data as the data of individual scenarios has a fixed size. The persistence component's network traffic with the database manager increases with a factor of 3.88, respectively with 11.33 for the larger data sets (see Table 1). This indicates that we need to take special attention for the evaluation of the modules in charge for processing the medical data. The number of total connections (see Table 1) is the same for all cases, it is not influenced by the properties of the data we operate on, but only by the executed operations.

The results, presented in Figure 6, clearly show a non-linear increase in the runtime as the test data's size is raised. The non-linear increase of network traffic presented before is also reflected and seems to have some relation with the runtime of the different operations. For these test scenarios we also gathered informations about the operations performed on the database manager. We accessed and saved information of MySQL 5 offered by its phpMyAdmin platform. A detailed list of profiling data with relevant MySQL operations can be found in Appendix C. These are information about the network traffic, number of retrieve, write, update and delete operations on the data tables, information about the database manager's cache and much more. The results are presented on Figure 6, 7 and Table 1. These MySQL runtime information reflect the data gathered for one iteration with each sample data.

The most important operations, with the highest execution numbers, for the three sample data sets are presented in Figure 7. Beside the known update, insert and delete operations we can find the so called stmt operations. The stmt operations are related to statement handles. A statement handle is a pointer to a data structure that contains information about a single SQL statement [MySQL2008]. Statement handles can be associated with queries. Typical statements can be also update, delete or insert commands. The values on Figure 7, just like the network traffic, also show a growth of factors around 4 and 12. The total number of queries for the three scenarios are: 15979, 61432, 182640. These also reflect the above mentioned growth ratios. From these data we can see that the database system itself does not raise the complexity but the persistence component's modules do.

4.5. A Benchmark Scenario in Details

With the TPTP profiler tool deployed we had to make separate runs as the tool introduces major overhead by its monitoring operations. However with the profiling we can get detailed information about the behavior, performance and memory usage of the modules of the persistence component. We have chosen to monitor the persistence component in details for the most simple test case. We have performed again an automated test with the SEE++ client, however at this time the persistence component was deployed into a web service container (Tomcat) having the TPTP tool attached to it.

During the test, the execution flow is the following:

1. The requests are first processed by the web service facade.
2. For read-only, query operations go to step 5.
3. Map data from the XML structures of the transport model into binary representation.
4. The transport model is transformed into the internal metamodel based representation.
5. The necessary operations are performed by Hibernate on the actual database manager.

All these steps are performed by the modules in the package *at.uarmi.seegrid,* however these might use third party packages, such as the jdbc driver (*com.mysql.jdbc*) or the ones specific to the spring framework (*net.sf*). Detailed profiling data on which we based our conclusions are presented in Appendix A for the execution time and in Appendix B for the memory analysis.

Highest 10 base time

Package	<Base Time (seconds)	Average Base Time (seconds)	Cumulative Time (seconds)	Calls
com.mysql.jdbc	228,512141	0,000020	307,286947	11284396
com.mysql.jdbc.util	78,747862	0,000055	78,747862	1443849
at.uarmi.seegrid.transformations	55,858571	0,000547	312,860829	102111
at.uarmi.seegrid.dao.hibernate	14,716115	0,001850	91,935966	7953
at.uarmi.seegrid.hibernate	13,423752	0,000022	13,423752	597819
at.uarmi.seegrid.persistence	10,622481	0,000224	418,889351	47406
net.sf.cglib.core	7,200442	0,000007	8,307267	980238
at.uarmi.seegrid.domainmodel.da	4,271069	0,039547	369,017934	108
at.uarmi.seegrid.security.core	1,364043	0,003240	396,490188	421
net.sf.cglib.beans	1,304900	0,000020	13,946124	66877

Figure 8: Profiling session summary, provided by TPTP, with the execution time data for the sample data set with 2 scenarios

In Figure 8 we can see the time profiling data of the persistence component and other auxiliary modules integrated into it. The above mentioned packages are presented on the figure. The different columns of Figure 8 have the following meaning [tptp]:

- Base time – The time spent executing a particular method. Base time does not include time spent in other Java methods that methods of the respective package are calling.
- Average base time – the average base time per call.

- Cumulative Time – The amount of execution time, including the execution time of any other methods called from methods in the respective package.
- Calls – the number of calls for methods of the classes from the corresponding packages.

In this case not the time values themselves are interesting, but the proportions between them. The highest number of calls and the highest base time belongs to the jdbc module, it consumes more than 54% of the total execution time (calculated from the statistics of the data of all packages, not the just the ones present in Figure 8). The database driver can receive direct calls only from the *hibernate* module, thus the main performance bottleneck of the software are the object/relational mapping operations performed by this component. The second most time and resource consuming operation is the transport model representation – internal data representation transformation (performed by the *at.uarmi.seegrid.transformations* package), with around 10% of the base times of all packages.

Package	Total Instances	Live Instances	Collected	<Total Size (bytes)	Active Size (bytes)
com.mysql.jdbc	295904	1477	294427	23091408	114280
net.sf.ehcache	4327	104	4223	242312	5824
at.uarmi.seegrid.hibernate	6707	1839	4868	207984	57128
at.uarmi.seegrid.transformations	8778	0	8778	200664	0
at.uarmi.seegrid.persistence	2662	1046	1616	160536	62192
net.sf.cglib.proxy	968	21	947	15488	336
java.lang	110	110	0	10560	10560
com.mysql.jdbc.log	90	15	75	1440	240
com.mysql.jdbc.util	45	5	40	1440	160
at.uarmi.seegrid.security.datamodel	45	5	40	1440	160
at.uarmi.seegrid.security.core	9	1	8	144	16
at.uarmi.seegrid.persistence.implementation	3	0	3	48	0

Figure 9: Profiling memory analysis statistics, provided by TPTP, for the test case with the sample data set with 2 scenarios

In the next phase of the profiling we have collected memory usage information for the same scenario as in the execution time analysis. A package level view of the results is presented in Figure 9.

The data in the different columns have the following meanings [tptp]:

- Total instances – the total number of objects instantiated from the respective package.
- Live Instances – the number of objects in memory, when the monitoring has finished.
- Collected – the number of objects collected by the garbage collector.
- Total size – the total size of all instantiated objects.
- Active size – the size of live instances.

As we could have expected, the jdbc operations require the most memory, having a 45% share of the active objects and a 96% of the total instances. The difference between these two values is the result of the garbage collection. This high percentage is not reached because of the inefficiency of the jdbc driver but because of the large number of calls which imply that more than 92% (295904 instances) of the overall objects instantiated belong to the jdbc package.

Moving further on we analyzed the work of the MySQL database manager, where finally all the above mentioned JDBC calls will trigger some activities. Some of these data can be already found in the previous section, here we highlight some additional details. This test case has produced a total number of 15 979 SQL queries, generating 12 MB of network traffic. Most of the operations were insert (1398), select (3339) and update (1003) commands, the rest of them being various stmt commands (close, execute and prepare) which overlap with the previous commands as they cover all the operations completed via statement handles.

4.6. Results Evaluation

Based on the above analysis we can conclude that our application has two main bottlenecks: the database operations requested by the object-relational mapping and the object graph transformation to and from the transport model. These two major tasks use up more than two thirds of the resources. The web service facade and the security module's operations are quite fast. The remote delegate classes, related to the web service facade's connection to the actual persistence component, use 1,83% of the total base time. This is around 10 seconds, for 13 calls, in the case of the test scenario with the smallest data set, having an acceptable average of 0,8 seconds per call (with profiling overhead). Via these calls the client can access all the features of the persistence component, which are the search, read, write, update and delete operations for patient and medical data. The execution time (base time) spent by the security module's packages is less then 0,5% of the total base time.

The operations of the O/R mapping represent around 60% of the runtime of a test. The object graph transformations need more than 10% of the runtime. These two tasks also have the highest memory usage. The rest of the execution time is spread among other smaller tasks. Taking into account that both the transport model and the metamodel based representations are quite complex with a 3-4 level tree based organization, we deduce that this complexity should be the main cause of the poor performance. Because the software internally does not operate with the transport model at all, we can see that the performance of the object-relational mapping is influenced the most by the internal representation.

Possible improvement could be reached by the usage of a simpler metamodel. A more optimized database access could also improve the performance. The usage of an object oriented database would completely eliminate the object-relational mapping so in this way the performance would depend mostly on the database manager system.

5. Data Resource Access Middleware for SEE-GRID

We need to integrate the SEE-GRID persistence component in a Grid environment. For this we need to design an access interface for data resources, which will eventually use higher level Grid services. These Grid services can be provided by special middleware solutions, such as the ones mentioned in Section 2.4. The appropriate middleware solution for the integration of Grid database resources should fit in the scope of the SEE-KID project by offering various features. Our most important requirements are related to client APIs, back-end database support and Grid middleware support. With respect to these conditions, we have considered to evaluate the OGSA-DAI and AMGA solutions in the following sections. Based on this evaluation we formulate recommendations for the design of a Grid-aware persistence component in this chapter.

5.1. Evaluation of OGSA-DAI

The main goal of the OGSA-DAI project [OGSA-DAI2008] is to develop a free middleware tool to support the access to various databases through web services, in particular in Grid environments. These services are intended to provide data integration services to clients. In this section we evaluate some important features and capabilities of this software, taking into account the needs of the SEE-GRID project regarding to support for clients, back-end databases, security and possible limitations.

5.1.1. Overview

The OGSA-DAI project works in collaboration with the Globus project. Also thanks to this, the OGSA-DAI web services are compliant with the Web Services Resource Framework (WSRF) but there also exists an OGSI compliant version for older Globus Toolkit releases and a WS-I version based on Apache Axis. These different versions were not designed to interoperate. The access to data is independent from these versions and the actual resources, however the clients can also make resource specific operations [OGSA-DAI-DevGuide2008].

5.1.2. Installation

OGSA-DAI can be deployed and used easily with the Globus Toolkit. The installation process is not very complicated and there is good support provided in the chapter “System Administrator's Guide” of [OGSA-DAI-DevGuide2008]. Standard deployment can be performed by building and using a WAR file, in the case of Tomcat, or a GAR file for the Globus Java WS. It is also possible to deploy OGSA-DAI in a non-standard way, by copying its files in the web

service container's appropriate folders. OGSA-DAI is not installed as part of the standard Globus Toolkit installation, however it can be found as a bundle in the installation directories.

5.1.3. Security

As OGSA-DAI was primarily designed to work with Globus Toolkit, it does not provide any additional Grid security features. However to access various database systems, usually a username/password based authentication is required. In OGSA-DAI this is solved by the so called role handlers. Role handlers can map the Grid credentials to database authentication information. This can be done with the usage of a simple text file or there is also an advanced solution, where the mapping information is encrypted. It is also possible to develop custom role mappers by implementing a specific interface.

5.1.4. Client Support

Clients can access the data resources through the OGSA-DAI data services which can expose 0, 1 or more data service resources. The requests to web services have a uniform format, regardless of the type of data resources. The services can be accessed by any OGSI-, WSRF or WS-I-compliant client. It is also possible to have a higher level interaction through the Client Toolkit, which offers a Java API for the easy development of more sophisticated clients. It is important to note that this is not yet provided for the latest WSRF OGSA-DAI version [OGSA-DAI-DevGuide2008]. The requests and responses, regardless of the types of the client or the server, are SOAP messages over HTTP.

5.1.5. Back-end Data Source Support

OGSA-DAI offers support for various solutions to be used as data sources, which can be relational, XML or file based. The documentation [OGSA-DAI2008] mentions a few data resource types which have been successfully tested with the current release and are officially supported. These are: MySQL 5.0.15 and above, PostgreSQL 8.1.4, IBM DB2 8.1, Oracle 10g, MS SQLServer 2000, Derby, Xindice 1.0, eXist 1.1.1, Unix, Linux and Windows file systems. HSQL 1.7.1 and Apache Xindice 1.1 are also known to work with OGSA-DAI as well as other solutions which have appropriate JDBC or XMLDB drivers, but these are not officially supported. There also exist so called OGSA-DAI resource groups which represent data resource federations. They can be used in a similar way as other resources. These federations or resource groups can be formed of multiple heterogeneous data resources but logically can be accessed as single data resources.

The interaction with data resources is realized via three main building blocks [OGSA-DAI-

DevGuide2008]: activities, perform documents and response documents. Activities are basic operations for data manipulation, transformation and delivery both on client and server sides. There are predefined activities but it is also possible to develop application specific ones. Predefined activities can be categorized as query activities, transformation activities and delivery activities. It is interesting to note that transformation activities can convert data between different formats such as Row Sets, XML format or simple tuples. Activities can be collected together and they form in this way the so called “perform documents”, so they can be executed in a single workflow. Perform documents can be produced on the client side and then submitted as requests. Response documents are the responses returned by the OGSA-DAI services to the clients. These can contain information about the execution status of the current operations as well as the return data for the requests. Multiple requests can share states which are implemented as named contexts [Antonioletti2007].

5.1.6. Known Problems and Limitations

Known problems and limitations are published in Appendix D of the documentation [OGSA-DAI-DevGuide2008]. Some of the database related limitations are the following: OGSA-DAI does not support the creation or deletion of databases within DB2 and Oracle; OGSA-DAI does not currently support the addition of BLOBs (BLock Of Bytes) to tables in Oracle databases. Another issue with persisting larger amounts of binary data, BLOBs or CLOBs (Character Large Object), is that they are held only in memory by the OGSA-DAI modules what can risk out of memory errors in the container. Meta-data can be case-sensitive for data returned by different database back-ends. In certain scenarios the container can run out of memory with some third-party services deployed in the same container (when using tomcat) and for the synchronous transfer of very large data sets. It is not specified the threshold for the size of data sets as it might depend on the underlying systems, but the problem can be overcome by returning large datasets via FTP, GridFTP or asynchronous streaming. An additional source of problems can be that there is no full backwards compatibility of the new OGSA-DAI versions and there are also some deprecated features. Some modules and features are not yet available for the latest release and their usage might not be possible just with the older releases, with the recommendation to migrate to the new versions once it becomes possible.

5.1.7. Other Projects' Experiences

There is a list maintained on the OGSA-DAI home page [OGSA-DAI2008] presenting a large number of projects (over 40) which are using or have used their product. Most of these are

medical, biological, astronomical or geographical Grid projects. The needs of these projects are generally related to query, bridge and process large, possibly heterogeneous data sets, distributed over several data storage sites. Main reasons of choosing OGSA-DAI are its support for the widespread Globus Toolkit and the integration and federation of heterogeneous data sources by the abstraction of these as OGSA-DAI data services.

5.1.8. Future Outlook

The development of OGSA-DAI is continuous, so far there were regular releases approximatively every 6 months, except for version 3, which required 12 months as it was a full top-to-bottom re-design. For the future it is planned to increase the system's scalability, implement new standards, such as the Web Services Data Access and Integration standards of the Global Grid Forum, overcome current limitations and to port OGSA-DAI to other Grid middleware platforms, like the Unicore/GS and gLite.

5.2. Evaluation of AMGA

The AMGA metadata catalog is developed in the frame of the ARDA project studying the requirements for such technologies in a Grid environment [AMGA2008]. Initially it was a prototype implementation for the specifications resulting from the previously mentioned project. Later it has evolved jointly with gLite and has become the official EGEE metadata interface [Santos2005].

5.2.1. Overview

The main goal of the ARDA metadata project was to define the requirements for a Grid metadata catalog interface. Metadata information is seen as data modeled by *(key, value)* pairs with type information [Santos2005]. These data has to be structured and described by dynamic schemes so applications can define easily the structure of their own data collections and eventually change it dynamically. AMGA can be run as an add on to file catalogs, and manage file related metadata or to simply manage and replicate arbitrary structured data [AMGA2008]. AMGA uses a hierarchical file-system model to represent the structure of data. Because of this model, data schemes are also referred as directories although the storage is normally a relational database [Santos2005]. Another important requirement was scalability, because many Grid projects have to handle large data sets with several millions of entries. The solution for scalability in AMGA is based on master-slave replication mechanisms and enforced by support for asynchronous data transfer [Santos2006]. In the next sections we analyze also other important aspects such as the support and ways of interaction with clients, support for back-end

databases, security and possible limitations of the software.

All these tools and modules of the software are free and available for download, together with the appropriate documentation, from the main web page of AMGA [AMGA2008].

5.2.2. Installation

The installation of AMGA can be done using an RPM package or it is also possible to install by building it from its source. The first option however is officially supported just for CERN Scientific Linux and RedHat Enterprise Linux. This is valid both for the server and client parts of the software but there are also platform-independent Java clients which does not require source installation on other systems. The server also needs a database system to be previously installed and it requires ODBC drivers for that, because it is implemented in C++. Descriptions of the installation process can be found in the documentation section on the project's web page [AMGA2008].

Installation should be followed by the configuration of the server and the replication daemon. By default the server configuration is done via a single configuration file having different sections. Detailed description about the configuration options can be found in the documentation [AMGA2008].

5.2.3. Security

AMGA is shipped with a user management system which controls the authorization to access database entries and metadata. Users can be grouped and every user is allowed to create new groups. Permissions are maintained in POSIX-like permission schemes. When AMGA is used on a file catalog access control is overwritten by the controls of the respective file catalog [AMGA2008].

Each client must be authorized to use the role of an AMGA user. Authentication can happen either on a certificate based or a password based way, depending on the actual configuration, but each option must be explicitly enabled. Certificate based authentication requires to use SSL both on the client and server side. Other connections can also be secured using SSL. Furthermore once a user has performed authentication, depending on the current configuration, it may use any role he wishes or additional authorization can be enabled on the server side. Authorization options are defined in a configuration file *(mdserver.config)*. The only password based option is to setup a user database on a database back-end; the other options are based on using a VOMS or VOMS certificates. Certificate subjects optionally can be mapped based on the grid-map file.

All the database entries have attributes inherited from their corresponding schemas. It is not

possible to directly set attributes for individual entries but different schemas can be defined for different logical groups of attributes. Schemas can be defined dynamically but each entry must be associated to at least one schema [Santos2005].

5.2.4. Client Support

Clients can interact with the AMGA server via a command line or applications can use client APIs and there is also a special AMGA client-server protocol, based on ASCII stream communication. The server has two front-ends, a SOAP based one and a TCP streaming based one.

C++, Java and Python clients and the C++ API can be installed with RPM packages. The server package depends on the client package but the different client versions can also be installed independently. There is also a configuration file for clients (*mdclient.config*), which is present in each work directory. In this way we can provide custom configuration for the different users. The installed command line tool comes in two versions. There is a command line tool which can be used as a utility for an application to submit a single command to the server and there is also an interactive command line shell.

There are also additional Perl, Ruby and PHP API client modules and there is also available a Web Frontend, developed separately by INFN Catania (http://grid.ct.infn.it/).

For the clients there are available several commands to search, query, write and update databases. Transactions are supported and they can be eventually committed or aborted. The AMGA interface does not define any specific language for querying because different implementations might have different needs and features. Queries are submitted with two parameters, a string, representing the query itself, and second, the name of the language. There are also file operations available, more useful when AMGA extends file catalogs. The different attributes of directories can be manipulated by the clients, both from command line or through the API. Index and key operations are supported. Views are also supported but their representation depends on the actual database back-end. There is a feature present to create unique IDs with the so called sequences. Back-end independent backups can be created by a feature which is capable to dump the stored data into AMGA command sequences, so they can be restored later. There are also a few commands to manage and maintain a list of replicated databases. These databases are called simply sites. Script execution is also allowed under the credentials of a predefined user-id, offering in this way a CGI-like functionality.

AMGA tests show that the TCP-streaming based communication compared to the SOAP based frontend is performing much better. Benchmarks have shown that bulk reading can be up

to ten times faster while arbitrary queries can be two to five times faster by using TCP-streams [Santos2005].

Speed and efficiency is further increased by asynchronous data delivery to clients. AMGA uses cursors on relational databases to access the results. Data is read in chunks into a local buffer and transmitted to clients on request. The connection with the clients is stateful and database connections are kept open during one session with certain restrictions on the maximum number of sessions and idle time [Santos2005].

5.2.5. Back-end Data Source Support

Back-end databases can be connected to AMGA via ODBC drivers. Currently the system supports PostgreSQL, MySQL, Oracle and SQLite. The default is PostgreSQL and this one is required to be set-up when installing from RPM. For other databases one has to follow special installation instructions [AMGA2008]. The setup of replica and/or master nodes is easy, it is just required to enable/disable master, respectively slave modes for the actual node, setup an open port for the replication daemon and provide a node name in the configuration file. It is the responsibility of the administrator to define unique node names among the nodes of a system [AMGA2008].

There is also a standalone AMGA implementation that uses as database back-end directly the file-system [Santos2005].

5.2.6. Known Problems and Limitations

In the standard documentation there is no chapter dedicated to describe known problems and limitations. These can be figured out by investigating, based on the documentation, the possibility of implementation for specific needs.

Some known limitations are shown in the presentation of replication mechanisms [Santos2006]. The reliability depends on the failure of master nodes, and distributed updates are difficult to perform. The installation guide [AMGA2008] mentions that if AMGA is using the grid-map, the configuration becomes static, and the authentication configuration can be updated just by restarting the application. The documentation and benchmarks state that the SOAP based communication is generally slower, especially for the work with large datasets, the use of the TCP streaming communication is suggested [Santos2006].

Already fixed bugs and new features are described in the changelog published on the AMGA web site [AMGA2008].

5.2.7. Other Projects' Experiences

The most important partner in the early development and evaluation of AMGA was the LHCb [LHCb2008], the Large Hadron Collider beauty experiment. The experiment requires to persist bookkeeping information of more than 20 million entries in data catalogs, representing around 15GB of data. These information come from jobs executed on the Grid and are static metadata of large amount.

A different project using AMGA is the Ganga [Ganga2003]. Ganga is a user interface giving access to the Grid and providing job-configuration and data-management functions. It stores metadata about the status of jobs submitted to the Grid. Since the status of jobs changes very often, in this case the stored data is of smaller amount but it experiences highly dynamic changes.

These two projects were the early testers of AMGA and have experienced success with the product. Since then it become known by other user communities. An example is Biomed, a medical data manager, which works with medical images and sensitive patient data so encryption and security are important aspects. It uses AMGA as a simple database access interface, being deployed on the EGEE production Grid.

There is no official overview about the users of AMGA, but there are several other users not just the above mentioned ones. These projects have good results and their requirements are various indicating that AMGA is a versatile product. The number of users is expected to grow as AMGA has become the official EGEE metadata interface implementation.

5.2.8. Future Outlook

The development team plans to cope with the growing number of users and focus on the scalability and fault-tolerance of AMGA. Fault-tolerance comes more important on the failure of master-nodes, in which case the whole system might be able to operate only in read-only mode. Further improvement can be expected in the performance of communication between the different database nodes of a system as well as in the reliability of transmissions. In a next phase it is planned to provide support for distributed updates by extending the current master-slave model. [Santos2006]

5.3. Evaluation Conclusions

Based on the above evaluation results we can see that both products can offer an alternative for the further development of the SEE-GRID project, but there are also important differences between them.

Both products are free software and developers are allowed and encouraged to adapt or extend them for their specific needs if required. OGSA-DAI has different versions based on different standards, such as the OGSI-, WSRF or WS-I, but all of these are web service flavors. We have to note that these versions can not inter-operate and different features and utilities (like the Client Toolkit API) might not be compatible with all the latest releases of the core product. AMGA does not have such problems however it has only two front-ends, incorporating both the SOAP and TCP stream based solutions in the same time. As AMGA is not exclusively web service-based it also offers a standalone version.

Comparing their security support we can see that AMGA has a more sophisticated solution, offering a larger variety of features compared to the simple role mapping applied in OGSA-DAI. It is also important to mention that with AMGA's user management system it is possible to define different access levels even inside the structure of one data schema.

Regarding the client support we can state that AMGA has a larger variety of client APIs while OGSA-DAI is more sticked to Java. Potentially any client can be used with both solutions as long as it can handle SOAP messages or also TCP streams in the case of AMGA. In change OGSA-DAI offers more sophisticated operations and features such as the perform documents and the data transformation activities which are not offered by AMGA.

Analyzing the back-end data source support we can note that AMGA is exclusively operating with solutions using well structured data, relational databases and optionally the filesystem based persistence. OGSA-DAI is more dynamic from this point of view, it offers good support not only for relational databases but also for XML databases and data handling and in the same time it also covers the file-system based requirements. Both systems support the data resource federations but AMGA applied a replication mechanism on the middleware layer. Both systems have recognized the need for asynchronous data transfers, the only reliable way to handle large data sets. In this view AMGA is more advanced by its TCP stream front-end and by applying cursor based data retrieval from the relational databases, avoiding memory usage problems in this way. In change OGSA-DAI has built-in support for FTP and GridFTP and in some cases it might be possible to use asynchronous data streams to return large data sets to clients but this latter feature is not yet present in all versions.

Both products are used by several Grid projects and can show convincing results which may indicate that they have reached a certain level of maturity and are stable. OGSA-DAI is more linked to the Globus Toolkit but in the future it might be ported to other environments also, however this might happen just after a longer time period. The deployment of AMGA is linked to gLite and has official support from EGEE. Both can be used outside the Grid, OGSA-DAI with

Axis while AMGA offers a standalone version.

In the next step of development it is required to precisely define the future lines of conduct of the SEE-GRID project. The definition of requirements shall take into account different aspects: the type of the database systems which would be preferred to be used in the future; the Grid middleware for the next version of the persistence component; evaluate the usage of TCP stream and/or SOAP based communication; specify the authentication and authorization requirements; evaluate the need for database replication and the level of scalability.

Taking into account the evaluation results, with the development team of SEE-GRID we decided to extended the architecture of the database layer based on a Globus web service interface and OGSA-DAI. The future integration into gLite is also desirable, however the possible usage of XML databases and the already existing SEE-GRID components integrated into Globus have motivated mainly our choice.

In the following chapter we present the extended design of SEE-GRID, our prototype of the Globus web service interface and our experience with deploying the already existing medical database with OGSA-DAI.

6. Extended SEE-GRID Architecture

We present in this chapter the extended design of SEE-GRID. We have provided a Globus web service interface for the database and we also present our experience about the possible integration of OGSA-DAI with the persistence component.

The next two sections contain a description about prototyping the web service interface and our first results with OGSA-DAI. In the last section we provide an overview of the whole design and we describe the main ideas of the planned integration of the SEE-GRID medical database into the Grid. Since this integration is already in progress, in the overview, beside the design concepts, we also give some additional implementation details.

6.1. Globus Web Service Interface for the SEE-GRID Database

Our prototype implementation is based on the Globus Java Web Service Resource Framework (WSRF) as the persistence component itself is also implemented in Java. This version acts as a facade to the persistence component, simply forwarding all the requests to the main persistence implementation. We have not changed the original security concept of SEE-GRID with its username/password based authentication. For the future the transport level security and a certificate based authentication should be implemented. This is essentially a simple task on the server side (see Part III of [Sotomayor2005]) but it also needs appropriate support from the clients, respectively from the SEE++ to Grid bridge.

Section III-A of [Bosa2007] indicates that the WSRF-based extension of the SEE++ database could not be completed because of some restrictions from the side of Globus while generating ANSI C stubs for data structures containing the so called "*SOAP Encoded Array"* data type. We have investigated the problem and concluded that this is a general restriction in the Globus WSRF because it follows the WS-I Basic Profile standards from OASIS [OASIS_WS-I2008]. This standard has several restrictions on array types because of interoperability problems raised by different interpretations of the WSDL (Web Service Description Language) recommendations. By now the SEE-KID team has adapted SEE++ in such a way that its web service interface use no more "*SOAP Encoded Arrays"* thus now it respects the corresponding standards of the Globus Toolkit.

6.2. Experiences with OGSA-DAI

The choice of making experiments with OGSA-DAI in our project had multiple reasons. Since we wanted to further investigate in practice with the Globus Toolkit, OGSA-DAI was the best choice based on our evaluations. As this Grid data-resource manager supports MySQL as a

back-end it makes possible to easily integrate the already existing medical database. Its XML support also provides good outlook for the project's future plans to investigate the usage of this technology.

Our experience shows that essentially it is easy to deploy OGSA-DAI with the Globus Toolkit and the available documentation gives satisfying support for this. We used the latest available stable releases of these software, namely Globus 4.0.7 and OGSA-DAI 3.0. OGSA-DAI exposes its services through a WSRF based web service interface, completely implemented in Java [OGSA-DAI-DevGuide2008]. Since we also implemented the Grid-interface of the persistence component as a Java WSRF web service, this seems to be beneficial for the performance. The interaction between services deployed in the same container can be carried out locally, skipping the network, thus faster in this way.

In a first phase it is planned to integrate OGSA-DAI's services in the persistence component in a simple way, by directly forwarding database queries. Because the internal architecture of the persistence component is quite complex we have decided to investigate the applicability of this design in an indirect way. As a first step we have implemented a standalone Java client, which invokes the OGSA-DAI services in a similar way as the persistence component would, however in this case working just with the MySQL database. This showed that it is possible to give a satisfactory implementation and in the future the full integration can be completed.

Another possibility for the future development is the usage of XML database back-ends. A possible approach to improve the performance of the interaction between clients and the persistence component is to provide a two-level architecture. The first level would be a custom solution using an XML back-end, as the SEE++ rich client can already handle patient data in this format. This level would then replicate the stored data, perhaps in an anonymized form, to the second level. The second level would keep the general metamodel based approach and its task would be limited then to the non-time critical data mining operations. OGSA-DAI also provides us good opportunities here by providing data management services for both the half-structured XML back-end as well as for the already existing relational database.

6.3. Design Overview

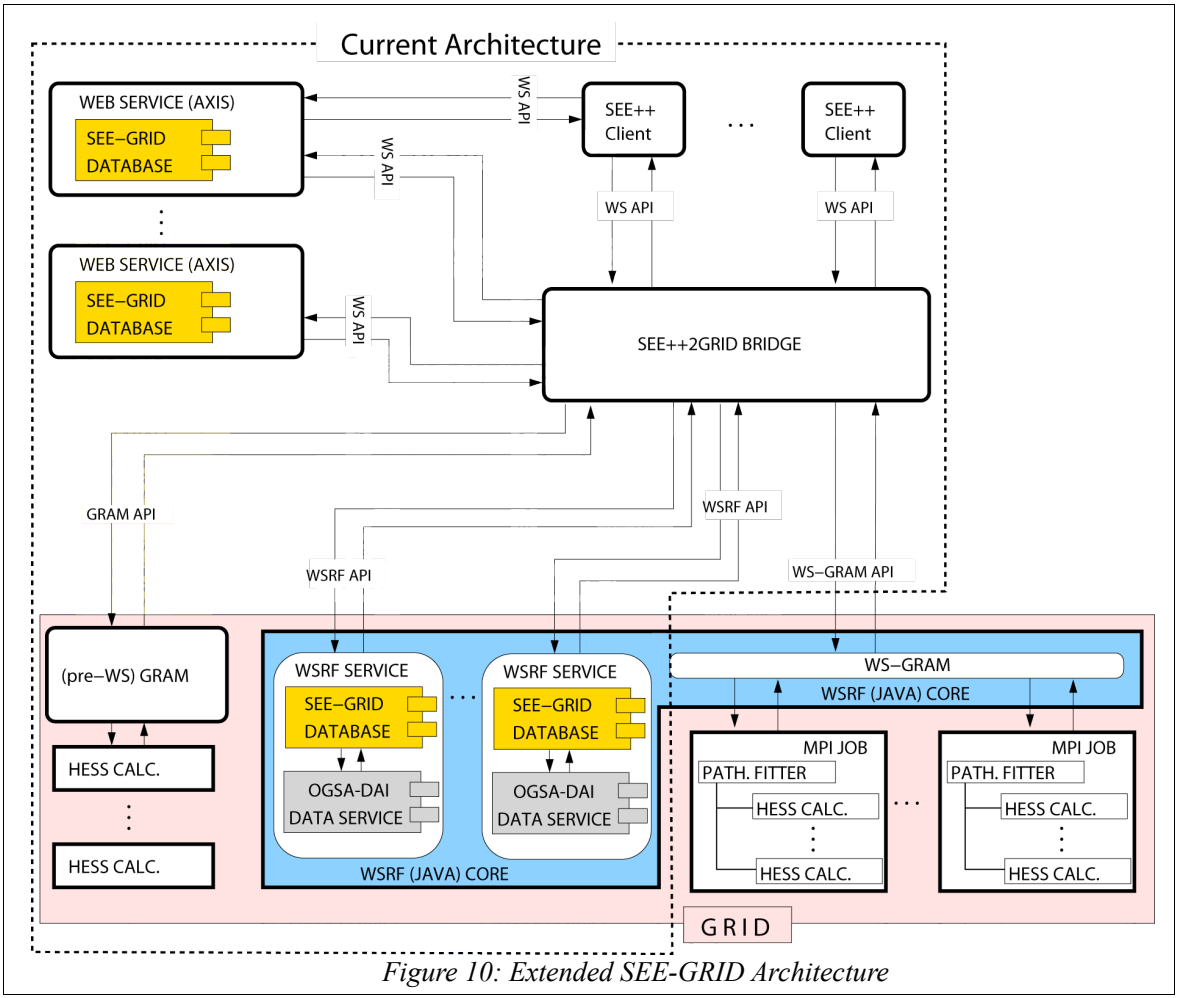

Figure 10: Extended SEE-GRID Architecture

We present the extended SEE-GRID architecture in Figure 10, which is mainly based on the design presented in [Bosa2007], however in this thesis we just focused on the database layer. The figure also includes the planned Java web service interface for the pathology fitter (see Section II-C of [Bosa2007]), while the dashed line denotes components which are already implemented or are currently under development.

The Axis based web services represent the current database implementation which works outside the Grid. These web services can be deployed in common containers such as the Apache Tomcat. The SEE++ clients can communicate with these via SOAP messages with the help of a WS API, as indicated in the figure. These clients can also connect to the SEE-KID calculation server, which exposes its operations through a web service interface, providing in this way the possibility to use many different client implementations [SEE-KID2008]. The *SEE++2GRID BRIDGE* component makes use of this already existing feature of the clients and acts as a server for them but in the same time it also acts as a client for the Grid, forwarding all the requests to the corresponding Grid services [Bosa2007]. This approach implies that the clients doesn't have

to be aware that they are using Grid services, they can continue to use the simple WS API for communication and the bridge component implements the Grid calls through the WSRF API provided by the Globus Toolkit. This does not introduce a too high overhead, because the format of the WS API and WSRF API operations are similar, since WSRF can be rather seen as a specific extension of the common web services.

Since the restrictions noticed in Section III-A of [Bosa2007] were overcome in SEE-KID by the replacement of the so called "*SOAP Encoded Arrays"* with other custom data types which comply with the standards applied by the WSRF [OASIS_WS-I2008], the development now can be continued. Our prototype presented in Section 6.1 proves that the *SEE++2GRID BRIDGE* component can be extended now to enable the forwarding of database requests from the clients to the Grid services. In conformity with the design presented here, the database requests should be then forwarded to the SEE-GRID Database's WSRF interface which can be deployed on the Grid into web service containers supporting Globus' Java WSRF Core (see the section marked *Grid* in Figure 10).

The already existing modules of the SEE-GRID persistence component can be easily integrated into a Globus web service. These WSRF web services are normally included with all their dependencies and necessary configuration files in a so called *Grid ARchive* or GAR file [Sotomayor2005], similar to the WAR or *Web ARchives*. In order to be able to port the persistence component to the Grid we have to include the persistence component and all its specific dependencies, such as the Spring Framework or Hibernate, into this GAR file. In this way the web service logic can easily invoke the methods of the persistence component which can remain unchanged.

In order to use OGSA-DAI in an efficient way, in our design we show that it should be deployed in the same container as the database's Grid interface to be able to use internal calls for calls between the two services. Standard deployment of OGSA-DAI is easy, since it is available as a GAR file for the Globus Java WS container or even a WAR file can be built for Apache Tomcat [OGSA-DAI-DevGuide2008]. In this way the usage of OGSA-DAI services require only to implement a special client module for it based on the WSRF API, which should be integrated into the persistence component.

As a first step the existing MySQL medical database should be integrated into OGSA-DAI and the queries of the persistence component, generated by Hibernate, could be then sent to the OGSA-DAI data service. However this would limit for the moment the usage of back-ends to MySQL but extensions can be made to any other back-end supported by Hibernate. The advantage of this approach is that the persistence component needs no more to connect directly

to a single data-source. We can deploy multiple data sources which can be managed by OGSA-DAI and furthermore data-resource-federations can be created and distributed queries applied on those.

A second possible approach is to configure OGSA-DAI to use an XML database back-end (such as Exist). This would require to simplify or re-implement the persistence component. The database layer would extract the medical data from the SOAP messages directly in XML format and persist them and also it could retrieve the data from the back-end in XML format. In this case the existing relational database still could be used in a read-only manner because OGSA-DAI is capable to transform the retrieved data into XML format. The SEE++ client can already handle XML format data so it just has to be adapted to create and receive messages in a corresponding format. The main advantage of this approach is that we obtain a light, perhaps very fast service which is capable to interact with the clients in real-time, solving in this way the performance issues of the current implementation. However the metamodel based implementation could be still kept, as it is a general solution, not customized for SEE++. Thus, as a long term goal, data mining operations designed for general medical applications could be realized on it. In this case the customized SEE-GRID database should replicate impersonated medical evidence into the metamodel based repository which may store then medical data from heterogeneous medical applications.

The security features of the Globus Toolkit have to be considered in every component which directly makes or receives web service calls. This means that both the client and server components can be and should be implemented or adapted in such a way that they would apply the transport level security and the certificate based authentication of Globus. As a first step this can include just the bridge component and the actual Grid web services, which may apply secure network level transport for the calls and use a Grid certificate dedicated for a specific instance of a bridge component. Meanwhile the persistence component would still rely on its username and password based authentication and authorization. Later this can be upgraded to either map the already existing SEE-GRID users with certificates or also change the security infrastructure on the client side and apply only the Grid certificate-based security on all levels of the software system.

The other services present on Figure 10 are not directly connected to the database. There is a pre-WS implementation of gaze pattern calculations (Hess-Lancaster test simulation) and the planned parallel pathology fitter component. In the future this latter may also be extended in such a way that it will need to query multiple SEE-GRID databases for specific pathologies to improve the speed and accuracy of its computations (see Section 4.2 of [Bosa2005a]).

7. Conclusions

In this chapter we present the goals achieved by this work and its applicability and relation with the possible ways for the future development of the SEE-GRID database system, based on Grid technologies.

7.1. Achieved Goals

The main goals of this thesis were the performance analysis of the current implementation and the evaluation of state of the art Grid database solutions respect to their applicability for the SEE-GRID database.

Performance is important for SEE-GRID as the current implementation can not be used in "real-world scenarios" because of its limitations. We have performed an extended benchmarking and profiling of the SEE-GRID database system. The design, results and conclusions of this performance analysis are presented in Chapter 4. As the most important is the product's interaction with the users, we have decided to do a benchmark by the automation of the user interface of SEE++. We have also realized a profiling of the persistence component. We have gathered information about the behavior of the different modules of the persistence component, respect to memory usage, the number of instantiated and live objects, the share of runtime for each. We also collected information from the underlying database manager, about the amount of network traffic and number and types of operations requested by the persistence component. This work has revealed many interesting aspects. Based on our results (Sections 4.4, 4.5) we have concluded that the main performance bottleneck of the software is the over-engineered, very complex metamodel which causes a non-linear decrease in performance as we increase the size of the processed data. The optimization of the metamodel and of the data transformation process would bring notable performance increase. The replacement of the metamodel based approach, in the direct interaction with clients, with a fast, custom solution (such as an XML database manager) could also provide very good performance. This latter approach can be interesting also because SEE++ can already handle data in XML format.

In Section 2.4 we have briefly presented a few Grid data-resource management tools. We have chosen to evaluate in detail the OGSA-DAI solution for the Globus environment, and the AMGA solution for the gLite Grid middleware as these are the two Grid environments currently used by the SEE-GRID project. A detailed evaluation and a brief comparison is described in Chapter 5 of the thesis. We have decided to make an extended design of the SEE-GRID database layer, this is presented in Chapter 6. The new Grid interface is based on high-level services of

Globus, such as the WSRF compliant web interface for the persistence component and the data management solution offered by OGSA-DAI. We have chosen the Globus Toolkit and OGSA-DAI because they seem to fit better the current needs of the project. The applicability of our new design is proven by our prototype implementation of the web service interface, described in Section 6.1, as well as by our practical results with OGSA-DAI presented in Section 6.2. However the future integration of the database into gLite should also be considered. In Section 6.3 we give a detailed overview of the extended design.

As the work carried out in the frame of this thesis is meant to be a contribution to an already existing project with an ongoing development, we have to mention that the current status is continuously changing. We currently work on the refinement of the prototype database Grid interface and the integration of OGSA-DAI with the medical database and with the persistence component is also ongoing.

7.2. Future Outlook

Taking into account the results of the performance analysis, the future development of the SEE-GRID database may split the system in two different parts. There might be a fast, custom solution for the direct interaction with clients. The local database then may submit anonymized medical data to the Grid. The actual Grid database could preserve a simplified metamodel based approach to keep its generality.

The Grid-database can form the basis for complex data mining operations. These operations could help, for an example, the automated pathology fitter algorithm to search for similar cases. These cases can be then set as the starting point for the iterations of the algorithm which may produce more accurate results in less time (see [Bosa2007]).

Another possible usage of data mining on the Grid could be a special service which should be able to provide for the doctors the possibility to search for and consult the medical data and evolution of patients with a similar pathology compared to a specific case selected by them. Doctors often have to consult and make their decisions based on the available medical evidence, so it is crucial to find the best available relevant evidence. Such a service, supporting evidence based medicine, is being developed based on the BurnCase 3D software (www.burncase.at) at the Medical Informatics Research Unit of RISC Software. Their first prototypes and results show that it is possible to design such a system independent from the medical domain, so in the future it would be interesting to collaborate and integrate such a service into the SEE-GRID database.

Security and privacy is also a very important aspect in the field of medical informatics. The available Grid toolkits, such as Globus or gLite, provide good support for secure Grid-

applications. The Grid based services of SEE-GRID may adapt in the future the transport level security, as well as the certificate-based authentication and authorization of users. The existing username and password based authentication of SEE-GRID can be replaced by a certificate based one, which is possible thanks to the modular design of the product. As an intermediate solution the mapping of SEE-GRID users to Grid users can also be considered.

Appendix A - Execution Time Analysis Data

Here we present the data collected by the execution time analysis of the Test & Performance Tools Platform (TPTP) with the profiling described in Section 4.5. The table contains the package level view of the results, including the packages of the persistence component as well as the packages from its dependencies.

Package	Base Time (seconds)	Average Base Time (seconds)	Cumulative Time (seconds)	Calls
com.mysql.jdbc	228,512141	0,000020	307,286947	11284396
com.mysql.jdbc.util	78,747862	0,000055	78,747862	1443849
at.uarmi.seegrid.transformations	55,858571	0,000547	312,860829	102111
at.uarmi.seegrid.dao.hibernate	14,716115	0,001850	91,935966	7953
at.uarmi.seegrid.hibernate	13,423752	0,000022	13,423752	597819
at.uarmi.seegrid.persistence	10,622481	0,000224	418,889351	47406
net.sf.cglib.core	7,200442	0,000007	8,307267	980238
at.uarmi.seegrid.domainmodel.dao .implementation	4,271069	0,039547	369,017934	108
at.uarmi.seegrid.security.core	1,364043	0,003240	396,490188	421
net.sf.cglib.beans	1,304900	0,000020	13,946124	66877
net.sf.cglib.proxy	1,179723	0,000016	7,647682	73033
at.uarmi.seegrid.utils	0,444784	0,000095	1,847383	4670
net.sf.cglib.reflect	0,344702	0,000014	4,524946	23789
net.sf.ehcache	0,233823	0,000008	0,394537	31016
net.sf.ehcache.config	0,117236	0,000075	0,118054	1561
at.uarmi.seegrid.security.datamodel	0,097651	0,000115	0,097651	850
net.sf.ehcache.store	0,043477	0,000008	0,060918	5448
com.mysql.jdbc.log	0,026944	0,000090	0,026944	298
at.uarmi.seegrid.persistence .implementation	0,019946	0,000322	369,025887	62
filters	0,000000	0,000000	0,000000	0

Notes:

- Base time – The time spent executing a particular method. Base time does not include time spent in other Java methods that methods of the respective package are calling.
- Average base time – the average base time per call.
- Cumulative Time – The amount of execution time, including the execution time of any other methods called from methods in the respective package.
- Calls – the number of calls for methods of the classes from the corresponding packages.

Appendix B - Basic Memory Analysis Data

The following table contains data collected by the basic memory analysis of the Test & Performance Tools Platform (TPTP) with the profiling described in Section 4.5. The table contains the package level view of the results, including the packages of the persistence component as well as the packages from its dependencies.

Package	Total Instances	Live Instances	Collected	Total Size (bytes)	Active Size (bytes)
com.mysql.jdbc	295904	1477	294427	23091408	114280
at.uarmi.seegrid.transformations	8778	0	8778	200664	0
at.uarmi.seegrid.hibernate	6707	1839	4868	207984	57128
net.sf.ehcache	4327	104	4223	242312	5824
at.uarmi.seegrid.persistence	2662	1046	1616	160536	62192
net.sf.cglib.proxy	968	21	947	15488	336
java.lang	110	110	0	10560	10560
com.mysql.jdbc.log	90	15	75	1440	240
com.mysql.jdbc.util	45	5	40	1440	160
at.uarmi.seegrid.security.datamodel	45	5	40	1440	160
at.uarmi.seegrid.security.core	9	1	8	144	16
at.uarmi.seegrid.persistence.implementation	3	0	3	48	0
at.uarmi.seegrid.utils	2	0	2	32	0

Notes:

- Total instances – the total number of objects instantiated from the respective package.
- Live Instances – the number of objects in memory, when the monitoring has finished.
- Collected – the number of objects collected by the garbage collector.
- Total size – the total size of all instantiated objects.
- Active size – the size of live instances.

Appendix C - MySQL Operations Statistics

The following table contains the relevant data for our evaluation related to MySQL operations, extracted from the phpMyAdmin platform for all of our test scenarios. The data from the *Monitor* column represents the overhead caused by the monitoring operations which we measured to see that it does not influence significantly the results. The other columns present the data corresponding to the benchmark cases with 2, 5 and 13 medical data scenarios.

Operation	**Monitor**	**2 scenarios**	**5 scenarios**	**13 scenarios**
Traffic				
Received	2,49 KB	3 025 KB	12288 KB	35840 KB
Sent	47 KB	9 633 KB	36864 KB	107520 KB
Connections (no)	9	61	61	61
Query stats				
total querys	68	15979	61432	182640
change db	2	3	3	3
commit	0	9	9	9
insert	0	1398	5550	17 k
select	9	3339	13 k	39 k
set option	16	202	202	202
show charsets	4	7	7	7
show collations	4	53	53	53
show databases	4	7	7	7
show tables	10	15	3	15
stmt close	0	4909	19 k	57 k
stmt execute	0	5725	23 k	67 k
stmt prepare	0	4909	19 k	57 k
update	0	1003	3985	12 k
Flush_commands	1	1	1	1
InnoDB (transactions, row-level locking, and foreign keys engine)				
buffer_pool_pages_data	38	495	586	852
buffer_pool_pages_dirty	0	0	0	0
buffer_pool_pages_flushed	0	192	360	610
buffer_pool_pages_free	922	461	366	89
buffer_pool_pages_latched	0	0	0	0
buffer_pool_pages_misc	0	4	8	19
buffer_pool_pages_total	960	960	960	960

Operation	Monitor	2 scenarios	5 scenarios	13 scenarios
buffer_pool_read_ahead_rnd	1	4	5	6
buffer_pool_read_ahead_seq	0	0	0	0
buffer_pool_read_requests	209	200 k	724 k	2 076 k
buffer_pool_reads	15	302	325	422
buffer_pool_wait_free	0	0	0	0
buffer_pool_write_requests	0	19 k	64 k	164 k
data_fsyncs	3	48	62	98
data_pending_fsyncs	0	0	0	0
data_pending_reads	0	0	0	0
data_pending_writes	0	0	0	0
data_read	2 806 k	9 900 k	10 M	13 M
data_reads	48	481	515	644
data_writes	3	235	416	703
data_written	1536	6 830 k	14 M	26 M
dblwr_pages_written	0	192	360	610
dblwr_writes	0	6	8	13
log_waits	0	0	0	0
log_write_requests	0	1192	4392	13 k
log_writes	1	29	39	59
os_log_fsyncs	3	36	45	69
os_log_pending_fsyncs	0	0	0	0
os_log_pending_writes	0	0	0	0
os_log_written	512	535 k	2 000 k	5 956 k
page_size	16 k	16 k	16 k	16 k
pages_created	0	24	81	218
pages_read	38	471	505	634
pages_written	0	192	360	610
row_lock_current_waits	0	0	0	0
row_lock_time	0	0	0	0
row_lock_time_avg	0	0	0	0
row_lock_time_max	0	0	0	0
row_lock_waits	0	0	0	0
rows_deleted	0	0	0	0
rows_inserted	0	1398	5550	17 k
rows_read	0	214 k	844 k	2 526 k
rows_updated	0	955	3796	11 k
Query cache				
Qcache_free_blocks	1	1	1	1

Operation	Monitor	2 scenarios	5 scenarios	13 scenarios
Qcache_free_memory	8 380 k	8 380 k	8 380 k	8 380 k
Qcache_hits	0	0	0	0
Qcache_inserts	0	0	0	0
Qcache_lowmem_prunes	0	0	0	0
Qcache_not_cached	42	3486	13 k	39 k
Qcache_queries_in_cache	0	0	0	0
Qcache_total_blocks	1	1	1	1

Bibliography

[AGrid2008] "Austrian Grid website - www.austriangrid.at", 2008.

[AMGA2008] "AMGA - The ARDA Metada Catalogue Project – http://amga.web.cern.ch/amga/", 2008.

[Andrieux2007] Andrieux, A., Czajkowski, K., Dan, A., Keahey, K., Ludwig, H., Kakata, T., Pruyne, J., Rofrano, J., Tuecke, S. and Xu, M. "Web Services Agreement Specification (WS-Agreement)", http://www.ogf.org/documents/GFD.107.pdf, 2007.

[Antonioletti2007] Antonioletti, M., Hong, N. P. C., Hume, A. C., Jackson, M., Karasavvas, K., Krause, A., Schopf, J. M., Atkinson, M. P., Dobrzelecki, B., Illingworth, M., McDonnell, N., Parsons, M. and Theocharopoulous, E. "OGSA-DAI 3.0 - The What's and Whys", in '*Proceedings of the UK e-Science All Hands Meeting*', 2007.

[autoit] "AutoIT web page - http://www.autoitscript.com/autoit3/", 2008.

[Banks2005] Banks, T., Djaoui, A., Parastatids, S., Mani, A., Tuecke, S., Czajkowski, K., Foster, I., Frey, J., Graham, S., Kesselman, C., Maguire, T., Sandholm, T., Snelling, D. and Vanderbilt, P. "Open Grid Service Infrastructure Primer", http://www.ogf.org/documents/GFD.31.pdf, 2005.

[Bosa2005] Bosa, K., Schreiner, W., Buchberger, M. and Kaltofen, T. "A Refined Design of the SEE-GRID Database and Pathology Fitter", Austrian Grid Deliverable *AG-DA1c-5-2005_v1*, Technical report, Research Institute for Symbolic Computation (RISC), Johannes Kepler University, Linz, Austria, 2005.

[Bosa2005a] Bosa, K., Schreiner, W., Buchberger, M. and Kaltofen, T. "SEE-GRID, A Grid-Based Medical Decision Support System for Eye Muscle Surgery", *in* 'Proceedings of 1st Austrian Grid Symposium 2005', *ed.* Volkert, J., et. al., Austrian Computer Society (OCG), Hagenberg, Austria, 2005, pp. 61--74.

[Bosa2007] Bosa, K., Schreiner, W., Buchberger, M. and Kaltofen, T. "A Grid Software for Virtual Eye Surgery Based on Globus 4 and gLite", *in* 'Proceedings of ISPDC 2007', ed. IEEE Computer Society, 2007

[Buchberger2004] Buchberger, M. "Biomechanical Modelling of the Human Eye", PhD Thesis, Johannes Kepler University Linz, Austria, 2004.

[Buchberger2008] Buchberger, M., Kaltofen, T. and Priglinger, S. "SEE++ User Manual", http://www.see-kid.at/download/SEEPP_Manual_ENG.pdf, Upper Austrian Research GmbH, 2008.

[DRS2008] "GT 4.0: Data Replication Service documentation - www.globus.org/toolkit/docs/4.0/techpreview/datarep/", 2008.

[Foster1999] Foster, I., Kesselman, C., Lee, C., Lindell, R., Nahrstedt, K. and Roy, A. "A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation", 'Proceedings of the International Workshop on Quality of Service', 1999.

[Foster2001] Foster, I., Kesselman, C. and Tuecke, S. "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", *Lecture Notes in Computer Science* (2150), 2001.

[Foster2005] Foster, I., Kishimoto, H., Savva, A., Berry, D., Djaoui, A., Grimshaw, A., Horn, B., Maciel, F., Siebenlist, F., Subramaniam, R., Treadwell, J. and Reich, J. V. "The Open Grid Services Architecture, Version 1.0", http://www.gridforum.org/documents/GWD-I-E/GFD-I.030.pdf 2005.

[Foster2006] Foster, I. "Globus Toolkit Version 4: Software for Service-Oriented Systems", 'IFIP International Conference on Network and Parallel Computing', Springer-Verlag, LNCS 3779, 2006, pp. 2--13.

[FosterKesselman1997] Foster, I. and Kesselman, C. "Globus: A Metacomputing Infrastructure Toolkit", *The International Journal of Supercomputer Applications and High Performance Computing* (11(2)), 1997, pp. 115--128.

[FosterKesselman1999] Foster, I. and Kesselman, C. "The Globus project: a status report," *Future Generation Computer Systems* (15:5--6), 1999, pp. 607--621.

[Fowler2004] Fowler, M. *Analysis Patterns: Reusable Object Models*, Addison-Wesley, 2004.

[Ganga2003] Harrison, K., Lavrijsen, W. T. L. P., Mato, P., Soroko, A., Tan, C. L., Tull, C. E., Brook, N. and Jones, R. W. L. "GANGA: a user-Grid interface for Atlas and LHCb," CoRR (cs.SE/0306085), 2003.

[GFD.47] Mandrichenko, I., Allcock, W. and T.Perelmutov "GridFTP v2 Protocol Description", http://www.ogf.org/documents/GFD.47.pdf, 2005.

[gLite2006] Burke, S., Campana, S., Peris, A. D., Donno, F., Lorenzo, P. M., Santinelli, R. and Sciaba, A. "gLite 3.0 users guide", https://edms.cern.ch/file/722398/gLite-3-UserGuide.html Enabling Grids for E-sciencE project, 2006.

[gLite2008] "gLite home page - www.glite.org", 2008.

[GridCafe2008] "GridCafé website – http://gridcafe.web.cern.ch", 2008.

[Grimshaw1994] Grimshaw, A. S., Wulf, W. A., French, J. C., Weaver, A. C. and Reynolds, P. F. Jr. "Legion: The Next Logical Step Toward a Nationwide Virtual Computer (CS-94-21)", Technical report, University of Virginia, 1994.

[Haslwanter2005] Haslwanter, T., Buchberger, M., Kaltofen, T., Hoerantner, R. and Priglinger, S. "SEE++ - A Biomechanical Model of the Oculomotor Plant", *Annals of the New York Academy of Sciences* (1039 (1)), 2005, pp. 9--14.

[Hedges2007] Hedges, M., Hasan, A. and Blanke, T. "Curation and Preservation of Research Data in an iRODS Data Grid", 'E-SCIENCE '07: Proceedings of the Third IEEE International Conference on e-Science and Grid Computing', IEEE Computer Society, Washington, DC, USA, 2007, pp. 457--464.

[hibernate] "Hibernate homepage - http://www.hibernate.org/ ", 2008.

[iRods2008] "iRods - www.irods.org", 2008.

[jvmti] "JavaTM Virtual Machine Tool Interface (JVM TI) - http://java.sun.com/javase/6/docs/technotes/guides/jvmti/index.html", 2008.

[Kaltofen2002] Kaltofen, T. "Design and Implementation of a Mathematical Pulley Model for Biomechanical Eye Surgery", Diploma Thesis, Fachhochschul OÖ, Hagenberg, Austria, 2002.

[Kesselman1998] Kesselman, C. and Foster, I. *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers, 1998.

[Koblitz2008] Koblitz, B., Santos, N. and Pose, V. "The AMGA Metadata Service", *Journal of Grid Computing* (Volume 6:Number 1), 2008, pp. 61--76.

[Langella2004] Langella, S., Hastings, S., Oster, S., Kurc, T., Catalyurek, U. and Saltz, J. "A distributed data management middleware for data-driven application systems", 'CLUSTER '04: Proceedings of the 2004 IEEE International Conference on Cluster Computing', IEEE Computer Society, Washington, DC, USA, 2004, pp. 267--276.

[LenzReichert2007] Lenz, R. and Reichert, M. "IT support for healthcare processes - premises, challenges, perspectives", *Data & Knowledge Engineering* (61:1), 2007, pp. 39--58.

[LHC2008] "LHC computing grid - www.cern.ch/lcg", 2008.

[LHCb2008] "LHCb public website, http://lhcb-public.web.cern.ch/lhcb-public", 2008.

[Miller1999] Miller, J. M. "Orbit™ 1.8 Gaze Mechanics Simulation", Eidactics, Suite 4041450 Greenwich Street, San Francisco CA 94109-1466, 1999.

[Miller1999a] Miller, J. M. and Demer, J. L. "Clinical Applications of Computer Models for Strabismus", 'Clinical Strabismus Management', W. B. Saunders, 1999.

[Mitterdorfer2005] Mitterdorfer, D. "Grid Capable Persistence Based on a Metamodel for Medical Decision Support", Diploma Work, Fachhochschul OÖ, Hagenberg, Austria, 2005.

[Mobius2008] "The Mobius Project – http://projectmobius.osu.edu", 2008.

[MySQL2008] "MySQL 5.0 Reference Manual", MySQL AB, http://dev.mysql.com/doc/refman/5.0/en/, 2008.

[Nair2007] Nair, A. S. "Computational Biology & Bioinformatics: A Gentle Overview", Technical report, Computer Society of India, Communications of the Computer Society of India, 2007.

[NHS2005] "NHS Healthcare Modelling Programme - http://www.standards.nhsia.nhs.uk/hcm/index.htm", 2005.

[OASIS_WS-I2008] "Web Services Resource Framework - www.oasis-open.org/committees/tc_home.phpwg_abbrev=wsrf", 2008

[OGSA-DAI-DevGuide2008] "GT 4.2.0 OGSA-DAI: Developer's Guide", Database Access & Integration Services Workgroup, University of Edinburgh, www.globus.org/toolkit/docs/4.0/techpreview/ogsadai/developer-index.html 2008.

[OGSA-DAI2008] "The OGSA-DAI Project - www.ogsadai.org.uk", 2008.

[OGSA-DQP2008] "The OGSA Distributed Query Processor (OGSA-DQP) - www.ogsadai.org.uk/about/ogsa-dqp/", 2008.

[Rajasekar2004] Rajasekar, A., Wan, M., Moore, R. and Schroeder, W. "Data Grid Federation", In proceedings of 'The International Conference on Parallel and Distributed Processing Techniques and Applications - PDPTA', 2004, pp. 541--546.

[RFT2008] "GT 4.0 Reliable File Transfer (RFT) Service documentation - www.globus.org/toolkit/docs/4.0/data/rft/", 2008.

[Santos2005] Santos, N. and Koblitz, B. "Metadata services on the grid", 'Advanced Computing and Analysis Techniques (ACAT'05)', 2005.

[Santos2006] Santos, N. and Koblitz, B. "Distributed Metadata with the AMGA Metadata Catalog", *CoRR* (abs/cs/0604071), 2006.

[SEE-KID2008] "SEE-KID homepage - www.see-kid.at", 2008.

[Sinnott2005] Sinnott, R.O.;Houghton, D. "Comparison of Data Access and Integration Technologies in the Life Science Domain", 'Proceedings of UK e-Science All Hands Meeting, September 2005, Nottingham, England.', 2005.

[Sotomayor2005] Sotomayor, B. and Childers, L. *Globus Toolkit 4: Programming Java Services*, Morgan Kaufmann, 2005

[spring] "Spring Framework - http://www.springframework.org", 2008.

[SRB2008] "Storage Resource Broker (2008) - www.sdsc.edu/srb", 2008.

[stunnel] "Stunnel website – Universal SSL Wrapper. http://www.stunnel.org/", 2008.

[tptp] "TPTP - Tracing and Profiling ProjecT - http://www.eclipse.org/tptp/", 2008.

[Tuecke2003] Tuecke, S., Czajkowski, K., Foster, I., Frey, J., Graham, S., Kesselman, C., Maguire, T., Sandholm, T., Snelling, D. and Vanderbilt, P. "Open Grid Services Infrastructure", http://www.ogf.org/documents/GFD.15.pdf, 2003.

[Woess2006] Wöß, W., Schreiner, W. and Buchberger, M. "Status Update on the Integration of SEE-GRID Into G-SDAM and Further Implementation Specific Topics", Technical report, RISC and Upper Austrian Research (UAR), Technical Report, Austrian Grid, 2006.

[WSII2008] "WebSphere Information Integrator (WSII) - http://ibm.com/software/data/integration", 2008.

[WSRF2008] "Web Services Resource Framework - www.oasis-open.org/committees/tc_home.phpwg_abbrev=wsrf", 2008.

Curriculum Vitae

Personal Information:

Last Name: Matkó

First Name: Imre-Zoltán

Citizenship: Romanian

Place of birth: Satu Mare, Romania

Date of birth: November 15, 1984.

Contact Information:

Telephone: +40(0)721765475; +40(0)261746040

E-mail: imre.matko@gmail.com

Home address: str. Astronauţilor bl. A4, ap. 10, Satu Mare, jud. Satu Mare, Romania, 440181

Education:

2007 – 2008 Master's studies at ISI-Hagenberg (International School for Informatics), Specialization Informatics: Engineering & Management, Johannes Kepler University and University of Applied Sciences, Hagenberg im Mühlkreis, Upper Austria, Austria. Master's Thesis work: Grid-aware Database Support for Medical Software.

2003 - 2007 Bachelor of Science (4 year studies) at The Faculty of Mathematics and Computer Science of "Babeş-Bolyai" University of Cluj-Napoca, Romania, Computer Science profile, Computer Science specialization, with Diploma work: Routing of Broadcast Messages in Ad-hoc Wireless Networks.

During my BSc study I was fascinated the most by topics related to operating systems, computer networks, concurrent and distributed programming in Java.

In my master's studies I was focusing on different aspects of Grid and high performance computing, with special consideration to Grid web-services and Grid-database services.

Spoken Languages:

Hungarian (mother tongue)

Romanian (advanced)

English (fair)

Skills and Competencies:

- Programming skills:
    - Java, Distributed Programming in Java, web technologies
    - Web-services, Java services for the Globus Toolkit Grid middleware.

- Basic education on object oriented programming.
- Basic Knowledge of Visual Studio .Net, C# and C++ programming.
- SQL data base programming (MS-SQL, MySQL)
- TYPO3 CMS – web development (basics)

- Software Tools:
  - Eclipse, IntelliJ IDEA, Microsoft Visual Studio
  - Microsoft Office, Open Office
  - Gimp (basics)
- Some experience in commerce

Academic Experience :

Master's Thesis work carried out at the Medical Informatics Research Unit of RISC Software GmbH, Hagenberg, Austria. In the frame of the SEE-GRID project, we dealt with the performance evaluation of the database system of the SEE++ virtual eye-muscle surgery application and the development of a Grid interface for this database component.

Fellowship from the "Farkas Gyula Association for Mathematics and Informatics", for the research work: The Application of Binomial Trees in Ad-hoc Wireless Networks, academic year 2006-2007.

3rd prize on the 10th Transylvanian Students' Scientific Conference, Computer Science Section, with the talk: Broadcast in the Ad-hoc Wireless Networks, Cluj-Napoca, Romania, 2007 may 26-27.

Fellowship for scientific research and support for participation at scientific conferences, from the foundation "Iskola Alapítvány" (Foundation for School), 2007.

I've participated in the one semester long "Uptime Project", in the frame of a university course, where the team managed to develop from scratch a web services and uptime monitoring system with a web based user interface and MySQL powered and Java salted back-end for different monitoring and checking functionalities.

Abilities :

Ability to work in team, dynamism, be productive. Be punctual, flexible, respect deadlines.

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Masterarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

.....

Hagenberg, im Juli 2008 Imre Zoltán Matkó