GridIJ – A Dynamic Grid Service Architecture for Scientific Image Processing

Alexander Frank, Rainer Stotzka, Thomas Jejkal,
Volker Hartmann, Michael Sutter, Hartmut Gemmeke
Forschungszentrum Karlsruhe
Institute for Data Processing and Electronics
Hermann–von–Helmholtz–Platz 1, 76344 Eggenstein, Germany
alexander.frank@ipe.fzk.de, rainer.stotzka@ipe.fzk.de

Abstract

The presented dynamic Grid service architecture provides a novel and comfortable access for scientific software developers and users without prior knowledge of Grid technologies or even the underlying architecture. A simple Java API allows the extension of WSRF–compliant web services by scientific software components deployed automatically on available GT4 containers. In preparation only two services are started on each container allowing hot–deployment and simple performance analysis.

GridIJ is a reference implementation of the presented architecture with a problem solving environment for image processing. A plugin with a GUI extends the free software ImageJ, providing control of the architecture, deploying scientific software components (GridPlugins), distributing data to process in parallel, controlling the workflow and returning the results.

First tests showed the simple usage and extension by GridPlugins of the system. In principle the performance in parallel execution scales linearly with an additional small overhead for connecting and data transfer.

1 Introduction

Scientific data processing and analysis have permanently growing demands on computing resources. For example image reconstruction of ultrasound computer tomography data [24] at Forschungszentrum Karlsruhe needs several weeks of computing time on a single workstation. A similar problem occurs in digital forensics: To locate stolen goods, which are sold on the Internet, thousands of images have to be compared using complex object recognition algorithms [26]. The computational expenses of this application do not allow to test the algorithms on single workstations, but can be accelerated by several orders of magnitude if adequate computing resources are available.

The best potential to exhaustively fulfill the requirements of such projects in respect of scalability, standardization and stability is the use of oncoming Grid computing technologies. Grid will finally provide today unimaginable amounts of computing resources.

Nowadays a quickly growing diversity of Grid solutions is rapidly emerging, overwhelming scientific users and software developers. For an overview see [18]. To increase the acceptance of Grid technologies in the scientific communities simple and easy–to-use access methods to Grid technologies are needed.

Different middleware systems aim to provide a unified access to Grid resources, e.g. the Globus Toolkit 4 GT4 [8, 12], gLite [1] and Unicore [27], just to name some of the well–known ones. Despite of the different internal hierarchical doctrines and architectures most of these middleware systems integrate computing resources like computing clusters in the form of job submission machines. Users have to describe their tasks to provide executables appropriate to the target machines and submit them as jobs. Extensive knowledge of the underlying middleware and Grid infrastructure is still needed.

To simplify the access to different middleware systems and the development of Grid software several projects aim to support the users. The SAGA research group [21] of the Open Grid Forum collects the requirements for application driven APIs and draws first specifications. The Grid Application Toolkit [14] of the GridLab project hides the middleware systems from the users using an engine with a unified interface and adaptors to the different middleware systems. Nevertheless the Grid software developer needs knowledge of the underlying target machines.

A different approach is shown by NetSolve/GridSolve [4]. Specific services, e.g. the functionality of the LaPACK library [3], can be called from different problem solving environments like Matlab [17], Octave [7], Mathematica [28] and different programming languages. The programmers interfaces are function calls, which are easy

to understand and to handle. Nevertheless, the extension of the client–server system by additional scientific functions needs high expertise and access priorities.

A spreading software technology to access software components over networks are XML web services, which are also driven by commercial applications. A web service is invoked from a client using an Uniform Resource Identifier (URI). The client hands over the parameters for the method of the addressed service in SOAP [23] messages. Web services based on the Web Service Resource Framework (WSRF) [5] show a high potential to be accepted in scientific communities, since

- they easily scale in growing systems and are arbitrarily extendable,
- accessing them is independent from the underlying operating system, the implementing programming language and the middleware,
- they represent an open standard and therefore are future–proof.

The middleware GT4 [8] allows to build Grids with heterogeneous resources. GT4 achieves this by implementing the interfaces as web services, using the Open Grid Services Architecture [9], supported by security.

Besides to administrative functions, scientific tasks can also be provided by web services. The Grid Services Toolkit [19] of Forschungszentrum Karlsruhe integrates a library of services for process data processing, consisting of services for database access and management, statistical data analysis and certain project specific services.

Developing WSRF–compliant web services is still a challenging task. Tools like the Grid Development Tools (GDT) [11] and the Grid Access and Instrumentation Tool (GRAIL) [16] support the development and the debugging of services, but still require a profound knowledge of the underlying web service technologies.

On the other hand the deployment of newly created web services needs administrative rights on the servers which is not always favored. After deployment the web service container needs to be restarted which will influence all other active services.

At University of Marburg an Hot Deployment Service (HDS) [10] to deploy services on a running GT4 container without restart has been introduced. Using HDS gives the opportunity to create dynamic Grid architectures based on WSRF web services.

The aim of the proposed work is to provide a novel dynamic Grid service architecture which is used by scientists and extended by scientific software developers without prior knowledge of Grid technologies. Software components are developed on the client side allowing the developer to concentrate on the scientific functions. The components are deployed automatically on all required servers without interaction of administrators and without harming the server side environment.

In the following sections the architecture, its included components, a reference implementation for image processing and its usage is described. As a result the performance is briefly evaluated.

2 Architecture

In principal the use of web services is independent of the underlying platform and programming language. In the architecture we restrict ourself to the Java programming language, since the Globus Toolkit GT4 basically supports Java web services based on WSRF, including the hosting environment and certificate based security.

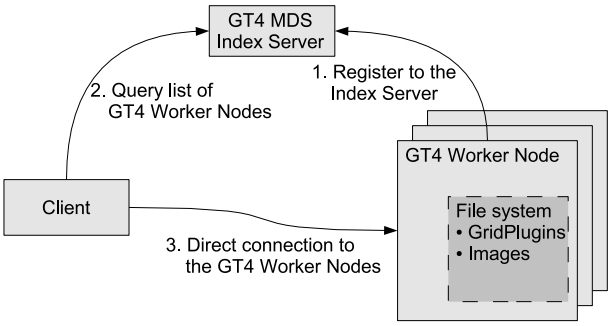

Figure 1. The architecture consists of three parts: The GT4 Worker Nodes (right), the GT4 MDS Index Server(top) and the Client (left).

The architecture consists of three fundamental parts (Fig. 1):

- Several **GT4 Worker Nodes** (Fig. 2) provide the computing power: Each GT4 Worker Node contains at least one GT4 Java web service container enabled with two additional services and is registered at the GT4 Monitoring and Discovery System (MDS) Index Server.
- The **GT4 MDS Index Server** is a computer running a GT4 Java web service container including the MDS [22]. It carries and distributes information about available GT4 Worker Nodes and deployed services.
- The **Client** computer executing the Client software is the interface of the user and/or scientific software developer to the architecture. The Java web service core libraries must be available for the Client software. Practically, the number of simultaneously accessing Clients is only restricted by the capacity of the servers.

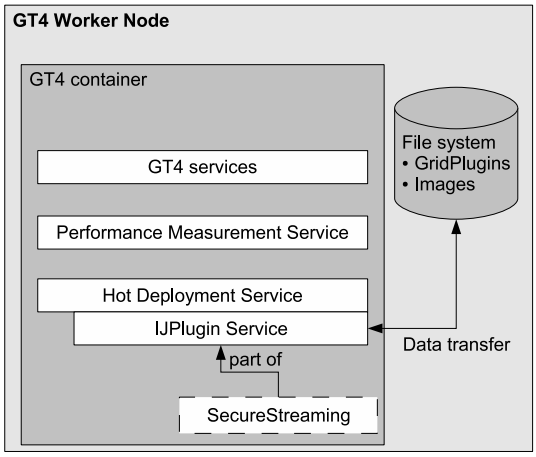

Figure 2. Internal structure of a GT4 Worker Node. Additional to the regular GT4 services the Performance Measurement Service and the Hot Deployment Service are deployed permanently. The IJPlugin Service (see section III) is deployed dynamically using the Hot Deployment Service.

2.1 The GT4 Worker Nodes

Each GT4 Worker Node runs a GT4 container providing the technological basis for the dynamic Grid service architecture. Among the standard GT4 services for execution management, data management, information management and security two additional services are installed:

- The **Performance Measurement Service** is used for very simple resource broking. It determines the actual load of the corresponding Worker Node. Information about the load of the CPUs during the last 5, 10 and 15 minutes, the maximum available and used memory of the Java virtual machine, as well as the number of CPUs and the architecture (x86 or x86_64), is provided. In future it is planned to be replaced by more sophisticated resource broking solutions provided by other research groups.
- The **Hot Deployment Service (HDS)** [10] of University of Marburg is mandatory to create a dynamic service architecture. It allows to start and control Grid services on a running GT4 container without restarting it.

2.2 The Client

The Client executes the Java Client software controlling the application and its distribution within the architecture. It is responsible for distributing the web services and the data to the GT4 Worker Nodes, the invocation of the methods of the services and the collection of the results after execution. Therefore it controls the whole distributed workflow.

Firstly the Client queries the GT4 MDS Index Server to get the URIs of available GT4 Worker Nodes. After accessing the Performance Measurement Service on each GT4 Worker Node a ranking of the currently best performing nodes is created. According to these informations the web services are distributed to the GT4 containers using the locally installed Hot Deployment Service. The HDS automatically deploys the selected services and returns the URIs to the Client (Fig. 1). After this step the connections between Client and GT4 Worker Nodes are established and the Client can start to upload additional functionalities and data, to execute the workflow by invoking the methods of the uploaded GridPlugins, and to download the results, when available.

To transfer greater amounts of data between the Client and the GT4 Worker Nodes the SecureStreaming Service, an in–house solution developed at Forschungszentrum Karlsruhe, is used. It is a very fast data transfer schema based on a producer–consumer pattern communicating via Java stream–sockets. In later developments the SecureStreaming Service might be replaced by other technologies, e.g. GridFTP [2, 13].

To demonstrate the potential and flexibility of this general architecture we developed GridIJ, which is introduced in detail in the following sections.

3 GridIJ

GridIJ is a reference implementation of the proposed architecture with a problem solving environment for scientific image processing. It aims to scientific users and programmers who want to accelerate and to extend the image processing functionality without prior knowledge of Grid computing and web services.

ImageJ [20] is a public domain Java image processing program with a huge variety of image processing tools. It runs as a stand–alone application, as an applet or can be used as a Java library. On one hand we use ImageJ on the Client computer and extended the graphical user interface by the Client software controlling the GridIJ architecture. On the other hand the library is called by the GT4 Worker Nodes to perform certain image processing tasks.

GridIJ is based on the architecture proposed in the section above and integrates it seamlessly into ImageJ. GridIJ implements three additional components:

- The **IJPlugin** Service wraps the API of ImageJ and manages data and GridPlugins, which are dynamically loaded into the IJPlugin Service;

- **GridPlugins** and the GridPlugin interface allow a comfortable way to extend the functionality of the IJ-Plugin Service;
- The **Client** software controls the GridIJ architecture and includes the graphical user interface.

3.1 IJPlugin Service

The IJPlugin Service represents the heart of the GridIJ architecture. Designed as a WSRF–compliant web service, it integrates:

- The management of GridPlugins, which are uploaded from the Clients and stored in the local file system as JAR files;
- The execution environment for GridPlugins, which are dynamically loaded when called from the Client. The parameters are embedded in SOAP messages;
- The data management and data transfer between the IJPlugin Service and the Client.

3.2 GridPlugin development

Scientific users or programmers create new scientific software components in Java as GridPlugins. A GridPlugin represents a Java application in which one class implements the GridPlugin interface:

```
package org.fzk.grid.plugin;
import ij.ImagePlus;

public interface GridPlugin{
  public void setupPlugin(ImagePlus[]
      imgp);
  public String runPlugin(Properties
      arg);
}
```

Listing 1. The GridPlugin interface consists of two methods, which will be extended by scientific functionality.

Two methods must be implemented: The first method is the `setupPlugin()` method, in which the images from the web service are delivered as an array of ImagePlus objects [15]. The second method is the `runPlugin()` method, which is similar to the main method of a normal Java program. The arguments are passed as a `Properties` object. All required parameters and corresponding information are specified in a separate XML file (see listing 2).

```
<properties>
  <Integer label="IntVal"
    min="-5" max="10"/>
  <Double label="DoubleVal"
    min="-3.5" max="9.7"/>
  <String label="StringVal"
    default="all" regex="*"/>
</properties>
```

Listing 2. The property file defines the required parameters of a GridPlugin.

Valid data types are `Double`, `Integer` and `String`. For `Double` and `Integer` the GridPlugin developer defines a valid range of values. The `String` property has a `regex` parameter in which regular expressions can be submitted. The property file will be included in the JAR file of the GridPlugin.

3.3 The Client software including the graphical user interface

The GridIJ Client software is implemented as an ImageJ plugin and is called from the ImageJ graphical user interface. Invoking the Client software opens an additional graphical user interface shown in Fig. 3.

The user interface of GridIJ is divided into five tabs representing five function groups, respectively. From left to right the tabs are called: ”Connect”, ”Plugins”, ”Images”, ”Run” and ”Expert”.

The ”Connect” tab controls the initial connection of the Client to the GT4 Worker Nodes. During connection the GT4 MDS Index Server is addressed to get a list of all available GT4 Worker Nodes suitable for GridIJ. The user configures a number of servers which are automatically assigned according to their current work load. After that the IJPlugin or another web service is uploaded and deployed in the GT4 Worker Nodes.

After this step the connection between Client and GT4 Worker Nodes is established and GridPlugins, remaining from prior usage are distributed to all GT4 Worker Nodes of the new configuration. Now the user can start to upload his own GridPlugins and data (tabs ”Plugins” and ”Images”). The GridPlugins will be directly stored in the file system of the GT4 container.

A private resource location will be assigned to each user to store the data (images) and GridPlugins. This guarantees that different users do not interfere with each other. The images are distributed equally, if possible, to the GT4 Worker Nodes and are stored locally in the file system. Within ImageJ a direct processing of single active images is also possible.

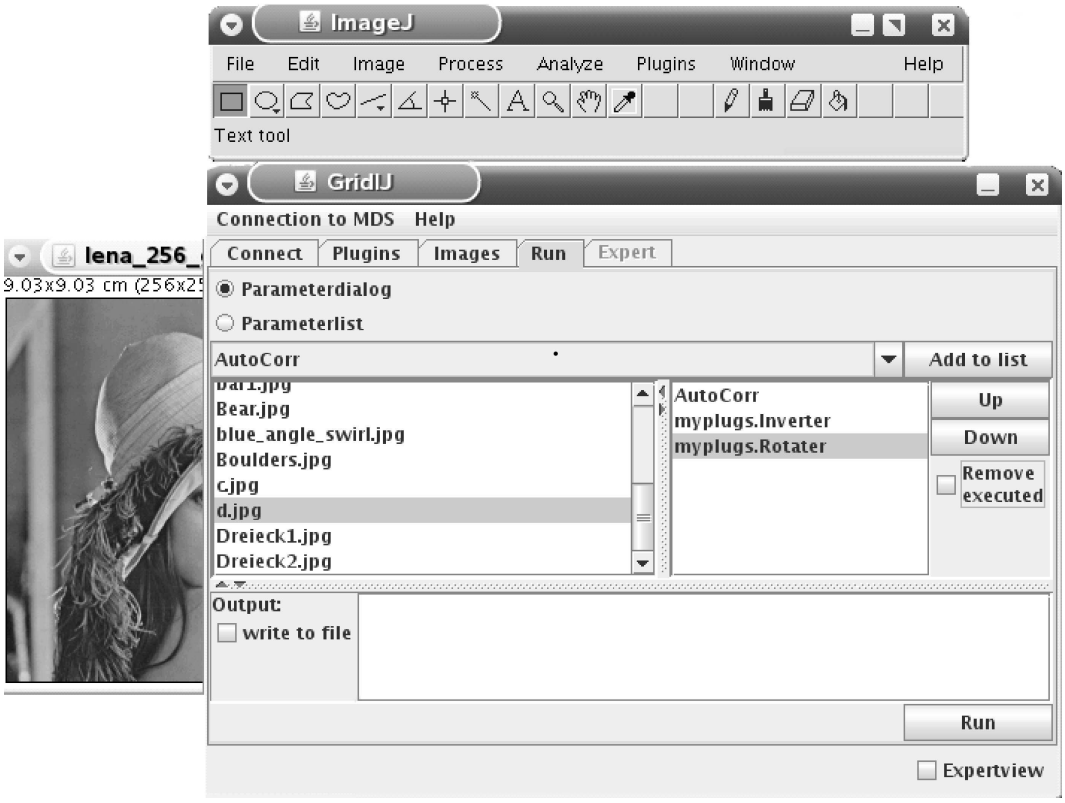

Figure 3. ImageJ with GridIJ. Shown is the ImageJ graphical user interface (upper window), an opened image (left) and the GridIJ graphical user interface (lower window). Within the active "Run" tab of GridIJ the workflow of the deployed GridPlugins (right list) and the distributed images (left list) are present.

Now the user can create a workflow by sequencing GridPlugins in the ”run” tab to a execution list. Running the workflow will start the execution on all GT4 Worker Nodes in parallel, applying the list of GridPlugins sequentially to the assigned images. Different parameter settings are possible for different GT4 Worker Nodes.

The generated results are returned to the Client after execution: Return values are displayed or the images are downloaded using the SecureStreaming Service.

4 Results

To give a quick estimation of the performance of the proposed GridIJ architecture an image registration algorithm based on local correlation has been pasted into a GridPlugin. The registration algorithm was originally implemented as a stand–alone Java program using the ImageJ library. Modifying it to a GridPlugin by implementing the GridPlugin interface took not more then 10 minutes programming time, demonstrating the ease of handling.

The registration was performed on a testbed consisting of 16 workstations. To estimate the parallel performance and the administrative offsets consisting of connecting time and data upload time, sequences of 64 images (192 Kbytes each) were processed. The average execution time of approximately eight seconds processing for a single image registration was intentionally chosen small to demonstrate the effects of the administrative offsets, connecting time and upload time. In typical grid applications the processing time will be much larger, thus decreasing the administrative effects and increasing the efficiency.

Since the GT4 Worker Nodes were also burdened by other tasks resulting in fluctuations in the processing time, four runs for each image set were used and averaged. The processing times composed of connection, upload and execution times were measured at the Client computer.

The performance of the GridIJ implementation with an increasing number of parallel GT4 Worker Nodes was compared to an original program run time (stand–alone execution) on a single workstation. Altogether the results show a

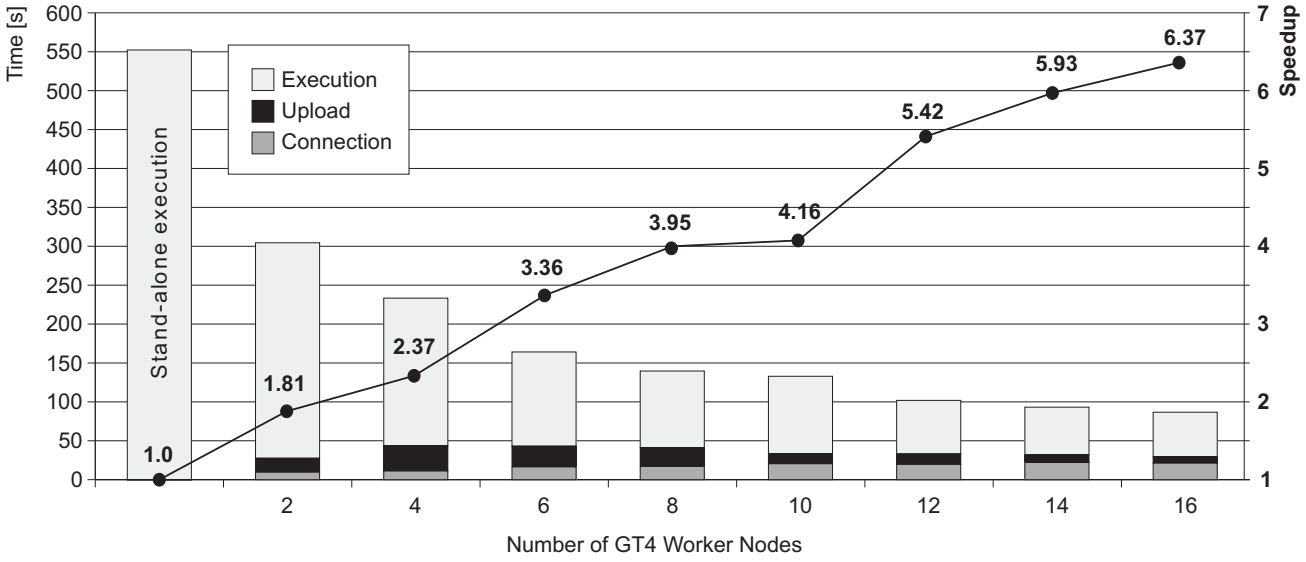

Figure 4. Comparison of the execution, connection and upload times for the original program (stand–alone execution) and the corresponding GridPlugin with a growing number of GT4 Worker Nodes. The left axis shows the processing time represented in the plotted bars, the right axis (bold) shows the corresponding speedup represented in the line plot and bold numbers.

typical behavior of parallel architectures: Figure 4 displays the processing time composed of execution, upload and connection times for different configurations. The workload for each GT4 Worker Node decreases, the more GT4 Worker Nodes are participating, and therefore the execution time decreases, like expected. The resulting speedup scales not perfectly, since the GT4 Worker Nodes were assigned according to their current work load and more ”slower” ones were incorporated. The execution times were measured at the Client computer and are dominated by the slowest GT4 Worker Node. Nevertheless, the resulting speedup grows approximately linearly with the number of GT4 Worker Nodes. The efficiency, defined as the ratio between the speedup and the number of GT4 Worker Nodes, can be regarded as a degree of the scalability. It decreases very slowly within the observed range and arrives 0.4 for the maximum number of GT4 Worker Nodes. Since the connecting time and upload time remain constant, we expect these administrative offsets will dominate the execution time for growing numbers of GT4 Worker Nodes resulting in an reduced efficiency.

5 Conclusion and Future Work

Regarding the original objectives of this work, it could be shown that the proposed design of a Grid computing architecture on the base of WSRF–compliant web services with dynamic service provisioning and management leads to an easy–to–use and easy–to–extend scientific problem solving environment. In contrast to standard Grid applications in job–based architectures, the scientific functions, embedded in web services, are executed immediately allowing nearly real time handling of spontaneous requests.

The demonstrated reference implementation for image processing, GridIJ, allows scientists to concentrate on the scientific questions and algorithms, not on the implementation problems appearing in Grid environments, web services and middleware architectures. The dynamic Grid service architecture, its management and administration remains hidden for the scientific users and the system operators, since it acts mostly automatically. Security, e.g. authentication, authorization and secure access to the deployed web services and data, is provided by GT4. The architecture is easily scalable and can be expanded by simply deploying two additional web services to new GT4 Worker Nodes.

Until now only a few performance tests have been accomplished. Thus efficiency and long–term stability have

not been sufficiently tested by concurrently accessing users and GridPlugin programmers.

Currently the architecture uses its own strategy to distribute the data and tasks to the GT4 Worker Nodes. The use of a resource broker [25] might improve the efficiency, if applicable. Furthermore for data transfer an in–house solution, the SecureStreaming Service, is used. It is considered to substitute this service by a standardized Grid solution, e.g. GridFTP.

In the next steps we will apply GridIJ in digital forensics. The comparison of image contents can be easily parallelized by GridPlugins, since the tasks are independent. This will allow to evaluate even new and complex algorithms with hundreds of images in acceptable time.

For the future it is suggested to contribute to the Higher Order Component — Service Architecture (HOC–SA) [6], which is a Globus incubator project [12] (dev.globus) to provide application programmers a higher level access to a Grid. Some of the ideas and implementations of GridIJ and HOC–SA show common features and might complement one another. Furthermore we plan to make the software freely available to a greater audience.

References

[1] The gLite middleware. http://glite.web.cern.ch/glite, 2007.

[2] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, L. Liming, and S. Tuecke. GridFTP: Protocol extensions to FTP for the Grid. Internet Draft, August 2001.

[3] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.

[4] D. Arnold, S. Agrawal, S. Blackford, J. Dongarra, M. Miller, K. Seymour, K. Sagi, Z. Shi, and S. Vadhiyar. Users' Guide to NetSolve V1.4.1. Innovative Computing Dept. Technical Report ICL-UT-02-05, University of Tennessee, Knoxville, TN, June 2002.

[5] T. Banks. Web Services Resource Framework (wsrf) – primer v1.2. Technical report, OASIS Open, 2006.

[6] J. Dünnweber and S. Gorlatch. HOC-SA: A Grid Service architecture for Higher-Order Components. In *International Conference on Services Computing, Shanghai, China*. IEEE Computer Society Press, 2004.

[7] J. W. Eaton. *GNU Octave Manual*. Network Theory Limited, 2002.

[8] I. Foster. Globus Toolkit version 4: Software for service-oriented systems. In *IFIP International Conference on Network and Parallel Computing*, pages 2 — 13. Springer-Verlag LNCS 3779, 2005.

[9] I. Foster and D. Gannon. The Open Grid Services Architecture platform, 2003.

[10] T. Friese, M. Smith, and B. Freisleben. Hot service deployment in an ad hoc grid environment. In *Proceedings of the 2nd Int. Conference on Service-Oriented Computing (ICSOC'04), New York, USA*, pages 75–83. ACM Press, 2004.

[11] T. Friese, M. Smith, and B. Freisleben. GDT: A toolkit for grid service development. In *Proc. of the 3rd International Conference on Grid Service Engineering and Management*, pages 131 — 148, 2006.

[12] The Globus Alliance website. http://www.globus.org/, 2007.

[13] Official Globus Toolkit documentation for GridFTP. http://www.globus.org/toolkit/docs/4.0/data/gridftp/, 2007.

[14] Gridlab Grid Application Toolkit website. http://www.gridlab.org/WorkPackages/wp-1/, 2007.

[15] ImagePlus class. http://rsb.info.nih.gov/ij/developer/api/ij/ImagePlus.html, 2007.

[16] T. Jejkal, T. Müller, R. Stotzka, M. Sutter, V. Hartmann, and H. Gemmeke. Grid services toolkit for process data processing. In *German E–Science Conference 2007, (GES 2007)*, 2007.

[17] Mathworks. Matlab. Mathworks, Inc., Natick, MA, 1999.

[18] R. Mondardini et al. GridCafe. http://gridcafe.web.cern.ch/gridcafe/, 2007.

[19] T. Müller, T. Jejkal, R. Stotzka, M. Sutter, V. Hartmann, and H. Gemmeke. Grid services toolkit for process data processing. In *Second IEEE International Conference on E–Science and Grid Computing, 2006*, Amsterdam, The Netherlands, 2006.

[20] W. Rasband. Imagej, U. S. National Institutes of Health, Bethesda, Maryland, USA. http://rsb.info.nih.gov/ij/, 1997-2006.

[21] Website of the SAGA Research Group. http://forge.gridforum.org/projects/saga-rg/, 2007.

[22] J. Schopf, L. Pearlman, N. Miller, C. Kesselman, I. Foster, M. D'Arcy, and A. Chervenak. Monitoring the grid with the Globus Toolkit MDS4. *Journal of Physics: Conference Series*, 46:521–525, 2006.

[23] Specification of SOAP. http://www.w3.org/TR/soap/, 2007.

[24] R. Stotzka, N. Ruiter, S. Pfleiderer, A. Malich, H. Gemmeke, and W. Kaiser. Ultrasound computer tomography for breast imaging. In *European Radiology*, 2003.

[25] W. Süß, W. Jakob, A. Quinte, and K.-U. Stucky. GORBA: A Global Optimising Resource Broker embedded in a Grid Resource Management System. In *IASTED PDCS*, pages 19–24, 2005.

[26] M. Sutter, T. Müller, R. Stotzka, T. Jejkal, M. Holzapfel, and H. Gemmeke. Inspector computer. In *German E–Science Conference 2007, (GES 2007)*, 2007.

[27] Website of the unicore forum. http://www.unicore.org/, 2007.

[28] S. Wolfram. *The Mathematica Book*. Wolfram Media, 2003.