Computing and Informatics, Vol. 27, 2008, 199–211

INTERACTIVE TECHNIQUES IN GRID COMPUTING: A SURVEY

Herbert Rosmanith, Jens Volkert

GUP Linz
Johannes Kepler University Linz
Altenbergerstrasse 69
A-404 Linz, Austria/Europe
e-mail: hr@uni-linz.ac.at

Abstract. In Grid computing, the dominating paradigm is batch processing. Grid middleware ships with batch-job support only, while lacking support for interactive applications. The reason is that grid middleware was developed for compute intensive jobs, which may run for a long time before a result becomes available. This leads to a "post-mortem" approach of analysing the output, possibly resulting in a waste of computing and research time. Adding the possibility to observe and steer the job during execution enables the researcher to modify job-parameters without restarting the entire job. In this paper, several interactivity support techniques are explored, followed by several examples proving their usefulness.

Keywords: Grid computing, interactivity, steering, visualisation

1 INTRODUCTION

In this section, the area of Grid computing is introduced first, followed by an introduction of interactivity. Section 2 presents various techniques to gain interactivity in Grid computing, then Section 3 deals with related work. Finally, Section 4 describes projects making use of the techniques presented herein, which have been successfully implemented in the past.

1.1 Grid Computing

Grid computing, introduced in 1998, is concerned with resource sharing, such as processing power and storage capacity, done in a coordinated way by "virtual organisations". The definition in [1] precisely describes the sharing and how it is controlled by the virtual organisations:

> The sharing that we are concerned with is not primarily file exchange but rather direct access to computers, software, data, and other resources, as is required by a range of collaborative problem-solving and resource-brokering strategies emerging in industry, science, and engineering. This sharing is, necessarily, highly controlled, with resource providers and consumers defining clearly and carefully just what is shared, who is allowed to share, and the conditions under which sharing occurs. A set of individuals and/or institutions defined by such sharing rules form what we call a virtual organisation.

Similar systems such as cluster computing and distributed computing can not be considered Grid computing. In cluster computing, the resources are usually owned and controlled by one central authority, there is no virtual organisation defining sharing rules. Distributed computing differs from Grid computing in that tasks run by distributed computing usually execute on idle workstations primarily used for other tasks, while, in contrast, Grid computing leans to dedicated systems.

One of the largest users of Grid technology is CERN. CERN created the Large Hadron Collider Grid (LHC Grid, LCG) to cope with 15 petabytes per year, which is expected to be produced by the LHC in Geneva. In 2004, the LCG spawned the EGEE-project (Enabling Grids for E-sciencE). The task of EGEE is to allocate resources, user-groups and applications from all over the world, to establish the largest production quality Grid. In parallel, development of the "gLite" [2] middleware was started, which is based upon the Globus Toolkit [3] and consists of more than 200 software packages.

The development of this middleware was done by CERN and for the needs of CERN. This means storing and processing huge amount of data, leading to long-term jobs which a typical physicist at CERN is interested in. Physical simulations can easily take days, weeks or even months to finish. In a multi-user environment this leads to the development of a computing environment in which jobs are put into a queue and processed one after another. The output of the job can only be analysed when the job is done. These are the typical characteristics of a batch-job environment. In such an environment, it is not possible to examine intermediate results or even intervene to change the job's parameters. This can, at worst, lead to a waste of valuable research time and to a waste of resources. We address this problem by adding interactivity to middleware.

1.2 Interactivity

In [4], J. C. R. Licklider talks about a symbiosis of man and machine: "Man-Computer symbiosis is an expected development in cooperative interaction between men and electronic computers." Quoting Webster's Dictionary, he understands that symbiosis means "living together in intimate association, or even close union, of two dissimilar organisms". In saying this, Licklider assigns the same significance as that of an organism to a machine.

The same trend can be observed in early Artificial Intelligence (AI). In 1970, Marvin Minsky predicted that in about three to eight years, computers will be as intelligent as the average human being. Working with such machines would then not simply mean using a computer. Working with a human like, intelligent computer would rather mean: social interactions. This way, a term which originally was coined in sociology [5] turned to become a well-known term of computer science.

One of the first projects leading to improvments in interactive computing was the cold war project SAGE (Semi-Automatic Grounding Environment) [6], on which Licklider worked, too. SAGE task was to detect and intercept hostile enemy airplanes, preferably bombers from the former Soviet Union. Since the users of SAGE were common soldiers, a complex man-machine interface has been developed for the first time, resulting in a system to display text and graphical drawings, using a light gun (a predecessor of the light-pen) to interact with SAGE.

Barely ten years later Sketchpad [7] appears. It was developed 1961 to 1963 as part of the Ph. D. thesis of Ivan Sutherland at MIT. Sketchpad was already using a lightpen and a cathode-ray-tube (CRT) and thus can be seen as a first step towards interactively working with computers.

Nevertheless, the title "father of interactivity" has been assigned to Douglas Engelbart. In December 1968, he presented the first computer mouse to the public in a conference in San Francisco. The body was carved out of wood, and it had only one button, because there was no room for more. Using two right-angledly mounted wheels, the movement of the mouse was tracked and its position reconstructed by the computer.

Despite this early success in the research laboratories yet about twenty years had to pass until interactivity was accepted in the public. The majority of computer work in these days happened to take place in computing centres: Halls filled with noisy, cabinet-sized systems from IBM and other companies. Data and source code were recorded on punch cards, and given to an operator to feed them into the computer system. As an undergraduate computer-science student, one usually had to wait a whole day in order to get the output of an exercise program. Home-computers such as the Apple-II (1977), the VC-64 (1982) from Commodore, the Sinclair ZX Spectrum (1982), the IBM PC (1981), the Macintosh (1984) appeared ten to fifteen years later. These early systems could not compete with the computing and storage capacities of the computing centres. Microsoft's mission statement: "A PC on every desk and in every home" is by no means only a for-profit maxim, but attests to a time when computers were not as omnipresent as they are today.

Thanks to the advancements made in computer technology, personal computers began to spread, replacing established systems and finally changed the way humans work with computers.

It is ironic that the descendants of the early slow PCs, which heralded the "interactive evolution" in computing, now service probably in the same halls of the computing centres of that time in the form of super computers, and, by being operated with non-interactive Grid middleware, exhibit the same shortcomings as the systems which they replaced.

2 INTERACTIVITY AND THE GRID

In most of current Grid middleware, two important properties of interactivity are missing: bidirectional data flow and event based data processing.

In interactive programs, data flows from and to the user as the program executes. This allows users to control and steer the program by sending new data using input devices such as keyboard, mouse, position trackers, video cameras and the like. In Grid middleware, input data can not be provided to programs interactively; instead, the whole input data set has to be available before the program starts to execute.

To interact with a computer system in a convenient way, messages to and from the system have to be exchanged at a speed fast enough to satisfy the problem's requirements. For instance, pressing the button of a graphical user interface should result in processing and confirmation by the program almost instantly. GUIs implement this feature by performing user centered, event based processing. Grid middleware, on the other hand, polls for the advent of new data. Since the middleware was not designed with interactivity in mind, the polling interval is set to a relatively high value, thus reducing system load, but also leading to slow notification times.

Examining the so-called interactive job manager of the Globus Toolkit reveals that it lacks the above mentioned features. This can be checked by running a simple test program, such as incrementally counting from zero to infinity, pausing for one second each increment. Starting this program with the interactive job manager (also called *jobmanager-fork*) in the Grid shows the following behaviour: for about ten seconds, nothing is displayed at the terminal. Then, the numbers from zero to nine appear all of a sudden. This behaviour results from polling for data instead of performing event based data transport.

Checking whether a user can send data to a Grid job (while it is running) can be done by examining the job's file descriptors. On a Linux system, the proc-filesystem holds the necessary information. It can be seen that by default, the input file descriptor zero (*stdin*) is connected to the null-device, */dev/null*, thus disabling any input to the Grid job. Even specifying an input location using the *stdin*-keyword in the Resource Specification Language (RSL) when submitting a job to the Grid does not lead to success: this input data set is read in as a whole by the middleware and put into the GASS-cache [8] before the jobs starts.

Conclusion: In order to add interactivity to the Grid, a fast and bidirectional way to enable communication between the Grid job and the user's interface is required. In the following, various methods how interactivity in Grid computing has been achieved are presented.

2.1 Glogin for GT2

In the Globus Toolkit version 2 (GT2), a client (such as *globusrun*) connects to the *globus-gatekeeper* and requests the execution of a job by passing a job description, specified by various keywords in RSL. The *globus-gatekeeper* delegates this task to the Globus job manager. When using the "interactive" job manager, *jobmanager-fork*, a unidirectional connection from the job manager to the client is present. Therefore, it was first explored whether it is possible to reuse this connection. This was not the case, because this connection always remains associated to the job manager. Consequently, interactivity in GT2 can only be achieved by establishing a separate connection.

A demand for a solution has been that Grid middleware must not be modified. This resulted in the development of a lightweight tool called *glogin* [9], which executes on top of Globus, requiring no changes to the middleware whatsoever. *glogin* implements pseudo terminals (PTY), allowing shell access to grid nodes, I/O interception of standard input, output and error file descriptors, TCP and X11 forwarding. To accomplish this, a lightweight protocol (called the GSH-protocol) has been defined and implemented in *glogin*.

How does *glogin* work? Just like *globusrun*, *glogin* contacts the *globus-gatekeeper*, but instead of specifying an arbitrary command to execute, *glogin* requests execution of a remote copy of itself. Prior to submitting, the local *glogin* allocates a TCP port and stores the address in the RSL. When the remote *glogin* starts executing, it obtains the TCP address from its parameter list. Additionally, the remote *glogin* allocates another TCP port and writes the address to its output file descriptor. The local *glogin* receives the output sent by the remote process. Now, both instances of *glogin* try to establish a connection to each other. By using two connections, the probability of successful connection establishment increases. The connection established first remains, while the second connection is discarded. The remote *glogin* now executes the command the user specified (or the login shell, if the parameter is omitted) and intercepts its I/O. Data from the command is transported securely by using the GSS-API [10, 11]. The local *glogin* receives all the input from the user, for instance, from the terminal it is connected to and securely sends it to remote *glogin*, which in turn sends the user input to the specified user command executing in the Grid. This way, a secure, interactive date transport between both processes is formed.

To speed up the transmission of the information about the remotely allocated TCP port, remote *glogin* has to flush the GASS-cache; otherwise, the connection establishment is delayed for ten seconds, the duration of the polling interval. One method to flush the cache is to fork a child process fast enough (within two seconds)

and then terminate the parent process, with the child process continuing processing.

2.2 An Interactive Job-Manager for GT2

In contrast to the *glogin* tool described above, this section describes a solution which is part of the Grid middleware. Analysing how jobs are started on the Grid led to the finding that implementing a separate job manager [12] is the best way to provide interactivity in the desired manner.

The reason why *glogin* creates a separate communication channel is the presence of the job manager, which can not not release the initial connection created by the client. Since now a separate job manager is designed, the question arises whether it is possible to reuse the initial connection this time. It turned out that the existing connections are still in use for sending and receiving job status requests/replies. Again, a separate connection is required for interactive data exchange. The connection is established such that an arbitrary client creates a TCP listener, while the job manager connects back to the client. When submitting a job to the interactive job manager, a parameter describing where to connect back is passed to the interactive job manager. This has been utilised by expanding the RSL, allowing to specify an *interactive* attribute:

```
(interactive=proto://host<:port>)
```

The interactive job manager supports a subset of the RSL only. Attributes not supported are ignored and throw a warning message. The following attributes are recognised:

```
executable, arguments, count, interactive, jobtype,
directory, environment
```

The Interactive job manager supports MPI and multiple jobs.

2.3 The Interactive Job-Manager for GT4 in GT2 Compatibility Mode

In 2004, GT2 was replaced by the Globus Toolkit 4 (GT4). GT4 offers Web Service [13] based job submission instead of the previous, pre-Web service protocol. To ease the transition from GT2, GT4 still offers pre-Web service support to be built. Today, pre-Web services are still in use on many sites using GT4 middleware. In GT4, the previous IO layer was replaced by an "extended IO" (Globus-XIO) layer, the IO interface used before was implemented by adding a compatibility layer. Unfortunately, this layer does not support the way the interactive job manager (see above) needs to access relevant data: the type definition *globus_io_handle_t* does not contain a file descriptor entry any more. Instead, *globus_xio_handle_cntl()* and *globus_xio_server_cntl()*, requesting *GLOBUS_XIO_TCP_GET_HANDLE* have to be used. The problematic part, however, is that GT4 hides previously accessible data

structures by using opaque type definitions, not providing an API for getting/setting required values. Developers writing external programs have to copy the definition, which is often not even available as a C-header file, but a private data structure, declared inside the source-code of the corresponding program.

2.4 Glogin with Web-Services

The current movement in Grid middleware, with respect to providing services, is replacing proprietary solutions by making use of Web services. Two prominent examples are the Globus Toolkit and Unicore [14], the latter switched to Web services when releasing version 6.

To keep up with this trend, methods to provide interactivity in the presence of Web services have to be explored. In GT4, the Web service enabled *globusrun-ws* is the counterpart of the pre-Web service *globusrun*. Thus, the Web service port of *glogin* uses the same job submission mechanism, the *ManagedJobFactory Service*, the *ManagedJob Service*, the *ManagedExecutable Job Service*, the *Delegation Service* and the *Subscription Manager Service*. Using these services, *glogin* invokes a remote copy of itself and creates a separate, bidirectional, event-based data transport channel. Once the connection is established, processing continues as described above.

Although *globusrun-ws* provides "data-streaming", its behaviour does not differ from *globusrun*. Firstly, there is no way to send data from the user interface to the running job in a fashion suitable for interactive communication. Secondly, *globusrun-ws* exhibits the same slow polling based mechanism as present in GT2. The implementation in GT4 differs in that "data-streaming" is implemented by the client (*globusrun-ws*) instead, using grid-ftp to poll for new data every ten seconds.

2.5 An Interactive Web Service for GT4

The major disadvantage of creating a separate communication channel is that connection establishment might fail. The security needs of Grid sites require the installation of firewalls, resulting in a certain possibility that a connection might be blocked by the network because of firewall rules.

Unlike GT2, a *globus-gatekeeper* is not used in GT4 any more. Instead, Globus executes within a Tomcat [15] container which in turn invokes the requested service. Therefore, a Web service client (such as e.g. *globusrun-ws*) is directly connected to the invoked service. By developing a dedicated Web service it has been explored whether this very connection can be reused to support interactivity.

When invoking a Web service, the request is encapsulated in SOAP/XML and sent to the Globus container, usually via HTTPS. After processing, the reply also is encapsulated in XML and sent to the requester. After exchanging request and reply, both partners terminate the connection. For demanding interactive appli-

cations, a persistent connection is required. The *InteractiveService* facilitates this requirement by turning the Web service connection into a persistent one.

The mechanism works as follows: a client connects to the Globus container by using asynchronous Web service invocation, which means that the client has control over when to send a request and when to receive a reply. After successfully sending the request, but before reading the reply, the client duplicates its connection to the Web service. The client can find the connection by examining its file descriptor table. On the container side, after having received a request, the Web service starts a helper process (thus creating a duplicate of the Web service connection), which passively waits for a message from the client. The Web service returns an empty reply to the client and terminates the connection. Now, the client sends a certain message to the helper process, indicating that it is ready to continue processing. This is perfectly valid, since although the initial connection does not exist any more, both the client and helper process are able to communicate by using their duplicated connections. Data transport is protected using the GSS-API.

2.6 I2Glogin – Glogin for the Int.Eu.Grid

For use within the Interactive European Grid project (I2G) [16] the *glogin* tool was modified. Connection establishment only occurs from the Grid to the user interface, making the GASS-cache flushing mechanism unnecessary. Thus, problems with local schedulers can be avoided [17].

When invoked locally, *i2glogin* allocates a TCP listener and displays this address in textual form on its output. The output is intercepted by I2G middleware and stored in a Job Description Language (JDL) file, describing which job to execute on the Grid. I2G middleware then submits the job. When the Grid job starts, remote *i2glogin* is started, too, contacting the local *i2glogin* using the provided address. Remote *i2glogin* intercepts all IO of the Grid job, thus establishing an interactive connection between the job and the user interface.

2.7 Interactivity using the Condor Bypass

As of version 3, interactivity support has been added to gLite, the grid middleware developed within the EGEE project. gLite uses the "Condor Bypass" [18, 19] mechanism, although "Condor" and "Bypass", to quote Condor's FAQ: "are separate programs. Bypass does not require Condor, nor does Condor use Bypass." Condor Bypass modifies the environment variable *LD_PRELOAD* to intercept *open, read, write, close* system calls implemented in the C-library. As a result, only programs which are dynamically linked with the C-library can be used interactively with this method. Modifying *LD_PRELOAD* is a great debugging technique, as, for instance, BuGLe[20] demonstrates. To prevent potential conflicts with debuggers, gLite middleware should refrain from using *LD_PRELOAD*.

According to the online documentation [21], interactive jobs in gLite do not work well:

> Due to timing or buffering problems, glite-wms-job-submit starts reading input before it has displayed first part of the text. Also the second input is skipped and the program finishes. Running non-interactive jobs using this facility also shows problems of repeated and duplicated output.

and

> Although gLite gives some basic interactivity between the user and the job, implementation is still having significant usability problems as shown by the example job.

According to the online manual of gLite's Grid console, the user has to specify *LD_PRELOAD* manually to overwrite system calls the C-library. This is a possible source of error, since common user should focus on application specific problems instead of learning how to operate middleware manually.

By default, gLite opens a window to a X-server the user works with. If an X-server is not available, textual interaction takes place in the user's terminal. The example in the manual gives the impression that user input is processed in a line-oriented fashion. In contrast to this, *glogin* offers a binary character- and block-oriented IO mature enough for implementing complex GUIs (as described later).

In CrossGrid [22], Bypass was used such that applications were linked with the bypass library, thus avoiding the use of *LD_PRELOAD*, enabling static applications. Of course, this restricts this method to programs available in source code only.

3 RELATED WORK

Interactivity is not restricted to fancy GUIs, but also includes accessing a computer via a monitor and a keyboard, called a "terminal". In a distributed environment, a way to remotely access a host is required. In the Internet, such a "Networked Virtual Terminal" (NVT) [23], is implemented by the telnet program. An alternate implementation is the remote shell (RSH), which was later replaced by the secure shell (SSH) [24]. For the Grid, the SSH was adopted to use the Grid Security Infrastructure (GSI), resulting in a Grid aware SSH (GSI-SSH). The Globus Toolkit 4 now contains the GSI-SSH by default. The disadvantage of using GSI-SSH is that a GSI aware ssh server (ssh daemon, sshd) needs to be installed, configured and started (eventually from inetd) outside of the context of the *globus-gatekeeper* or Globus servlet container. Starting jobs from GSI-SSH results in access to the host without being subject to scheduling mechanisms. Hence, GSI-SSH can not be a substitute for the usual job submission mechanism.

In "Interactive Grid Architecture for Application Service Providers" (I-GASP) [25], interactive sessions for the grid are introduced. The I-GASP solution makes use of Virtual Network Computing (VNC) [26], and thus can be compared to X11-forwarding with *glogin*. Using *glogin*, VNC sessions can be supported using

the TCP port forwarding feature; however, *glogin* is not limited to VNC but can also support a local visualisation frontend to remotely steer a Grid job.

Often, Grid portals are labeled "interactive". Here, the term "interactive" rather refers to an interactive environment built on top of non-interactive middleware, as can be seen in "Grid ENabled Interactive Environment" (GENIE) [27]. In such a Grid portal, accessing the Grid job is not necessarily interactive. In contrast, the methods presented above focus on interactive jobs.

In the Unicore 6 grid, the VISIT/GS [28] architecture for online visualisation and steering is presented. VISIT/GS uses SSH to get interactive access to the Grid job. As an interesting alternative, the approach used in the *InteractiveService* could be examined and implemented for the Unicore middleware. This way, using SSH could be avoided.

4 APPLICATION EXAMPLES

By describing a few examples, we demonstrate the usefulness of interactivity in Grid.

One of the first applications that made use of interactivity in the Grid was the "Virtual Radiology Explorer" (DesktopVRE). The DesktopVRE acts as a visualisation client for a medical application [29, 30] running on the Grid. Since DesktopVRE is a program using the Visualisation Toolkit (VTK), a wrapper was built around *glogin* to interface with the VTK-libraries.

The DesktopVRE showed that the amount of data produced by simulations such as the medical application can grow so tremendously large that performing the visualisation entirely in the Grid and transporting completely rendered frames is more efficient. The "Grid Video tool" (GVid) [31] drives this idea even further by using advanced video compression techniques to reduce network load. To demonstrate the feasibility of this method, the "father of first person shooters", the Doom-game (and later, the Quake-game) was ported to make use of the GVid-interface by replacing the original X11-interface. The game entirely runs in the Grid, while the graphical output is displayed in the Migrating Desktop (MD) [32].

To demonstrate a counter-example requring only low video bandwidth, a chess frontend (Xboard) has been implemented for the MD. While still being a demanding application, a chess application (e.g. gnuchess) has very low needs for visualisation, thus the visualisation, like drawing and moving chess pieces on the screen, can be done entirely locally. The protocol between the chess frontend and the chess application is extremely lightweight, since most of the time only piece movement instructions such as "e2-e4" needs to be transported.

As the last example, we describe I2G's piloting application, the "Interactive Fusion Visualisation and Simulation". The simulation executes in the Grid, using GVid to display the graphical output locally in the MD. A local GUI, displayed in the same window in the MD, is used for steering the simulation. To implement the transport of GUI events, such as the user pushing buttons or moving sliders, the GVid protocol had to be extended. State information about the GUI element

operated is sent to the remote application in the form of a "name = value" pair. Encoding this state information in XML has been thought about, but has been discarded for reasons of efficiency and implementation.

5 ACKNOWLEDGMENT

We would like to thank our colleague Paul Heinzlreiter for proof-reading.

The work described in this paper was supported in part by the European Union through the FP6-2005-Infrastructures-7-contract No. 031857 project "Int.eu.grid".

REFERENCES

[1] Foster, I.—Kesselmann, C.—Tuecke, S.: The Anatomy of the Grid – Enabling Scalable Virtual Organizations. International Journal of Supercomputer Applications, 2001.

[2] Laure, E. et al.: Programming the Grid using gLite. EGEE Technical Report, 2006.

[3] Foster, I.—Kesselman, C.: Globus: A Metacomputing Infrastructure Toolkit. The International Journal of Supercomputer Applications and High Performance Computing. Vol. 11, 1997, No. 2, pp. 115–128

[4] Licklider, J. C. R.: Man-Computer-Symbiosis. 1960, `http://memex.org/licklider.pdf`.

[5] Morris, Ch. W. (ed): Mind, Self and Society: From the Standpoint of a Social Behaviorist. George Herbert Mead, University of Chicago, 1934.

[6] Everett, R. R. (ed.): Special Issue: SAGE (Semi-Automatic Ground Environment). Annals of the History of Computing, 1983.

[7] Sutherland, I. E.: Sketchpad, a Man-Machine Graphical Communication System. MIT, 1963.

[8] Bester, J.—Foster, I.—Kesselman, C.—Tedesco, J.—Tuecke, S.: GASS: A Data Movement and Access Service for Wide Area Computing Systems. Sixth Workshop on I/O in Parallel and Distributed Systems, May 5, 1999.

[9] Rosmanith, H.—Kranzlműller,D.: glogin – A Multifunctional Interactive Tunnel into the Grid. In: R. Buyya (ed.), Proc. Grid 2004, 5th IEEE/ACM Intl. Workshop on Grid Computing, IEEE Computer Society, ISBN 0-7695-2256-4 Pittsburgh, PA, USA, pp. 266–272.

[10] Linn, J.: Generic Security Service Application Program Interface. RFC 2743, Internet Engineering Task Force, January 2000.

[11] Wray, J.: Generic Security Service API Version 2: C-Bindings. RFC 2744, Internet Engineering Task Force, January 2000.

[12] Rosmanith, H.—Kranzlműller, D.—Volkert, J.: An Interactive Job Manager for Globus. In: Roberto Moreno-Daz, Franz Pichler, Alexis Quesada-Arencibia (eds.), Computer Aided Systems Theory – EUROCAST 2007, 11th International Conference on Computer Aided Systems Theory, pp. 431–442.

[13] Alonso, G.: Web Services: Concepts, Architectures and Applications. Springer, 2004.

[14] Erwin, D.–Snelling, D.: UNICORE: A Grid Computing Environment. Lecture Notes in Computer Science, 2001.

[15] Tomcat Home Page: http://jakarta.apache.org/tomcat.

[16] Interactive European Grid Project, http://www.interactive-grid.eu.

[17] Rosmanith, H.—Praxmarer, P.—Kranzlmüller, D.: Jens Volkert: Towards Job Accounting in Existing Resource Schedulers: Weaknesses and Improvements. High Performance Computing and Communication (HPCC 2006), pp. 719–726, Munich, Germany, 2006.

[18] Thain, D.—Livny, M.: Bypass: A Tool for Building Split Execution Systems. In the Proceedings of the Ninth IEEE Symposium on High Performance Distributed Computing, Pittsburg, Pennsylvania, pp. 79–85, August 1–4, 2000.

[19] Thain, D.—Livny, M.: Multiple Bypass: Interposition Agents for Distributed Computing. Journal of Cluster Computing, Vol. 4, 2001, pp. 39–47.

[20] BuGLe Home Page: http://www.opengl.org/sdk/tools/BuGLe/.

[21] Interactive jobs using bypass (online documentation), http://wiki.egee-see.org/index.php/Interactive_jobs_using_bypass.

[22] The Crossgrid Project, http://www.crossgrid.org.

[23] Postel, J.: Telnet Protocol specification. ISI, June 1980.

[24] Tatu Ylőnen: SSH Secure Login Connections over the Internet. Sixth USENIX Security Symposium, pp. 37–42, Proceedings, SSH Communications Security Ltd., 1996.

[25] Basu, S.—Talwar, V.—Agarwalla, B.—Kumar, R.: Interactive Grid Architecture for Application Service Providers. Mobile and Media Systems Laboratory, HP Laboratories Palo Alto, Technical Report, July 2003.

[26] Richardson, T.—Stafford-Fraser, Q.—Wood, K.—Hopper, A.: Virtual Network Computing. IEEE Internet Computing, Vol. 2, 1998, No. 1, pp. 33–38.

[27] GENIE: Grid ENabled Interactive Environment, http://www.npaci.edu/online/v5.5/genie.html.

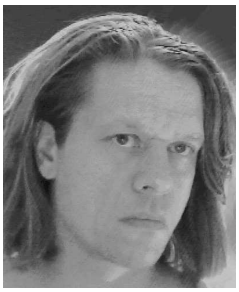
[28] Riedel, M.—Frings, W.—Dominiczak, S.—Eickermann, Th.—Mallmann, D.—Gibbon, P.—Dussel, Th.: VISIT/GS: Higher Level Grid Services for Scientific Collaborative Online Visualization and Steering in UNICORE Grids. Sixth International Symposium on Parallel and Distributed Computing (ISPDC'07), 2007.

[29] Tirado-Ramos, A.—Ragas, H.—Shamonin, D. P.—Rosmanith, H.—Kranzlmüller, D.: Integration of Blood Flow Visualization on the Grid: The FlowFish/GVK Approach. European Across Grids Conference 2004, pp. 77–79, Nicosia, Cyprus, 2004.

[30] Sloot, P. A. M.—van Albada, D. G.—Zudilova, E.—Heinzlreiter, P.—Kranzlmüller, D.—Rosmanith, H.—Volkert, J.: Grid-Based Interactive Visualisation of Medical Images. S. Norager (editor), Proceedings of the First European HealthGrid Conference, January 203, pp. 57–66, Commission of the European Communities, Information Society Directorate-General, Brussels, Belgium, 2003.

[31] Polak, M.—Kranzlműller, D.: Interactive Videostreaming Visualization on Grids. In: J. Dongarra, B. Tourancheau (Eds.), Cluster and Computational Grids for Scientific Computing. Future Generation Computer Systems, Special Section, Elsevier Science, Amsterdam, The Netherlands, Vol. 24, 2008, No. 1, pp. 39–45.

[32] Kupczyk, M.—Lichwała, R.—Płóciennik, M.—Wolniewicz, P.: "Applications on Demand" as the Exploitation of the Migrating Desktop Future Generation Computer Systems. Vol. 21, 2005, No. 1, pp. 37–44.



Herbert Rosmanith graduated in Computer Science from the University of Linz in 1998. In 2004, he joined GUP and started working in the EU-funded projects Crossgrid and Interactive European Grid. His main research interest is interactivity support for Grid computing.