Architecture of the Grid Services Toolkit for Process Data Processing

T. Jejkal[1], T. Müller[2], R. Stotzka[1], M.Sutter[1], V. Hartmann[1] and H.Gemmeke[1]

[1] Institute for Data Processing and Electronics, Forschungszentrum Karlsruhe GmbH, Hermann-von-Helmholtz-Platz 1, 76344 Eggenstein-Leopoldshafen, Germany
[2] Institute for Nuclear and Energy Technologies, Forschungszentrum Karlsruhe GmbH, Hermann-von-Helmholtz-Platz 1, 76344 Eggenstein-Leopoldshafen, Germany
email: Thomas.Jejkal@ipe.fzk.de
phone: (+49 7247) 82 4042

Abstract

Grid is a rapidly growing new technology that will provide easy access to huge amounts of computer resources, both hardware and software. As these resources become available soon, more and more scientific users are interested in benefiting from them. At this time the main problem accessing the Grid is that scientific users usually need big knowledge of Grid methods and technologies besides their own field of research. To fill the gap between high–level scientific Grid users and low–level functions in Grid environments the Grid Services Toolkit (GST) is developed at the IPE. Aimed to simplify and accelerate the development of parallelized scientific Grid applications, the GST is based on Web services extended by a rich client API. It is especially designed for the field of process data processing providing database access and management, common methods of statistical data analysis and project specific methods.

1 Introduction

Process data processing always have had very high demands on computing resources. Nevertheless in the near future the demands of many process data processing projects will even grow further up to a point where data processing and data storage on local workstations will not be feasible anymore. The best possibility to exhaustively fulfill the requirements of such projects in respect of standardization and stability is the use of oncoming Grid Computing technologies. Grid will finally provide access to today unimaginable amounts of computing resources. Therefore we naturally want to profit from these forthcoming resources by applying Grid technologies in process data processing projects. At a first glance this sounds fairly simple. Ideally the Grid is accessed from any process data processing application by simply using an Application Programming Interface (API). However, in real life Grid Computing is still a scientific field of its very own. At this time, scientists willing to use the Grid are very often forced to learn job description languages for cluster computing or they are urged to implement their own services onto low–level interfaces additionally

to programming their own clients. Also a good deal of background knowledge about Grid Computing and the particular computing environment like eg. locally installed software packages and operating systems is needed. This results in a large overhead and therefore in a low acceptance of Grid Computing in scientific communities, where it is not yet mandatory.
For integrating Grid technologies in our process data processing projects at Forschungszentrum Karlsruhe we need to achieve its acceptance in our community. Thus it is essential to provide easy–to–use high–level scientific methods. One imaginable use case is defined by the ultrasound computer tomograph (USCT) [1], an in–house project of IPE. The objective of this project is to detect breast cancer in a very early stage of appearance to allow a promising treatment of patients. One single USCT measurement causes about 20 Gbyte of signal data. One sequentially executed reconstruction into a 3D–volume takes, depending on the image resolution, several days up to weeks of computing time. The fact that there are no dependencies between single slices of the reconstructed volume makes a reconstruction easily parallelizable. USCT and almost all process data processing projects are sharing an equal processing structure. They need (high–performance) data transfer, the ability to execute their implemented algorithms, mostly implemented by using legacy–applications like Matlab [2], Octave [3] or ROOT [4] and they need a platform to integrate project–specific, highly–optimized functionalities. Thus an integration of this structure into a toolkit, which is seamlessly integrated into a Grid environment and can be accessed by an easy–to–use API, is desirable. The functional demands on such a Grid Services Toolkit (GST) are:

- High–performance and reliable data transfer,
- mechanisms for using legacy–systems accessible in the Grid and
- extension points to allow project–specific implementations to be integrated into the toolkit.

To realize these needs the technology of WSRF–compliant Web Services is chosen as it provides an easy–to–use access to high level functions [5]. Furthermore Web services have some very worthwhile advantages which fit perfectly into the scope of architectural demands on a GST:

- Web services easily scale in growing systems and are arbitrarily extendable.
- They allow a highly–modular design by dividing between the accessing–side (client) and the functional–side (server).
- By providing a rich client API GST should offer intuitive access and full transparency.
- Web service calls can be easily parallelized.
- Accessing them is independent from the underlying operating system and the programming language.
- They represent an open standard and therefore are future–proof.

The fundament of the GST was defined by the choice of WSRF-compliant Web services supported by the Globus Toolkit [6] as a middleware system. To be platform–independent all implementations should be done in Java. Available WSRF implementations in C#, C or Perl were not considered because we want to

realize platform–independence on the server– and on the client–side. Due to the fact that the Globus Toolkit does not provide any process data processing specific functionalities the next step is collecting knowledge about projects realizing one or more of the defined process data processing tasks to avoid re–implementations or to find solutions which can be included partly or fully into the GST. Some general software packages, as well as different Grid middleware systems, were discussed within a first architectural draft of the GST [7].
The following chapter 2 presents other projects which may be feasible in the scope of process data processing. Chapter 3 describes the detailed architecture of the GST and its components as well as details of their implementation. Chapter 4 discusses the current state of the project and presents ideas on how to proceed.

2 Related Work

2.1 Data–Transfer and –Access

An essential part of process data processing is data handling. There are quite a lot of projects dealing with data access in the Grid. A common way accessing files in the Grid is using GridFTP [8]. It is dedicated to secured high–performance file transfers and is deemed to be an established standard for file access in the Grid. The disadvantages of GridFTP are the restriction on file transfers and the platform–dependency on the server–side. Nevertheless an integration of GridFTP support should be realized later separately and as far as possible platform–independent.
The Storage Resource Broker (SRB) [9] is intended to realize transparent data access on the Grid by abstracting the location of data using a virtual file system. It utilizes a central Metadata Catalogue Database (MCAD) to store file information and a network of SRB servers to store the files in file systems, tape stores or databases. The data is accessed by a SRB client using virtual filenames. For the GST it is not quite feasible because SRB is a large–scale system which needs a lot of administrative overhead. Existing data stores are hard to integrate due to the use of virtual filenames. They have to be imported manually which causes less acceptance in our aimed community.
Another project dealing with data access is the Open Grid Services Architecture – Data Access and Integration (OGSA–DAI) [10]. OGSA–DAI is intended to provide standardized access to data resources on the Grid. OGSA–DAI is designed to be extensible easily as well as platform– and programming–language independent. A worthwhile extension to default access methods like JDBC [11] is securing the access to data resources by using X.509 certificates [12], [13]. OGSA–DAI is implemented as WSRF–compliant Web service. It should be integrated as far as possible and should be extended to fulfill special demands eg. accessing currently unsupported types of data resources.

2.2 Statistical analysis

There are some ambitions in this direction for example the Cactus Project [14]. Originally designed for parallel numerical relativity it became a more general platform as the code was rewritten for large scale computing. Finally a Grid–enabled version of Cactus was developed for the Globus Toolkit. Unfortunately Cactus provides only a few, very specialized mathematical methods and uses a proprietary interface definition. Therefore it does not qualify as basis for the GST but maybe can be integrated at a later date.
The project Weka4WS implements the functionality of the well known Weka toolkit as Web service for Globus Toolkit [15]. Accessing Weka4WS is solved by using a graphical client interface. Weka4WS is dedicated to data mining tasks but it unfortunately uses proprietary data formats.
Alltogether none of the evaluated tools fully fits into the scope of process data processing. They may be integrated at a later date driven by special demands of single projects.

2.3 Project specific tasks

Apart from generic tasks there is the need of special features for almost every project. It is very hard to define generic approaches for such demands. Thus these parts of the GST are for the time being exclusively project driven eg. by implementing dedicated Web services or providing transformations and wrappers to use existing technologies.
Altogether there is no integrated toolkit which fully fits into the scope of process data processing. In the domain of data access OGSA–DAI offers a good groundwork for process data processing, some developments in other domains are predestined to be included into GST for specialized tasks. Nevertheless there are plenty problems to reach a status of intuitive access and expected transparency.

3 Architecture of GST

GST is based on WSRF–compliant Web services. This allows to build a framework consisting of independent Web services for each of the aimed fields of application. On the client side of the respective Web service a rich API allows the user to access all functionalities using standardized interfaces. All Web services can optionally be seamlessly combinable. Furthermore reasonable functionalities are shared between them.
These demands result in the architecture presented in figure 1. The GST consists of three major parts which have in common, that they offer an API with intuitive access apart from dealing with Web service clients or more Grid related tasks than needed. The access can be done exemplary by one of the available example–clients or by using the respective API.

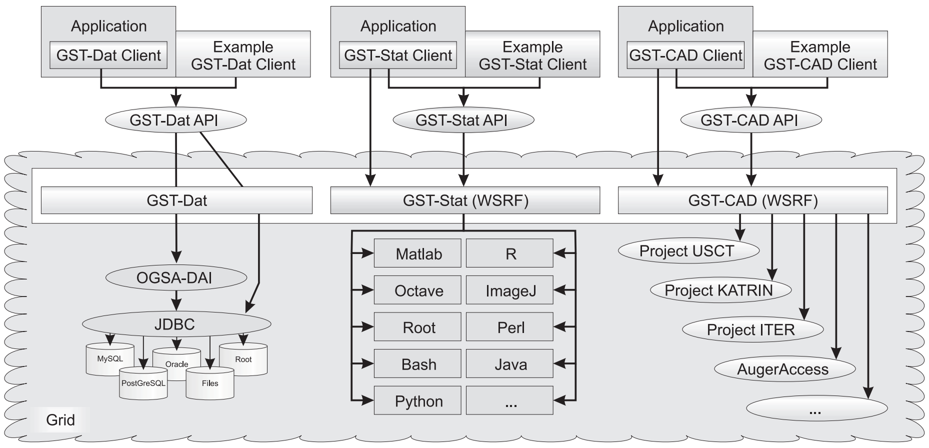

Figure 1: Overview on the Grid Services toolkit structure consisting of a rich client API (top) and the server–sided implementation of process data processing functionalities (bottom). Left) GST–Dat integrates JDBC and OGSA–DAI for accessing databases. Center) GST–Stat wraps various applications into interpreter services processing user provided scripts. Right) GST–CAD provides specialized services for various process data processing projects.

3.1 GST–Dat

GST–Dat is responsible for all tasks dealing with data–access and –transfer. In the scope of process data processing it is focussed on accessing structured data in the form of relational databases. The main task which is solved by GST–Dat is completely abstracting the data access by encapsulating SQL into an API. This enables the user to utilize similar functionalities of different database systems, which are partly implemented using a highly differing SQL syntax. The user does not have to care anymore about which database he is accessing. For the actual database access all functionalities are already provided by OGSA–DAI and direct JDBC access. For fast and large transfers JDBC is still the more efficient solution. OGSA–DAI is slower but offers additional features like security and transparency. Figure 1 shows that API calls of GST–Dat directly access OGSA–DAI respectively JDBC. A further encapsulation of OGSA–DAI and JDBC seems not be reasonable, because the interference while accessing data resources should be as small as possible.

3.2 GST–Stat

GST–Stat is responsible for solving data analysis tasks. Many process data processing projects are implemented using interpreters. Thus Grid access for these projects can be realized by enabling the execution of interpreter scripts on the Grid. Furthermore interpreters are, not only for process data processing, dedicated for rapid prototyping and they can be easily controlled by their standard–input stream. If they can be accessed on the Grid, no local installation is needed. Accessing arbitrary interpreters can be realized platform– and programming–language independent by wrapping them into a WSRF–compliant Web service as shown in figure 1. Thus any available interpreter can by called by the same interface. The integration of new interpreters must take place dynamically to avoid any interference with running interpreter instances. Also single instances should not interfere with each other. Special attention should lay on the handling of result data. It must be avoided that, especially private data, can be accessed by anyone else but the person who is allowed to.

3.3 GST–CAD

This part of the GST is mostly project–driven. CAD stands for Computer Aided Diagnosis and is responsible for solving project specific tasks. Depending on the projects needs GST–CAD can contain Web services which fulfill specialized and highly optimized tasks. Thus GST–CAD can currently not provide standardized interfaces. Maybe at a later date similarities become apparent and can be integrated into a uniform interface, at least for a couple of projects.

4 Implementation

4.1 Implementation of GST–Dat

For GST–Dat the interfaces accessing data resources are defined by JDBC and OGSA–DAI. The aimed transparency is implemented on the client side within the GST–Dat API. It allows transparent access to the database systems MySQL, PostgreSQL and Oracle by wrapping their specific SQL syntax into uniform Java objects. Thus accessing these databases can be realized without any knowledge about SQL syntax. The implemented data types are derived from an abstract class SQLObject. Basic data types eg. integer, double, varchar etc., having representations in all currently supported database systems, are already implemented. Special types like points or paths in PostgreSQL are not planned to be integrated in GST–Dat. Maybe they can be supported demand–driven in the form of extensions with the compromise of constrained transparency. The structuring of the data is realized by an abstract class SQLTable. Thus structure related differences between database systems remain hidden to the user. The execution of SQL–queries and –updates is done by using a database adapter implemented for each database system. Hence deriving these adapters from an abstract class DatabaseAdapter accessing the databases behind is fully transparent.

4.2 Implementation of GST–Stat

GST–Stat is mainly dedicated to the execution of scripts of arbitrary interpreters. As shown in figure 2 the entry point is the GST–Stat Factory Service.

This WSRF–compliant Web service manages the un–/registration of new interpreters, the management of GST–Stat projects and the instantiation of interpreter services, which are newly–created or loaded from the GST–Stat Project Store. Accessing GST–Stat can be realized by using the appropriate API which encapsulates all required Web service clients and hides the underlying Grid infrastructure.

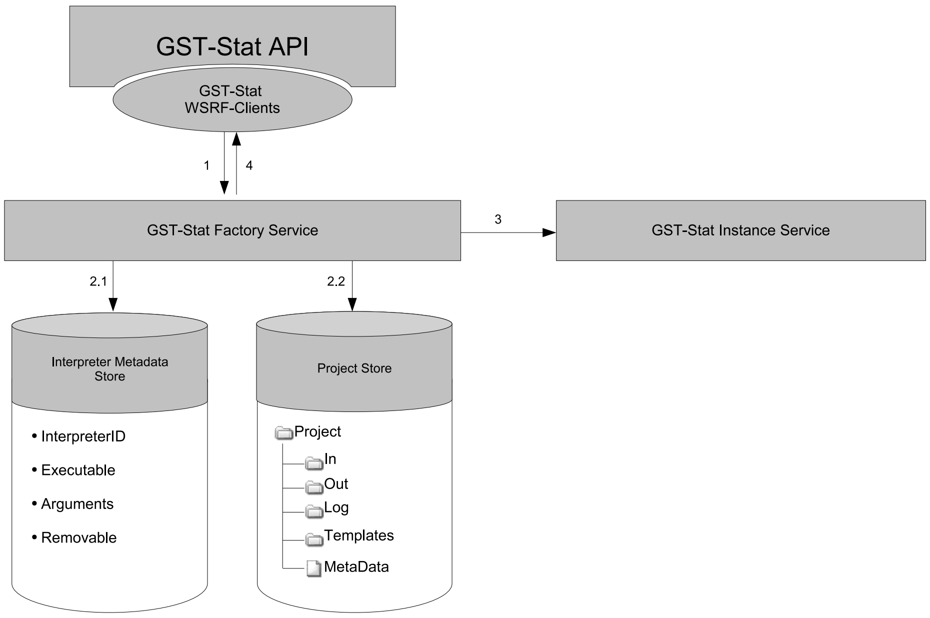

Figure 2: Procedure of creating a GST–Stat Instance Service. In the first step (1) the factory service is called by using the GST–Stat API wrapping the actual Web service client. This call can either create an instance of an interpreter registered at the Interpreter Metadata Store (2.1) or an instance is loaded from the Project Store (2.2) holding projects persistent in the file system. Finally the instance is created (3) and the endpoint–reference of the instantiated service is returned to the client (4).

The GST–Stat Instance Service is responsible for the actual interpreter execution. The instantiation of an Instance Service is done by the Factory Service. This gives the option to change the implementation of single components of the services without adapting the GST–Stat API, as long as the interfaces are unchanged. Also new interpreters can be registered at runtime without interfering already installed or running interpreters. GST–Stat is based on beneficial standards, eg. WS–Notification as part of the WSRF, as well as on own implementations to realize important functionalities eg. data streaming, which does not have standardized specifications yet. The implemented streaming functionality allows to stream data in different stages of security to access the interpreter streams directly.

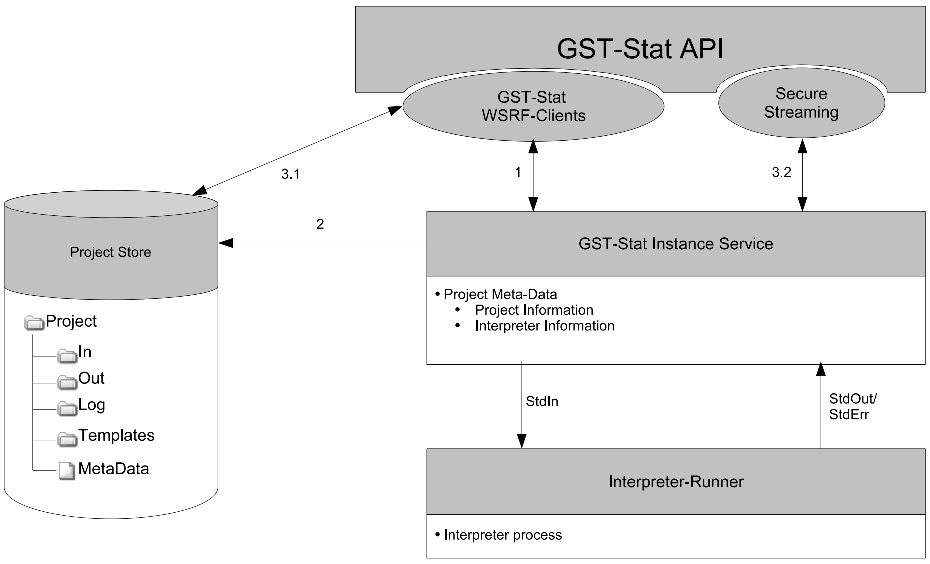

Figure 3: Process of running a script using the GST–Stat Instance Service. With the first call (1), executed by the GST–Stat API, the interpreter process is started within an instance of the Interpreter–Runner. Sequencing one or more scripts are transferred using the GST–Stat API, either by calling a Web service method or by streaming the input to the Instance Service. These scripts are forwarded to the Interpreter–Runner writing them into the standard–input of the interpreter process. All received output is returned to the GST–Stat Instance Service. The data is stored for later use into the according directories of the Project Store (2). From there the data can be obtained later using the Web service interface (3.1). Furthermore the user can decide to get the results directly streamed back, using the secure streaming functionality of the GST–Stat API (3.2).

4.3 Implementation of GST–CAD

GST–CAD allows the integration of arbitrary Web services. Frequently scientific projects require specific services, eg. for the reconstruction of USCT images. Thus many of these services might be only adequate for one single project. GST–CAD provides a platform for such services. To administrate the deployment of new services GST–CAD integrates a hot–deployment service provided by the University of Marburg [16]. This service extends the Globus server by a non–

intrusive deployment of new Web services. Web services can be deployed and re–/loaded dynamically without restarting the server, as it is typically necessary.

5 Results and Conclusions

Summarizing the GST integrates a lot of useful features for process data processing. GST–Dat currently allows transparent access to basic features of the database systems MySQL, PostgreSQL and Oracle. The next steps will be the extension by more complex database features eg. joins or large–scale select and update statements. Also additional support for database systems like DB2 or Microsoft SQL Server should be considered.
The implementation of GST–Stat is nearly finished. Arbitrary interpreters can be controlled remotely by submitting complete scripts or streaming single commands interactively. Open points are the integration of standards like GridFTP and to find a way how to guarantee privacy of project data. By default the WSRF offers mechanisms to secure the data transport by established standards like Transport Layer Security (TLS) and Grid Security Infrastructure (GSI) using public key cryptography [12]. The challenging task is the protection of the data located in the Project Store. Possibly there are developments which can be integrated into GST at a later date. Another solution would be the encryption of selective project data, which probably performs bad for large data sets.
For our projects we normally try to avoid depending on commercial projects by using their non–commercial pendants, if there is one available eg. Octave in the case of Matlab. Nevertheless commercial solutions normally provide additional features which ease the daily work. The portability between commercial and non–commercial versions is often hard to realize, so there should be sooner or later an integration of a licensing management, not just for the GST but for the complete field of Grid computing.
For GST–CAD there are currently implemented some services for signal processing. Other parts will follow depending on the needs of projects at the Forschungszentrum Karlsruhe.
GST is an easy–to–use and easy–to–extend collection of tools for process data processing, but can be also used for nearly all scientific projects demanding immediate access. Especially for process data processing projects it is crucial to keep delays between requesting an operation and starting its execution as small as possible. Several applications are in preparation eg. for USCT, Katrin and image processing. In the future it is planned to integrate the GST–APIs into several problem solving environments like Matlab or LabVIEW [17] and ImageJ [18] to allow an easy access for scientists to the Grid infrastructure.

References

1. Institute for Data Processing and Electronics, Forschungszentrum Karlsruhe, Germany. Ultrasound computer tomography. [Online]. Available: http://www.ipe.fzk.de/projekt/med/usct/e_index.html

2. "Matlab," Mathworks, Inc., Natick, MA, 1999.
3. (2007) Octave website. [Online]. Available: http://www.gnu.org/software/octave/
4. R. Brun and F. Rademakers, "ROOT — An Object Oriented Data Analysis Framework," in *AIHENP, NIM A389*, 9 1996, pp. 81–86.
5. T. Banks, "Web Services Resource Framework (WSRF) – primer v1.2," OASIS Open, Tech. Rep., 2006. [Online]. Available: http://www.oasis-open.org
6. I. Foster, "Globus Toolkit version 4: Software for service-oriented systems," in *IFIP International Conference on Network and Parallel Computing.* Springer-Verlag LNCS 3779, 2005, pp. 2–13.
7. 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.
8. 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.
9. A. Rajasekar, M. Wan, R. Moore, W. Schroeder, G. Kremenek, A. Jagatheesan, C. Cowart, B. Zhu, S. Chen, and R. Olschanowsky, "Storage resource broker — managing distributed data in a grid," *Computer Society of India Journal: Special Issue on SAN*, vol. 33, no. 4, pp. 42–54, 2005.
10. M. Antonioletti, M. Atkinson, R. Baxter, A. Borley, N. Chue Hong, B. Collins, N. Hardman, A. Hume, A. Knox, M. Jackson, A. Krause, S. Laws, J. Magowan, N. Paton, D. Pearson, T. Sugden, P. Watson, and M. Westhead, *Concurrency and Computation: Practice and Experience.* John Wiley & Sons, Ltd., 2005.
11. Sun Developer Network. JDBC technology. [Online]. Available: http://java.sun.com/products/jdbc/
12. V. Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Czajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman, and S. Tuecke, "Security for grid services," in *Twelfth International Symposium on High Performance Distributed Computing (HPDC–12).* IEEE Press., 2003.
13. V. Welch *et al.*, "X.509 Proxy Certificates for Dynamic Delegation," 3rd Annual PKI R&D Workshop, 2004.
14. G. Allen, T. Dramlitsch, I. Foster, *et al.*, "Supporting efficient execution in heterogeneous distributed computing environments with cactus and globus," in *In Proceedings of Supercomputing*, 2001.
15. D. Talia, P. Trunfio, and O. Verta, "Weka4WS: A WSRF–enabled weka toolkit for distributed data mining on grids," in *Proc. of the 9th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD 2005)*, vol. 3721. Springer–Verlag, 2005, pp. 309–320.
16. The Distributed Systems Group - University of Marburg, Germany. The marburg ad-hoc grid environment. [Online]. Available: http://mage.uni-marburg.de/
17. National Instruments, "LabVIEW – powerful graphical intuitive development," LabVIEW Web Site, http://www.ni.com/labview/, 2006.
18. W. Rasband. (1997–2006) ImageJ, U. S. National Institutes of Health, Bethesda, Maryland, USA. [Online]. Available: http://rsb.info.nih.gov/ij/