GES 2007 German e-Science

Available online at http://www.ges2007.de

GRAIL – Grid Access and Instrumentation Tool

T. Jejkal[1], R. Stotzka[1], M. Sutter[1] and H. Gemmeke[1]

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

Abstract

Since the release of Globus Toolkit 4 Web services enrich the world of Grid Computing. They provide methods to develop modular Grid applications which can be parallelized easily. The access to Web services is mostly solved by complex command line tools which need a good deal of knowledge of the underlaying Grid technologies. GRAIL is intended to fill the gap between existing Grid access methods and both the developer who wants to utilize the Grid for own developments and the user who wants to access the Grid without much additional knowledge. It simplifies the access and the testing of Web services for the Globus Grid middleware. GRAIL provides an easy to use graphical user interface for executing Web services and enables the user to construct complex relationships between services to realize parallel execution. The underlying framework allows an easy integration of any Web service or other arbitrary task without much additional effort for the developer. Existing technologies, shipped with the Globus Toolkit, are seamlessly integrated into GRAIL.

1 Introduction

The development of WSRF–compliant Web services [1] for the Globus Toolkit [2] is, even for experienced developers, a challenging and error–prone task. Several tools exist simplifying the general development process to a minimum effort eg. the Grid Development Tools for Eclipse [3] or Introduce [4]. The general procedure of using these tools is nearly identical:

- Defining interfaces for accessing the resulting Web service,
- automatic generation of the Web service WSDL document, client– and server–stubs and a simple command line client for testing purpose,
- implementing the functionalities of the Web service,
- packaging of the Web service and deployment into a Web service container and
- testing the functionalities by using the generated command line client.

As well as for every software development project also for Web services the testing is an important part of the development process, but for Web services the iteration loops contain additional steps. At least the developer additionally has to re–deploy the service followed by running different test cases for a

single service. The command line client might be feasible for simple tests after extending it by implementing interactions or graphical reassessments. If there are dependencies between the current and an other service the other service client has to be integrated to allow testing their cooperation. In most cases the developer writes scripts for automated testing, test classes or graphical user interfaces in which parameters are adjusted dynamically. Nevertheless these steps are highly recurring. Thus a tool which allows to realize complex testing and intuitive access to Web services would ease the developers work and accelerate the development process.

2 Objectives of GRAIL

The Grid Access and Instrumentation Tool (GRAIL) is a dedicated environment to allow intuitive and easy access to the Globus middleware. GRAIL is primarily intended to provide a uniform desktop–like interface to handle the following functionalities:

- Easy execution and management of Web services eg. WS–GRAM [5] or other, arbitrary Web services,
- seamless integration of default components (eg. security tasks like proxy–init, GridFTP [6], MDS–4 [7],
- easy extensibility without requiring much knowledge about the Grid or Web service development,
- task graphs for building up dependencies between eg. instances of different Web services,
- support for offline–tasks which have to be done before, after or instead using a Grid functionality eg. reading parameter files, building a job description for WS–GRAM, viewing result data in different formats or transforming data.

By using these functionalities, recurring tasks can be integrated and accessed easily at any time. Thus the development of Web services is considerably accelerated due to iteration loops are shortened.

This paper is structured as follows: Section 2 deals with the state of the art concerning possible solutions for the described demands; section 3 describes the architecture of GRAIL followed by conclusions and an outline concerning the future work on GRAIL in section 4.

3 Related Work

Due to the fact that accessing Web services of the Globus middleware is quite complicated. Thus a tool to simplify this access is needed. For traditional, job–based Grid architectures there are a lot of projects trying to simplify the access by graphical user interfaces, but none of them is partly or completely dedicated to Web services or service oriented architectures as provided by the Globus Toolkit. It has to be analyzed if any of the existing tools can be adopted to access such architectures in a comfortable manner.

3.1 Grid Desktop

The Grid Desktop is part of the Commodity Grid Kits (Cog Kits) [8] developed within the Globus Alliance. It addresses the Grid novice with an easy–to–use graphical interface. The Grid Desktop allows the user to execute simple jobs and to use GridFTP; it provides a Grid file browser and an integration of a tool to execute Karajan Workflows [9], which are also developed in the context of the Cog Kits. The idea of the Karajan Workflow engine is to use every kind of task e.g. Web services to orchestrate highly concurrent workflows. Currently the software and the documentation of the Grid Desktop as well as the Karajan Workflow engine are in early development state.

3.2 GridSphere

GridSphere [10] is developed by GridLab to provide a portlet based portal framework. It offers a generic environment for developing just about any Web application. It enables developers to plug in access methods to Grid technologies as favored by their user communities. Thus GridSphere can be placed directly on top of an existing Grid infrastructure or on top of Grid abstraction layers e.g. the Grid Application Toolkit (GAT) [11]. For the user GridSphere portlets are accessible via Web interface by any Web browser. The middleware or application behind is completely hidden from the user. GridSphere provides a set of basic portlets for security tasks, job submission and file management. All of these portlets are integrated within the portal and visualized by a portlet engine using a uniform graphical representation. GridSphere supports single sign–on authorization, where the user accesses the functionality after logging in on the portal Web page once. Due to the centralization of authentication, roles can be defined for accessing the infrastructure inside. The advantage of this approach is that the user does not have to deal with software installation. The Web front–end and the functionality inside can be designed community specific. One disadvantage is, that GridSphere provides a server sided access to the Grid. That means that all functionalities must be wrapped by a portlet. Realizing a portlet which dynamically accesses Web services and which allows communication in between, seems to be very hard to realize.

3.3 g–Eclipse

G–Eclipse [12] is a running EU–project with the intention to integrate an access–point to the Grid into the development environment Eclipse [13]. The target audience are Grid users as well as Grid developers and operators with the goal to hide complexity of the Grid [14]. The plug–in–based architecture allows users familiar with Eclipse and Java programming to extend the framework by own plug–ins. The concept of g–Eclipse is mainly based on conventional Grid environments which are job–based and aims the integration of middleware access into the programming environment. Service oriented approaches based on Globus Toolkit 4 are at the moment not supported.

All audited tools have in common that they support the user accessing available functionalities of the Grid eg. to submit conventional Grid jobs. They are not intended to support developers of integrated Grid functionalities like Web services. Thus these tools seem to be too static for adapting them to support developers working on service oriented approaches.

4 Architecture of GRAIL

GRAIL provides a uniform graphical interface which offers the user a consistent view onto the Grid to enable intuitive access to the Globus middleware. It is easily extensible by new functionalities to support service oriented architectures based on WSRF–compliant Web services. To allow assembling complex task graphs GRAIL also supports the local execution of tasks which are not enabled or feasible for running on the Grid.

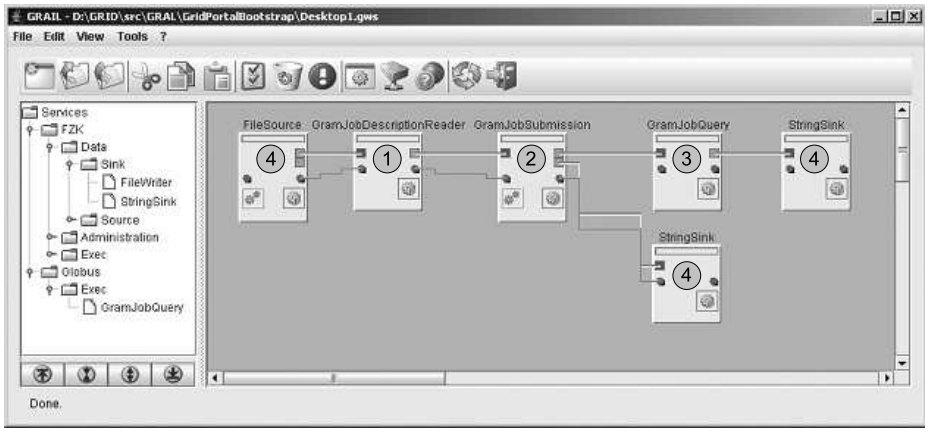

Figure 1: Graphical interface of GRAIL. Left: Component tree from which components can be simply dragged to the main desktop. Right: Main desktop which shows a task graph for loading a job description from a file (1), submitting the job to the Grid (2) and monitoring its execution (3). Furthermore GRAIL contains a large set of common helper components (4). Every component has a number of data connectors (square connectors) and one trigger connector (round connectors), in each case for triggering and being triggered. The two buttons in the lower part of each component are responsible for showing the individual parametrization (left) and to run the component manually (right).

4.1 Usability

GRAIL presents itself as a desktop–like user interface as shown in figure 1. During GRAILs startup a comfortable feature is introduced by the single sign–on functionality which allows the user to enter his certificates password during the first startup. Thenceforward the user does not have to authorize himself again unless he destroys his proxy–certificate. To allow an immediate utilization of GRAIL there is a set of basic components included:

- Helper components eg. for file I/O, viewers for different data formats, constants of different types, components for structuring task graphs,
- components for supporting the access to Web services provided by Globus eg. a job description builder for WS–GRAM, components for job submission and monitoring,
- full integration of in–house developments eg. the Grid Services Toolkit for Process Data Processing [15], which contains Web services for transparent data access and utilization of arbitrary interpreters on the Grid.

Furthermore GRAIL offers different wizards eg. for creating a Web service client from a WSDL [16], optionally directly followed by the creation of an according GRAIL component for a selectable Web service method.
The assembling of task graphs happens by dragging components from the component database tree to the desktop, connecting input–, output– and signal–connectors and starting one or more components, depending on relationships between graph branches.
Figure 1 shows an example of using GRAIL for job submission with WS–GRAM. For describing a job the user needs a job description. This description is loaded from a file by using the according component. Following the job description is passed to the next component which is responsible for submitting WS–GRAM jobs to a target URL. For obtaining the target URL a MDS–Browser is integrated into GRAIL. It offers the user an easy way querying available resources. After doing so the job can be submitted. For monitoring the job GRAIL provides a third component which can be used optionally in combination with the job submission component. It requests the current status of the running job periodically from the server based on the job handle obtained from the submitting component. If the job has completed the user is notified graphically or by an according message. Finally the user has to obtain results of the job, eg. output files. If there is no file–staging used for downloading results automatically GRAIL offers the user an integrated GridFTP–Browser for accessing the Grid file system for downloading the results. All these functionalities are integrated into the comfortably operated user interface which supports features like drag&drop, a component browser and the ability to store task graphs for later re–use.

4.2 Component architecture

Every component of GRAIL is defined as an independent piece of software which provides a single functionality. One component has input– and output–connectors to parameterize its execution and to access its results respectively.

The values of the connectors are obtained from or used as parameters by other components. Furthermore a triggering mechanism is integrated for every component to trigger the execution of other components for building up task graphs.

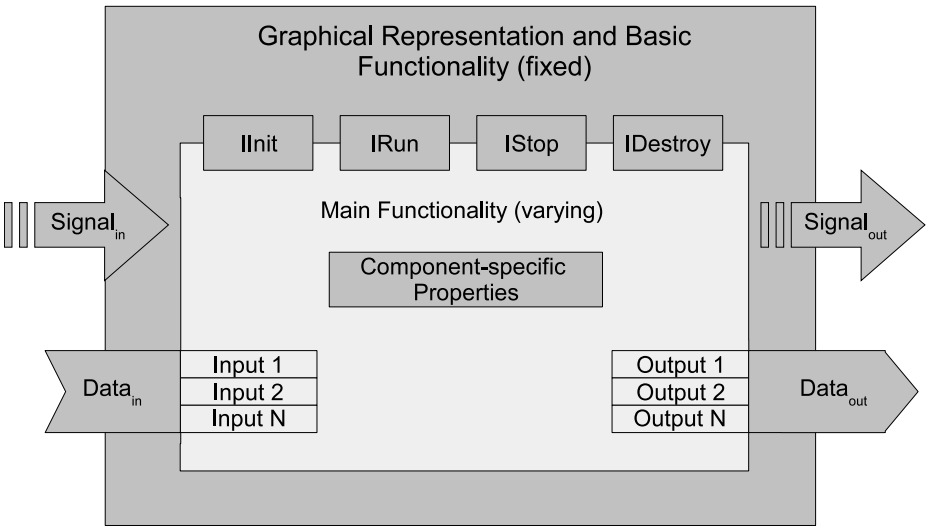

Figure 2: Architecture of a GRAIL component. It shows the general structure on which every component is based. The outer block is responsible for graphical representation and basic functionality like drag&drop. Trigger signals are handled by each component without any effort for the developer. Providing arbitrary data types is realized by defining the number of served connectors and associating each connector with a unique identifier. The actual component code is automatically generated. Finally the inner part, accessible by four interfaces (IInit, IRun, IStop and IDestroy), is implemented by the component developer using any favored development environment.

Figure 2 shows the architecture of a single GRAIL component. The graphical representations and the core functionalities eg. processing of user interactions and triggers are similar for every component. A component for solving a specific problem is derived from this abstract component definition by implementing the four interfaces. These interfaces are responsible for building a bridge between user interaction and component execution:

IInit The implementation of IInit is responsible for initialization tasks which are executed before calling IRun eg. connecting to a Web service.

IRun IRun contains the actual functionality of the component. It can contain everything which may be implemented in Java eg. the execution of a Web service client, locally running tasks or even the forking of a third–party tool, written in any arbitrary programming language.

IStop The interface IStop is called if the execution of the component was aborted by the user. By implementing this interface the developer can realize eg. error handling or cleanup tasks.

IDestroy This interface is called if the user has removed the component from the GRAIL desktop. It allows the developer to implement a sane destruction of instantiated objects.

In the domain of data handling there are no restrictions which data types or how much input– or output–connectors can be used. Adequate access methods are generated automatically during the creation of a new component by specifying desired connectors.
Due to the fact that complex algorithms may need plenty of parameters an additional way to setup a components execution is introduced by the ability to define component–specific properties, as seen in figure 2. The representation of these properties is defined very coarse to allow complex property dialogs as well as simple parametrization by using a single file. Therefor GRAIL offers an interface which allows to trigger loading and storing properties in a generic format as well as in a format which can be chosen by the components developer to allow the re–use of existing parameter files.

4.3 Portability

To allow easy publishing and distribution of components a XML format for describing components is defined. This format allows a detailed description of a component which is utilized for user documentation as well as for searching within GRAILs component database. By using the component search feature the user is able to find needed components easily, even if the component database grows very large.
Independently accessible functionalities which provide complex and dedicated user interfaces eg. the integrated MDS– or the GridFTP–Browser are realized as plug–ins. These special software components are intended to be used to integrate features, which can be utilized at any time, independent from the actual task graph execution. As well as for components there is a description format for plug–ins which allows easy portability.

5 Results and Conclusions

GRAIL offers intuitive and, also for a novice, comprehensible access to the Grid. The clear desktop–like interface enables a fast access to all functionalities of GRAIL and allows a familiar work with it. By offering basic functionalities eg. the integration of basic components and features like the MDS– or GridFTP–Browser GRAIL can be utilized immediately. In the scope of usability GRAIL offers handy features eg. single sign–on, drag&drop and task graphs. Extending GRAIL by new components is supported by wizards which generate either component skeletons for implementing own functionalities or completely functional components from a provided Web services WSDL. The Web service developer

benefits from GRAIL by getting a tools which allows to test Grid functionalities eg. Web services easily. Dependencies between different functionalities can be reflected by connecting components realizing according calls. Furthermore the developer can easily automate and parallelize the testing of Web services by creating and executing a task graph which contains different scenarios at once. Thus development cycles can be shortened by rapid testing without implementing a testing framework for a growing number of different functionalities.
For the future various extensions to GRAIL are planned. The probably most attractive feature is to enable GRAIL to execute task graphs completely or partly on the Grid. In this scope a headless mode of GRAIL is planned to execute task graphs without showing the user interface. Also a component compiler generating one single component from an arbitrary task graph might be a convenient feature. Furthermore additional components are planned to be provided within the default component collection to enhance the usability of GRAIL. Altogether with GRAIL a handy tool is available which allows its users to access not only the Grid in an intuitive and comfortable manner, but also to bring Grid functionalities in relation to each other and to locally running algorithms and tools. At the moment the work on the completion of the first GRAIL release is nearly finished and it will be available for public download under [17] at the second quarter of 2007.

References

1. T. Banks, "Web Services Resource Framework (WSRF) – primer v1.2," OASIS Open, Tech. Rep., 2006. [Online]. Available: http://www.oasis-open.org
2. The Globus Alliance. The Globus Toolkit homepage. [Online]. Available: http://globus.org/toolkit
3. The Distributed Systems Group - University of Marburg, Germany. Grid Development Tools for Eclipse. [Online]. Available: http://mage.uni-marburg.de/trac/gdt/
4. caGrid. Introduce - Grid Service Authoring Toolkit. [Online]. Available: http://www.cagrid.org/mwiki/index.php?title=Introduce
5. Globus Toolkit Team. GT 4.0 WS GRAM. [Online]. Available: http://www.globus.org/toolkit/docs/4.0/execution/wsgram/
6. 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.
7. 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*, vol. 46, pp. 521–525, 2006. [Online]. Available: http://stacks.iop.org/1742-6596/46/521
8. Argonne National Laboratory. Java CoG Kit. [Online]. Available: http://wiki.cogkit.org/index.php/Main_Page
9. G. von Laszewski and M. Hategan. Java CoG Kit Workflow Guide. [Online]. Available: http://wiki.cogkit.org/index.php/Java_CoG_Kit_Workflow_Guide
10. J. Novotny, M. Russell and O. Wehrens. GridSphere Portal Framework. [Online]. Available: http://www.gridsphere.org/gridsphere/gridsphere

11. H. Kaiser, K. Davis, T. Goodale and A. Merzky. Grid(Lab) Application Toolkit. [Online]. Available: http://www.gridlab.org/WorkPackages/wp-1/
12. H. Kornmayer. g–Eclipse. [Online]. Available: http://www.geclipse.org
13. The Eclipse Foundation. Eclipse – an open development platform. [Online]. Available: http://www.eclipse.org
14. H. Kornmayer. Deliverable 1.2 – Architecture. [Online]. Available: http://www.geclipse.org/fileadmin/Documents/Deliverables/D1.2-Architecture-I.pdf
15. 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.
16. World Wide Web Consortium. Web services description language (wsdl) 1.1. [Online]. Available: http://www.w3.org/TR/wsdl
17. T. Jejkal. GRAIL – Grid Access and Instrumentation Tool. [Online]. Available: http://fuzzy.fzk.de/~GRID/Grail/