# End-to-End QoS Support for a Medical Grid Service Infrastructure

Siegfried BENKNER, Gerhard ENGELBRECHT

*Institute of Scientific Computing, University of Vienna, Nordbergstrasse 15/C/3, 1090 Vienna, AUSTRIA.*
sigi@par.univie.ac.at

Stuart E. MIDDLETON

*IT Innovation Centre, University of Southampton, 2 Venture Road, Chilworth Science Park, Southampton, SO16 7NP, UK.*
sem@it-innovation.soton.ac.uk

Ivona BRANDIC, Rainer SCHMIDT

*Institute of Scientific Computing, University of Vienna, Nordbergstrasse 15/C/3, 1090 Vienna, AUSTRIA*

***Abstract*** Quality of Service support is an important prerequisite for the adoption of Grid technologies for medical applications. The GEMSS Grid infrastructure addressed this issue by offering end-to-end QoS in the form of explicit timeliness guarantees for compute-intensive medical simulation services. Within GEMSS, parallel applications installed on clusters or other HPC hardware may be exposed as QoS-aware Grid services for which clients may dynamically negotiate QoS constraints with respect to response time and price using Service Level Agreements. The GEMSS infrastructure and middleware is based on standard Web services technology and relies on a reservation based approach to QoS coupled with application specific performance models. In this paper we present an overview of the GEMSS infrastructure, describe the available QoS and security mechanisms, and demonstrate the effectiveness of our methods with a Grid-enabled medical imaging service.

**Keywords** Grid middleware, Web services, Quality of service, Medical simulation services.

## §1 Introduction

The GEMSS Project [17] has developed a secure, service-oriented Grid infrastructure for the provision of advanced medical simulation applications as Grid services [4, 22]. The medical prototype applications considered in GEMSS include maxillo-facial surgery simulation [23], neuro-surgery support [36], radio-surgery planning [16], inhaled drug-delivery simulation [24], cardio-vascular simulation [25] and advanced image reconstruction [3]. At the core of these bio-medical simulation applications are computationally demanding methods such as parallel Finite Element Modeling, parallel Computational Fluid Dynamics and parallel Monte Carlo simulation, which are realized as remote Grid services running on clusters or other parallel computing platforms. To enable the use of Grid services in a clinical environment, predictability of response times is of utmost importance. Addressing this issue, we have developed a flexible end-to-end Quality of Service (QoS) infrastructure for providing explicit response time guarantees for simulation services which are executed remotely on a Grid host. Response time guarantees are usually negotiated dynamically between a client and potential service providers on a case-by-case basis.

The GEMSS project went beyond a model of cost free sharing of Grid resources, to a business oriented Grid model, where services are offered by service providers within an economic context. Therefore, GEMSS also addresses the realization of Grid business models, and services may be configured to support dynamic price negotiation as well. QoS guarantees agreed between a service consumer and a service provider are expressed in form of a Web Service Level Agreement (WSLA) [40]. Besides explicitly negotiable QoS guarantees, the GEMSS infrastructure provides implicit QoS by realizing highest security levels and by supporting error recovery.

The GEMSS Grid infrastructure and middleware has been built on top of standard Web services technologies [35, 39, 41] ensuring future extensibility and interoperability. Furthermore, GEMSS addresses privacy, security and other legal concerns by examining and incorporating into its Grid services the latest laws and EU regulations related to providing medical services over the Internet [29].

In this paper we present an overview of GEMSS, outline the provision of parallel simulation codes running on clusters as Grid services, and describe the QoS support infrastructure in more detail. Section 2 presents an overview of the GEMSS Grid infrastructure and discusses the provision of applications as Grid services. Section 3 describes the GEMSS QoS infrastructure and the basic strategy for QoS negotiation. Section 4 discusses security and legal issues pertaining to the Grid provision of medical simulation services. Section 5 presents a case study of a medical image reconstruction service. Finally, a discussion of related work and conclusions are presented in Sections 6 and 7, respectively.

## §2 GEMSS Grid Infrastructure

The GEMSS infrastructure is based on a service-oriented architecture comprising multiple clients and services, one or more service registries, and a certificate authority. Service registries maintain a list of service providers and the services they support. The certificate authority provides the basis for an operational PKI infrastructure based on X.509 certificates for establishing an identity for clients and service providers as well as for realizing transport and message layer security. Grid Clients are usually Internet-enabled PCs or workstations with GEMSS client software installed that permits communication with a service provider through the GEMSS middleware. The client side applications handle the creation of service input data and visualization of service output data. GEMSS services encapsulate native HPC applications (usually parallel simulation kernels written in Fortran or C and MPI) and provide support for QoS negotiation, data staging, job execution, job monitoring, and error recovery. GEMSS services are defined via WSDL and securely accessed using SOAP messages. For large file transfers SOAP attachments are utilized.

### 2.1 Service Access Model

GEMSS supports a three step process to job execution. First there is an initial business step, where accounts are opened and payment details fixed. Next there is a quality of service negotiation step, where a job's quality of service and price, if not subject to a fixed price model, is negotiated and agreed. Finally, once a QoS contract is in place, the job itself can be submitted and executed. Since GEMSS supports a client driven approach for accessing services, it is not required that holes be tunneled through site firewalls. End-to-end security is realized on top of transport layer security (HTTPS, SSL) and message layer security utilizing WS-Security standards [41].

### 2.2 Service Provision Infrastructure

Figure 1 depicts the main architectural components of the GEMSS service provision infrastructure. Medical simulation applications are exposed as

**Fig. 1** GEMSS service provider infrastructure.

Web Services and hosted using a Web server and a service container (Apache and Tomcat/Axis). The quality of service management component handles reservation with the resource manager and provides input to the quality of service negotiation process. The error recovery component handles check pointing and re-starting of applications if required. The logger manages a database for logging auditable information and a low level system log for event logging. The session management component manages a state database that contains information about any client-service interaction allowing it to be resumed at a later time if the user logs off.

The provision of medical simulation applications as Grid services is based on the concept of generic application services [5]. A generic application service comprises configurable software components with common methods for data staging, remote job management, error recovery, and QoS support, which are to be supported by all GEMSS services. In order to customize the behavior of these methods for a specific application, an XML application descriptor has to be provided. The application descriptor usually specifies the input/output files, the script for initiating job execution, and, as described in Section 3, a set of QoS parameters and corresponding QoS models.

In order to provide support for automatic service deployment, a corresponding deployment tool has been developed. The deployment tool enables the user to enter the information required in an application descriptor via a GUI and to control the deployment process. Internally, the deployment tool creates the XML application descriptor, generates an appropriately customized

Web service which encapsulates the application, publishes the corresponding WSDL document in a registry service, and finally deploys the service within the GEMSS hosting environment which is based on Apache Tomcat/Axis [1, 38)]. Usually during this process no application code changes are required, provided the application can already be executed in batch mode and that files in I/O operations are not accessed with absolute path names.

## 2.3 Client Infrastructure

The main architectural components of the GEMSS client infrastructure are shown in Figure 2.

**Fig. 2** GEMSS client infrastructure

The client-side application code usually relies on the GEMSS client application programming interface (API) which hides most of the complexity of dealing with remote services from the client-side application developer by providing appropriate service proxies. The high-level client API provides several different methods for discovering and selecting services. All these methods return service proxies, which can then be used to interact with services without having to deal with the complexities of client-side stub generation. Service proxies are in charge of discovering services and negotiate with Grid services to run jobs. The session management component manages client sessions, and a security context is maintained allowing authentication of the current user and providing the access criteria for the certificate and key stores. A service discovery component is provided for looking up suitable services in a service registry. Marshalling and security is handled transparently to the client application at the lowest two layers of the API and completely hidden from the user. This prevents the user from dealing with low-level issues such as message generation, signing, and encryption. However, each client is required to obtain a valid certificate from the GEMSS CA in order to access GEMSS services.

The client typically runs a business workflow to open negotiations with a set of service providers for a particular application. The quality of service negotiation is then run to request bids from all interested service providers who can run the clients jobs subject to QoS criteria required by the client; this results in a contract being agreed with a single service provider. The client then uploads the job input data to the service provider and starts the server side application by calling appropriate methods of the service.

The client infrastructure is centered on a pluggable client side component framework which provides support for dynamic configuration and replacement of client-side components.

## §3 QOS Support

Using the GEMSS infrastructure, medical simulation applications available on clusters or other parallel hardware may be exposed as QoS-aware services which are capable of negotiating with clients QoS guarantees on execution time, price and others. The GEMSS QoS negotiation mechanisms enable a client to negotiate with one or more service providers the required end-time at which the results of a time-critical simulation job have to be ready. Service providers utilize machine-specific application performance models in order to estimate the required execution time for a specific job based on input meta-data supplied by the client during QoS negotiation. In order to ensure the availability of appropriate computing resources for a service request, the QoS infrastructure relies on a scheduling system that provides support for advance reservation [34].

### 3.1 QoS Infrastructure

Figure 3 presents the main parts of the GEMSS QoS infrastructure separated into client-side and service-side parts. The service-side QoS management module is centered around the QoS manager, which interacts with the resource manager, the application performance model and the business model. The QoS infrastructure relies on four different XML schemata for the specification of QoS descriptors, request descriptors, performance descriptors and machine descriptors. The QoS event database is utilized by the QoS manager and the QoS

**Fig. 3** GEMSS QoS infrastructure.

monitoring service to store and query specific QoS events. A client-side QoS negotiation component is provided for the development of client applications that interact with remote Grid services. It offers the QoS Proxy interface with methods for requesting, confirming and for canceling QoS contracts.

## 3.2 QoS Management Module

The QoS manager is the central server-side module of the QoS support infrastructure and provides the interface `QoS` with basic QoS negotiation operations. The QoS manager receives a QoS request from a client, checks whether the client's QoS constraints can be met, and generates a corresponding QoS offer which is returned to the client.

The QoS manager utilizes the application performance model to estimate the performance requirements (runtime, memory and disc requirements). The performance model takes as input a request descriptor and a machine descriptor and returns a performance descriptor. A request descriptor contains meta-data about a specific service request (e.g. mesh size, image resolution, etc.) supplied by the client during QoS negotiation. A machine descriptor contains machine specific information, usually specifying the number of processors, or a range of feasible processor numbers that should be used for executing an application. A performance descriptor comprises information on job capacity estimations including runtime, disc requirements and memory requirements.

Since in general, it will not be possible to build an analytical model which allows for precise predictions of memory and computing time requirements for all applications, GEMSS does not prescribe the nature of a performance model. For applications where an analytical performance model is not feasible, for example, a data base may be used, relating typical problem parameters to resource needs like main memory, disk space and runtime.

The resource manager module realizes a high level interface to an underlying scheduling system which has to provide support for advance reservation, The GEMSS infrastructure currently supports the use of two schedulers that offer advanced reservation, NEC's COSY scheduler [11)] and the Maui scheduler [27)]. The resource manager provides methods for requesting and for confirming temporary reservations, for canceling reservations, for job submission and for inquiring information about a submitted job.

The service provider's business model defines a generic mechanism to calculate a price based on estimated resource allocation. A concrete business model implementation has to be provided by the service provider. Within GEMSS two different pricing models have been realized, a fixed price telephone pricing model where users are charged at a prearranged CPU hour rate, and a dynamic pricing model where the CPU hour rate is dependent on the current load levels the service provider is experiencing.

It should be noted that advance reservation may be in conflict with a service provider's desire to maximize the utilization of its resources. In GEMSS, however, the focus has been on the realization of mechanisms to ensure a client's QoS requirements with respect to response time and price. As a consequence, mechanisms for balancing the tradeoff between advance reservation and resource utilization have not been addressed within the GEMSS project.

## 3.3 Basic QoS Negotiation

The basic QoS negotiation as shown in Figure 4 is based on a request-offer model where a client requests offers from service providers. If the client agrees to an offer, it is signed by both parties resulting in a QoS contract.

In an initial task the client may access a GEMSS registry service to obtain a list of candidate services. The client then invokes for each candidate service the operation `requestQosOffer`, passing along a request descriptor with input meta data and a QoS request document with the required QoS constraints. On the service side, the QoS manager executes the performance model with the request descriptor as input and compares the estimated execution time in the resulting performance descriptor with the time constraints specified in the QoS

**Fig. 4** Basic QoS Negotiation Scenario

request. If the client's execution time constraints can be met, the QoS manager instructs the resource manager to check whether the required resources can be made available by invoking the operation `getResourceDsc` passing along the performance descriptor and the QoS request document. The resource manager contacts the scheduler to check whether the required resources as specified in the performance descriptor (number of processors for the estimated runtime) can be made available within the time frame (begin time, end time) specified in the client's QoS request. If this is possible, a temporary reservation is made with the scheduler and a corresponding resource descriptor is returned. The QoS manager then executes the business model, passing as argument the resource descriptor, to determine if the price for the required resources is within the client's price constraints. If the price constraints can be met, the QoS manager generates a corresponding QoS offer, and returns it to the client. If the time or price constraints cannot be met, the QoS manager may repeatedly execute the performance model with a different number of processors. If the client's QoS constraints cannot be met at all, no offer is generated.

On the client side, the QoS offers from different service providers are received and analyzed. The client confirms the best offer, or, if it is not satisfied with the offered QoS constraints, may set up a new QoS request with different constraints and start a new negotiation. If the client confirms an offer, the QoS manager confirms the temporary resource reservation made for the offer, signs the QoS contract and returns it to the client.

Within the GEMSS project also more sophisticated negotiation strategies based on auction models have been realized, a description of which is beyond the scope of this paper.

### 3.4 QoS Descriptors - WSLAs

A QoS descriptor, as used in GEMSS during QoS negotiation, is an XML-based document representing a (potential) agreement on a single service usage between a service consumer (client) and a service provider, following the Web Service Level Agreement (WSLA) specification [40)] developed by IBM. Depending on the state of a QoS negotiation, a QoS descriptor either is a QoS request, a QoS offer, or a QoS contract.

QoS descriptors consist of three main parts: parties, service definition, and obligations. The parties section comprises information about the signatory parties involved, which is usually extracted from the GEMSS certificates of users and service providers, respectively. The service definition section contains the actual subject matter of the agreement by defining all operations subject to the agreement and a set of SLA parameters, usually comprising the begin time, end time, and price of a job execution. Furthermore, the service definition section specifies the overall contract duration and a metric for each parameter. The obligations section contains a list of objectives. Each objective is linked to an obliged party, has an according validity and defines an expression that is associated with a defined SLA parameter. For example, the SLA parameter price has to be equal to 5 EUR or the end time of the job execution must not exceed 19 May 2007, 11:00 CET.

## §4 Security and Legal Aspects

The use of Grid technology in the life sciences sector raises significant legal and security issues. The GEMSS project examined both the legal and security framework in which Grid technology may be exploited [29)]. Our legal study analyzed the pertinent European regulations from the viewpoint of privacy protection with regards to the processing of patient data by means of Grid services. This analysis allowed us to draw up the common legal framework under which GEMSS applications can be developed in Europe. Moreover, we developed technical and procedural security solutions for GEMSS, and proposed a methodology for assessing a specific site's security.

## 4.1 EU Privacy law regarding medical services

The EU Directive 95/46 applies to medical simulation services since, according to its article 3.1, it applies to wholly or partly automated processing of personal data [20]. According to this directive, a medical imaging service, for example, would contain a patient, some personal data (images that need reconstruction), a controller (eg. legal representative from a UK hospital), a processor (eg. legal representative from an Austrian service provider) and an electronic register (Grid service registry). Directive 95/46 makes it clear that any patient data, sent to the service provider via the Internet, is personal data since it is related to a well-identified natural person. If the data sent via the Internet is not directly nominative, but can via some code be attributable to an identified person, it is also personal data. The transmission via the Internet and subsequent processing by a service provider constitute sets of operations performed upon personal data by automated means.

When processing personal data on behalf of the controller, the GEMSS service provider acts as a processor. The controller must thus choose a processor who provides sufficient guarantees in respect of the technical and organizational measures governing the processing to be carried out. The controller must ensure compliance with those measures. All processors must therefore be governed by a contract or legal act binding the processor to the controller, and stipulating in particular that the processor shall act only on instruction from the controller.

## 4.2 Best practice security for a medical Grid

The GEMSS Grid infrastructure is capable of providing a high degree of security for the processing of personal data. Our security mechanisms ensure the confidentiality and integrity of personal data, and that data processors are identified, authenticated and authorized. Our security solutions are based on a public key infrastructure, transport level security protocols, end to end message security standards and an authorization mechanism. Clients are not allowed shell access or to upload software, since the costs involved in securing this level of access to the service provider hardware are too severe. Along with an intrusion detection system these solutions provide GEMSS with security in depth.

A public key infrastructure (PKI) uses certificates to identify parties, employing asymmetric public key encryption and a trusted third party to control certificate issue and revocation. All GEMSS certificates are X.509 compliant [12)] and are used by people or machines for authentication, identification and authorization purposes on the GEMSS Grid. The certificate policy and the certification practice statement of the GEMSS certificate authority ensure that certificates issued to people are in line with directive 1999/93EC of the European parliament and the council on a community framework for electronic signatures [13)].

We have set-up intermediate demilitarized zones (DMZ's) at our service provider's sites, which forward relevant messages to more protected internal network domains. Demilitarized zones provide a buffer zone, so should a hacker gain access to computers with public IP addresses they would still need to discover and access computers with private internal IP addresses. Transport level security involves the basic security and encryption mechanisms involved with transmitting data over the Internet. We authenticate our communication paths using the HTTPS protocol, allowing our data to be transmitted confidentially. Within the GEMSS Grid both clients and service providers are protected by firewalls, and as such belong to different trust domains.

End-to-end security protocols apply a security policy to ensure that the message originator is authenticated, that the message itself has not been tampered with and for mutual authentication. Our end-to-end security mechanisms are based on the Web Service Security specifications [41)]. In GEMSS we use certified identities to determine the right of access to selected services and resources. The access rights associated with a certified identity are assigned according to the applied business process, and enforced through a service-level dynamic access control module. The GEMSS end-to-end security mechanisms are established between a client and a service. It is thus up to the service provider, to ensure appropriate security standards for accessing its local compute resources. As mentioned before, the compute resources used by a service provider to fulfill a service request are not visible to a client. Delegation of security credentials between different GEMSS service is not supported by GEMSS. For a more detailed description of the GEMSS security mechanisms the reader is referred to [29)].

## §5 Case Study - Medical Image Reconstruction

This section presents experimental results for a set of infrastructure tests performed with medical image reconstruction services for single photon emission computer tomography (SPECT). The parallel reconstruction kernel utilized within the SPECT service is based on a compute-intensive fully 3D ML-EM reconstruction algorithm [3], which has been implemented in C/MPI.

| | Small | Medium | Big |
|---|---|---|---|
| Resolution | 128 | 128 - 256 | 256 |
| Projections | 60 | 60 - 120 | 120 |
| Slices | 8 - 32 | 64 - 128 | 8 - 32 |
| Iterations | 5 - 25 | 5 - 25 | 5 - 25 |

**Table 1** Job characteristics

For the evaluation we utilized 24 different SPECT jobs with varying job characteristics and input data, as listed in Table 1, which cover a range of use-cases an end-user wants to perform with the help of a medical Grid. A SPECT image reconstruction job can be characterized by its image resolution, the number of projections acquired from the CT or MRI scanner, the number of slices that should be computed for the output image volume, and the number of iterations to be performed. These parameters are the main input for the performance model and supplied by the client during QoS negotiation. Note that the QoS manager attempts to automatically determines the number of processors that should be used for a certain SPECT job in order to meet a client's time constraints by repeatedly executing the performance model with varying numbers of processors. Our test infrastructure comprised three 16 CPU clusters in different administrative domains for the deployment of the SPECT service and up to 10 different workstations and PCs for running SPECT clients. Table 2 shows the average measured runtime in minutes for the different classes of SPECT jobs on 2, 4, 8, and 16 processors, respectively.

In order to demonstrate the rational behavior of our QoS infrastructure we have devised a scenario with a single service provider, with a 16 CPU cluster, and clients submitting concurrently medium-sized SPECT jobs with the same characteristics. The runtime of these jobs is about 40 minutes on 2 processors, 21 minutes on 4 processors, and 12 minutes on 8 processors. All clients started

| Nodes# | Small | Medium | Big |
|---|---|---|---|
| 2 | 13 | 40 | 130 |
| 4 | 8 | 21 | 80 |
| 8 | 5 | 12 | 53 |
| 16 | 4 | 8 | 41 |

**Table 2** Average job runtimes in minutes

submitting jobs at time $t0$. As expected the QoS manager schedules the first eight jobs on 2 processors and starts them immediately. Since after 8 jobs the cluster is fully utilized, the QoS manager schedules the next four jobs on 4 processors, but with a later start time $t1 \geq t0 + 40\ minutes$. Finally, two more jobs are scheduled on 8 CPUs and an even later start time $t2 \geq t0 + 52\ minutes$. After this, no more offers were generated within this time slot.

For proving the evidence that the infrastructure works properly in a more complex setting, we conducted a stress test with three service providers and ten concurrently running clients. These clients randomly picked up a job set, defined their QoS constraints, queried a registry in order to determine the endpoints of our service providers, performed a sequential QoS negotiation with the service providers, agreed to the first matching offer, and initiated the transfer of input data and job execution. For this scenario we have performed three different test runs with clients requesting small (1), medium (2) and big (3) jobs as characterized in Table 1. Moreover, we have configured the clients to select their QoS constraints randomly from 3 different types of preferences: (a) request a fast job execution, (b) request a medium time job execution or (c) request just the execution with a long time constraint. This scenario reflects three different kinds of user groups: one with a high priority of a fast job execution, one group with medium time constraints, and one group with no hard time constraints. Using this strategy, we concurrently started ten clients for each test (1) to (3) for a longer time period.

Table 3 summarizes the main results of these tests. The requested job count indicates how many jobs have been requested by all clients. The rejected job count indicates how many jobs the system was not able to provide an offer for because the QoS manager was not able to fulfill a client's time constraints. The number of errors shows how many jobs failed due to technical reasons explained later. Robustness is defined as the number of completed jobs divided by the number of submitted jobs, and throughput as the number of successfully finished jobs per hour. Moreover, the average accuracy of the performance model for estimating the required runtime has been measured.

| | Small Jobs | Medium Jobs | Big Jobs |
|---|---|---|---|
| Testbed uptime (h) | 8,4 | 25,7 | 21,4 |
| Jobs requested | 401 | 401 | 401 |
| Jobs rejected | 0 | 1 | 319 |
| QoS contract | 401 | 400 | 82 |
| Errors | 5 | 3 | 4 |
| Jobs completed | 376 | 397 | 78 |
| Robustness | 98.75% | 99.25% | 95.53% |
| Throughput (Jobs/h) | 46.94 | 15.42 | 3.66 |
| Avg. Perf. Model Acc. | 97.28% | 97.62% | 97.57% |

**Table 3** Summary of experimental results.

We can see from these experiments that the system operates stable over a longer period of time and handles most of the requests properly (i.e. robustness is high). The few errors that occurred can be attributed to two different problems that arose during the tests. The first problem was related to network timeouts while querying the QoS event database (i.e. on status queries). This may happen when other threads (associated with a different client) concurrently write to this database. As writing threads have priority, reading processes may time out too soon. A potential solution to this would be a fine tuning of the timeout settings. Another problem occurred on the client side if an interaction with the service could not be performed within the validity time frame of the security token. A solution to this problem would be to extend the validity of the security tokens of our end-to-end security implementation. Despite these few problems, our system shows satisfying behavior. The system was also able to cope with a number of errors. General network timeouts have been handled transparently by retries and failed negotiations have been balanced by other service providers.

As can also be seen from Table 3, the throughput tends to be higher when submitting a lot of small jobs, which also indicates a rational behavior, because smaller jobs are executed faster than bigger ones. The varying average utilization correlates to the number of requested jobs and the number of actually started jobs. In test 1, where all requested jobs could be handled by the system, the total utilization is smaller than in test 3, where a lot of jobs have been rejected by the system because of capacity constraints. Finally, our results show that the accuracy of the SPECT performance model is very high and the system has not failed due to inaccurate performance predictions.

## §6 Related Work

Standard Web Service technologies have now been adopted as the base middleware technology by many Grid computing environments including Globus GT4 [18)], gLite [15)], OMII [31)] and Unicore [37)]. The Open Grid Service Architecture (OGSA [14)]) outlines the vision for a service level management and attainment model based on a generic control loop pattern, however, this vision has not yet been realized in available Grid environments. The OGSA specification mainly discusses system-level QoS assurance (i.e. of the overall Grid infrastructure) while our work presented here focuses on end-to-end QoS support at the application level for individual Grid services.

The work presented in [28)] deals with a QoS based Web services architecture. The system consists of QoS-aware components which can be invoked using a QoS negotiation protocol. As opposed to our work, this system does not deal with the Grid-provision of HPC applications. Several projects have proposed economy-based Grid systems [10, 19)]. Buyya proposed a Grid Architecture for Computational Economy (GRACE) providing a generic way to map an economic model into distributed system architecture and the Grid resources broker (Nimrod-G) supporting deadline and budget based scheduling of Grid resources. The GRIA [33)] QoS infrastructure utilizes a performance estimation service which relies on a workload estimation model to predict the execution time of a job using application specific parameters, and on a capacity estimation model to estimate the execution time of a submitted job using resource specific parameters.

There are a number of other Grid projects related to life sciences that deal with bio-medical applications, including the EU BioGrid Project [6)], the OpenMolGRID Project [32)], EU MammoGrid Project [26)], the UK e-Science myGrid Project [30)] and the US BIRN initiative [9)], too name a few. While most of these projects focus on data management aspects, the GEMSS project focuses on the computational aspect of the Grid, with the aim to provide hardware resources and HPC service across wide area networks in order to overcome time or space limitations of single HPC systems. Other projects in the bio-medical field which also focus more on the computational aspect of the Grid include the Swiss BiOpera Project [7], the Japanese BioGrid Project [6], and the Singapore BioMed Grid [8]. However, none of these Projects addresses the issues of application-level end-to-end QoS support.

## §7 Conclusions

In this paper we presented a generic QoS support Grid infrastructure targeting the on-demand provision of medical simulation applications that has been realized in the context of the EU Project GEMSS. The QoS infrastructure relies on a reservation based approach to QoS coupled with application specific performance models, advance reservation mechanisms, and client-driven negotiation of service level agreements. The GEMSS Grid provides guarantees to clients regarding quality of service, and the legal and security framework needed to provide a platform for future exploitation. The GEMSS project has demonstrated the potential of Grid technology to provide medical practitioners and researchers with access to advanced simulation and image processing services for improved pre-operative planning and near real-time surgical support. However, more work with respect to security, trust and business models will be required until Grid technologies can be utilized within day-to-day clinical practice.

The Grid technology developed in the course of the GEMSS project is being utilized and further developed within the EU Project Aneurist [2], which aims to create an IT infrastructure for the management of all processes linked to research, diagnosis and treatment development for complex and multi-factorial diseases. Although the GEMSS infrastructure has been developed with a focus on medical applications, it is applicable to other application domains in science and engineering where compute intensive applications should be provided as services that are accessible on-demand by clients over the Internet.

## ***References***

1) Apache Tomcat. http://jakarta.apache.org/tomcat/
2) Aneurist - Integrated Biomedical Informatics for the Management of Cerebral Aneurysms. EU IST Integrated Project IST-2004-027703. http://www.aneurist.org/
3) W. Backfrieder, M. Forster, S. Benkner, G. Engelbrecht. *Locally Variant VOR in Fully 3D SPECT within A Service Oriented Environment.* Proceedings of the International Conference on Mathematics and Engineering Techniques in Medicine and Biological Sciences, CSREA Press, p. 216-221, Las Vegas, USA, June 2003.

4) S. Benkner, G. Berti, G. Engelbrecht, J. Fingberg, G. Kohring, S.E. Middleton, R. Schmidt. GEMSS: Grid Infrastructure for Medical Service Provision. Journal of Methods of Information in Medicine, Vol. 44, 2005.

5) S. Benkner, I. Brandic, G. Engelbrecht, R. Schmidt. *VGE - A Service-Oriented Grid Environment for On-Demand Supercomputing.* Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (Grid 2004), Pittsburgh, PA, USA, November 2004.

6) The BioGrid Project. http://www.bio-grid.net/index.jsp

7) BiOpera - Process Support for BioInformatics. ETH Zrich, Department of Computer Science. http://www.inf.ethz.ch/personal/bausch/bioopera/main.html

8) BiomedGrid Consortium. http://binfo.ym.edu.tw/grid/index.html

9) The Biomedical Informatics Research Network. http://www.nbirn.net

10) R. Buyya. *"Economic-based Distributed Resource Management and Scheduling for Grid Computing"*, Ph.D Thesis, Monash University, Melbourne, Australia, 2002.

11) J. Cao, F. Zimmermann. *"Queue Scheduling and Advance Reservations with COSY"*, Proceedings of the International Parallel and Distributed Processing Symposium, Santa Fe, New Mexico, 2004

12) S. Chokhani, W. Ford, R. Sabett, C. Merrill, S. Wu. *Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework*, http://www.ietf.org/rfc/rfc3647.txt, The Internet Society, 2003.

13) European Parliament. Directive 1999/93 on a Community Framework for Electronic Signatures. Official Journal of the European Communities, 19/01/2000 : L013, 0012-0020.

14) I. Foster et al. The Open Grid Services Architecture, Version 1.5., Open Grid Forum, GFD-I.080, July 2006. http://forge.gridforum.org/projects/ogsa-wg

15) F. Gagliardi, B. Jones, and E. Laure. The EU DataGrid Project: Building and Operating a large scale Grid Infrastructure. In B. Di Martino, J. Dongarra, A. Hoisie, L.Y. Yang, and H. Zima, editors, Engineering the Grid: Status and Perspective. American Scientific Publishers, January 2006.

16) A. Gill, M. Surridge, G. Scielzo, R. Felici, M. Modesti, G. Sardu. RAPT: A Parallel Radiotherapy Treatment Planning Code. In: Liddell H, Colbrook A, Hertzberger B, Sloot P, editors. High Performance Computing and Networking Europe, Lecture Notes in Computer Science: Springer; p. 183-193, 1996.

17) The GEMSS Project: Grid-Enabled Medical Simulation Services, EU IST Project, IST-2001-37153, http://www.gemss.de/

18) The GLOBUS Toolkit, http://www.globus.org/

19) The GRASP Project. http://eu-grasp.net/

20) J.A.M Herveg, Y. Poullet. Directive 95/46 and the use of GRID technologies in the healthcare sector: selected legal issues. Proceedings HealthGrid 2003, pp. 229-236, Lyon, France, January 16-17, 2003.

21) The Japanese BioGrid Project. http://www.biogrid.jp/

22) D. M. Jones, J. W. Fenner, G. Berti, F. Kruggel, R. A. Mehrem, W. Backfrieder, R. Moore, A. Geltmeier. *"The GEMSS Grid: An evolving HPC Environment for Medical Applications"*, Proceedings HealthGrid 2004, Clermont-Ferrand, France, 2004.

23) Koch R.M., Roth S.H.M., Gross M.H., Zimmermann A.P., Sailer H.F. A framework for facial surgery simulation. In: Proceedings of the 18th spring conference on Computer graphics; ACM Press; p. 33-42, 2002.

24) S. Ley, D. Mayer, B. Brook, E. van Beek, C. Heusell, R. Hose, D. Rinck, H. Kauczor. Radiological imaging as the basis for a simulation software to advance individualised inhalation therapies. Eur Radiol 2001;11(Suppl):216-217, 2001.

25) Li JK-J. The Arterial Circulation: Physical Principles and Clinical Applications. Totowa, NJ: Humana Press; 2000.

26) The MammoGrid project. http://mammogrid.vitamib.com/

27) Maui Cluster Scheduler. http://www.clusterresources.com/products/maui/

28) D. A. Menascé. *"QoS-Aware Software Components"*, Internet Computing Online, Vol. 8, No. 2, pp.91-93, 2004.

29) S.E. Middleton, J. Herveg, F. Crazzolara, D. Marvin, Y. Poullet, GEMSS Security and Privacy for a Medical Grid, Methods of Information in Medicine, 2005.

30) The myGrid Project. http://mygrid.man.ac.uk/

31) The Open Middleware Infrastructure Institute. OMII 2.0 User Guide. http://www.omii.ac.uk/docs/2.3.3/omii_2_user_guide.htm

32) OpenMolGRID - Open Computing GRID for Molecular Science and Engineering. http://www.openmolgrid.org/

33) A. Panagakis, A. Litke, A. Doulamis, N. Doulamis, T. Varvarigou, E. Varvarigos. An Advanced Grid Architecture for a Commercial Grid Infrastructure. The 2nd European Across Grids Conference, Nicosia, Cyprus, Jan. 2004, Springer.

34) A. Roy, V. Sander. Advance Reservation API, GGF Scheduling Working Group, 2002. http://www.ggf.org/documents/GFD/GFD-E.5.pdf,

35) SOAP Version 1.2. http://www.w3.org/TR/soap/

36) Tittgemeyer M, Wollny G, Kruggel F. Visualising deformation fields computed by non-linear image registration. Computing and Visualization in Science 2002;5(1):45-51.

37) The UNICORE Forum. http://www.unicore.org

38) WebServices - Axis. http://ws.apache.org/axis/

39) Web Services Description Language (WSDL) 1.1, http://www.w3.org/TR/wsdl

40) Web Service Level Agreement (WSLA) Language Specification. http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf, IBM 2003.

41) Web Service Security. SOAP Message Security 1.0, OASIS Standard 200401, 2004.