GRAIL – A Tool for Accessing and Instrumenting WSRF–compliant Web Services

Thomas Jejkal, Rainer Stotzka, Michael Sutter
Institute for Data Processing and Electronics
Forschungszentrum Karlsruhe GmbH
Hermann-von-Helmholtz-Platz 1, D-76344 Eggenstein-Leopoldshafen, Germany
thomas.jejkal@ipe.fzk.de

Abstract

Grid access methods are still dominated by command line tools or manually developed, problem specific graphical user interfaces. This fact reduces the acceptance of the Grid for users, who are not familiar with the usage of the Linux shell or the Grid itself. The first step away from the command line was done by the Globus Alliance by interpreting and realizing the Grid as a Service Oriented Architecture (SOA). The access to this SOA is defined platform and language independent. Nevertheless the access is still realized by command line tools. Only some of these tools are platform independent and the usage is still complicated. So the abilities of a SOA are not applied consequently. GRAIL is a tool to simplify the access to Web services of the Globus Grid middleware. It provides an easy to use framework for executing Web services and to construct complex relationships between independent services. The underlying framework allows an easy integration of new Web services or generic tasks. GRAIL is intended to be used by developers of Web services for rapid testing as well as by users of the Globus based Grid for simple and flexible access. This paper describes the structure of GRAIL and the benefits from using it.

1. Introduction

Web services are platform and language independent software components which are accessed via the SOAP protocol. They provide the ability to create modular applications for solving problems in parallel, e.g. on a Grid. The Globus Toolkit 4 [1] is based on Web services and allows to build up Computing Grids as a Service Oriented Architecture (SOA) [2]. Web services in Globus are compliant to the Web Services Resource Framework (WSRF) [3]. Thus their behavior is different from common Web services since WSRF–compliant Web services have to be instantiated and accessed through a unique endpoint reference. The development of Web services for a Grid middleware like Globus Toolkit is, even for experienced developers, a challenging and error–prone task. There are tools to simplify the general development process to a minimum effort [4], [5]. For immediate access these tools generate rudimental service clients during the build process. Of course, Web services are also not a new technology and there are several testing tools for common Web services [6], [7].

Challenging and new aspects arise while testing and using Web services focussed on Service Oriented Architectures and the Grid for performing:

- security tests;
- scalability tests;
- tests which contain relationships between several services;
- tests with large argument sets.

For service developers there is the need of a tool which provides generic interfaces to realize easy and rapid testing concerning these new challenges.

The second target group of GRAIL are the Grid users. The acceptance of the Grid is intimately connected to the usability of its access methods. Interfaces, shipped out with the middleware, are mostly realized as command line tools or problem specific graphical user interfaces. The following scenario points out the difficulties in integrating Web services of the Globus Toolkit: A user wants to solve a complex calculation in parallel. The input data is obtained from a relational database using OGSA–DAI [8], the actual execution is realized using the Grid Resource Allocation and Management (GRAM) service of the Globus Toolkit [9]. Prior to the execution on the Grid the user has to deal with Grid security, the OGSA–DAI programming interface, the
GRAM job description language and with the GRAM command line client. Afterwards the user has to execute three independent applications, at first the grid-proxy-init to allow secure access to the Grid, then the data preprocessing via OGSA-DAI and finally the GRAM job submission. Of course, it is possible to integrate all these tasks into one specific application, but a more general solution is desired. The situation gets even worse if more than three independent parts, which change dynamically depending on particular results or arbitrary border conditions, are involved.

With every additional part of the application the effort of maintaining the specific application increases. All these facts minimize the acceptance of using the Grid in a service oriented manner. The motivation behind GRAIL for the user is to provide easy and intuitive access to a Service Oriented Architecture as described above. Generic interfaces must allow an easy extensibility and comprehensive possible applications.

This paper is structured as follows: Section 2 deals with the state of the art concerning possible solutions of the described problems; section 3 describes GRAIL and its architecture; in section 4 the results and benefits are discussed, followed by a conclusion.

2. Related work

Due to the fact that the access and the instrumentation of Web services of the Globus middleware is quite complicated for an inexperienced user, who does not want to deal with complex access methods, a tool to simplify these tasks is needed. There are some activities to realize tools that solve the mentioned problems. On the one hand existing tools are adapted to work with Grid infrastructures, on the other hand new developments, dedicated to the daily work with the Grid, are developed. These two kinds of tools are discussed in this chapter.

2.1. Migrating Desktop

The Migrating Desktop (MD) [10] was developed by the EU CrossGrid project [11]. The intention of the MD is to provide a user friendly front end to a Remote Access Server (RAS) providing access to Grid resources wrapped by common Web services. The RAS offers a well-defined set of Web services which can be used as an interface for accessing high performance computing systems and services in a common, standardized way. Due to the definition of interfaces for this service collection, the Grid user is able to create plug-ins which call the desired Web service using problem specific parameters. One of the existing services provides a job submission interface for using the CrossGrid/DataGrid resource broker. The parametrization of the service call is realized by using a problem oriented graphical wizard. The simplification of this approach is the limitation of development effort to the creation of the mentioned wizard. The developer needs no knowledge about the underlying resource broker and its job description language. However, this approach contains a lot of constraints. All features depend on the existence of a corresponding Web service on the RAS, making it quite inflexible in terms of a SOA realized by the Globus Toolkit. If a special feature is required on the RAS, the developer has to implement a new Web service which encapsulates the application he has implemented before to solve a specific problem. Summarizing, the Migrating Desktop is a tool fulfilling its task to be a user friendly interface to LCG [12] and gLite [13], but it is too inflexible to adopt it for a Globus Grid environment.

2.2. Grid Desktop

The Grid Desktop is part of the Commodity Grid Kits (Cog Kits) [14] developed by the Globus Alliance. It addresses the Grid novice with an easy to use graphical interface similar to the Migrating Desktop. The Grid Desktop allows the user to execute jobs and to use GridFTP, it provides a Grid file browser and a tool to execute Karajan Workflows [15], which are also developed as part 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. Due to the fact, that software and documentation of the Grid Desktop and the Karajan Workflow Engine are in early development state, this paper can not discuss the usability or features of these tools. The Grid Desktop, shipped out with the Cog Kit 4.1.5, is a collection of tools for accessing the Globus middleware. The correct combination of these tools, which is needed to utilize the Globus middleware in a feasible manner, is left to the user.

2.3. GridSphere

GridSphere is developed by GridLab to provide a portal based portal framework [16]. GridSphere is, in fact, a generic environment for developing just about any Web application. It allows developers to plug in their own Grid technologies required by their user communities. So GridSphere can be placed directly on top of an existing Grid infrastructure or Grid abstraction layer e.g. the Grid Application Toolkit (GAT) [17]. For the user GridSphere portlets are accessible via a Web interface using an arbitrary Web browser [16]. 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 into the portal and visualized by a portlet engine. GridSphere supports a single sign-on authorization, to access the functionality after logging in at the portal Web page. This mecha-
nism allows the creation of virtual organizations, depending on the login name. Different user groups might have different portlets enabled, depending on their role within a virtual organization. The user does not have to deal with software installation or libraries and the command line. The Web front end and the functionality can be designed community specific.

GridSphere provides a server sided access to the Grid. That means that all functionalities have to be integrated into a portlet. A realization of a portlet, which dynamically accesses Web services and which has the ability to realize data exchange between them, is a very challenging task. Problem specific extensions to the functionality or on-the-fly modifications are almost impossible to realize.

2.4. JOpera

JOpera is a project of the Department of Computer Science at the Swiss Federal Institute of Technology [18]. JOpera is realized as plug-in for the Eclipse Integrated Development Environment (IDE) [19]. The aim of this project is to combine a visual programming environment for Web services with a flexible execution engine capable of interacting with Web services through the SOAP protocol [18]. However, JOpera is not strictly limited to common Web services. The user is also allowed to execute e.g. Java code and scripts, submitted to the the Condor scheduler [20]. JOpera provides a workflow engine to describe workflows by an proprietary description language, the JOpera Visual Composition Language (JVCL) [21], or by the integration of other workflow description language standards e.g. BPEL [22]. The user is supported by different graphical views of the defined workflow e.g. the data flow, which visualizes the data transferred between the different components and the control flow showing the communication between the involved processes. The integrated access to WSRF-compliant Web services seems to be quite limited and there is no build-in support for the secure access to the Grid. Due to the fact on complex workflows, JOpera is partly too abstract to give the novice an immediate entry to the Grid.

2.5. Kepler

Kepler is a tool to create and execute scientific workflows which can contain access to emerging Grid based approaches to distributed computation [23]. It is build on top of the Ptolemy II framework, developed by the UC Berkley, which is responsible for modeling and executing workflows [23]. Kepler focuses on actor-oriented design. Actors are reusable components that execute and communicate with other actors in a scientific workflow via input and output channels. Kepler inherits Vergil, Ptolemy II’s graphical user interface [24]. The Vergil interface is an intuitive front end for designing, prototyping and executing scientific workflows. The workflows are described using an proprietary language called Modeling Markup Language (MoML) [23]. Due to the XML format of the MoML, workflows can be shared between different users. Developers can easily add functionality to the system according to the users needs. The user can extend Kepler by accessing Web services by their WSDL document. Kepler is a comprehensive tool with a lot of integrated components for creating complex scientific workflows, also for the Globus middleware. However, it is still focused on workflows which make the realization of user friendly Grid access quite difficult.

2.6. g–Eclipse

The g–Eclipse project aims to build an integrated workbench framework to access the power of existing Grid infrastructures [25]. The framework is fully integrated into the Eclipse development environment and adopts parts of e.g. the Migrating Desktop [10] and the GridBench suite [26]. The primary focus of g–Eclipse is on traditional, job based Grid environments. Thus, the integration of Web services does not make much sense for g–Eclipse at the moment.

3. GRAIL

Some of the tools, mentioned in chapter 2, offer useful functionality for accessing job-oriented Grid infrastructures, some of them allow easy access to Web services, but none of them seems to be fully applicable to the Service Oriented Architecture, described in the introduction. GRAIL is a tool providing a uniform graphical user interface to access the Grid in a job–oriented as well as in a Web service–oriented manner. Easy extensibility is guaranteed by using a component based design. The functionality of GRAIL can be easily extended to support the dynamics of a Service Oriented Architecture. GRAIL is focused on the Globus middleware and on WSRF–compliant Web services, but is not limited to a single middleware. The architecture of GRAIL is described in the following sections.

3.1. Usability

GRAIL presents itself as a desktop–like user interface (figure 1).

During GRAIL’s startup a comfortable feature is introduced by the single sign–on functionality allowing the user to enter his certificates password during the first startup. Afterwards the user does not have to authorize himself again unless he destroys his credentials. To allow an immediate utilization of GRAIL a set of basic components is included:
Figure 1. Graphical interface of GRAIL. Left: Component tree from which components can be simply dragged to the main desktop. Right: Main desktop showing 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 two trigger connectors (round connectors), one for triggering and another for 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).

- Helper components e.g. for file I/O, viewers for different data formats, constants of different types, components for structuring task graphs,
- components to support access to standard Web services provided by Globus e.g. a job description builder for WS–GRAM, components for job submission and job monitoring,
- full integration of in–house developments e.g. the Grid Services Toolkit for Process Data Processing [27], which contains Web services for transparent data access and utilization of arbitrary interpreters on the Grid.

Furthermore GRAIL offers different wizards, e.g. for creating a Web service client from a WSDL [28], optionally directly followed by the creation of a corresponding GRAIL component for one or more selectable Web service operations. The assembling of task graphs is realized 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. The job description is loaded from a file by using the FileReader component. Following the job description is passed to the GramJobSubmission component which is responsible for submitting WS–GRAM jobs to a target URL. The URLs of accessible GRAM services can be obtained using the integrated MDS–Browser. This Browser offers the user an easy query of MDS hierarchies for registered services. For monitoring the job GRAIL provides the GramJobQuery 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 job submission component. If the job is completed the user is notified graphically or by an according message. Finally the user obtains the results of the job, e.g. output files. For downloading job output files GRAIL offers the user an integrated GridFTP–Browser for accessing the Grid file system. All these functionalities are integrated into the user interface which supports features like drag&drop, a component browser and the ability to store task graphs for later re–use.
3.2 Component architecture

Every component of GRAIL is defined as an independent module providing one enclosed functionality. Each 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 each component to trigger the execution of other components.

Figure 2. Basic architecture of an abstract GRAIL component. The outer block is responsible for graphical representation and basic functionality like drag&drop. Trigger signals are handled by each component automatically. Providing arbitrary data types is realized by defining the number of 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), has to be implemented by the component developer using any favored development environment.

Figure 2 shows the architecture of an abstract GRAIL component. The graphical representations and the core functionalities, e.g. processing of user interactions and triggers, are identical for each 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 e.g. creating a Web service resource.

IRun IRun provides the actual functionality of the component. It contains arbitrary Java code, e.g. the execution of a Web service client, locally running tasks or even the forking of a third–party tool, written in an 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 e.g. error handling or cleanup tasks.

IDestroy This interface is called if the user removes 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 about data types or how many input– or output–connectors can be used. Adequate access methods are generated automatically during the creation of a new component by specifying the number and the type of required connectors. Due to the fact that complex algorithms may need plenty of parameters, an alternative way to setup for components execution is introduced by the definition of 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. For this purpose 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. Results and Conclusions

GRAIL offers intuitive and, also for a novice, comprehensible access to the Grid. The clear desktop–like interface allows a fast access to all functionalities of GRAIL and a familiar work with it. By offering basic functionalities, e.g. the integration of basic components and features like the MDS– or the GridFTP–Browser, GRAIL can be utilized immediately. In the scope of usability GRAIL offers handy features, e.g. 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 fully functional components from a provided Web services WSDL. The Web service developer benefits from GRAIL by getting a tools which allows to test WSRF–compliant Web services easily. Dependencies between different services 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 task graphs containing 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 an interesting task is the distribution of the execution of GRAIL components itself. Computationally intensive components are moved from the users PC to the Grid, are executed and results are directly used by dependent components. An integration into one of the tools, mentioned in chapter 2, is not planned at the moment due to the expected complexity, the focus on job–oriented Grids or the early state of development.

Now it has to be evaluated, if GRAIL fulfills the needs of a broad community: The developer who may test new Web services rapidly and within a uniform environment; the Grid novice who can easily access the Grid; finally the scientist who wants to solve complex problems without much additional effort. Currently GRAIL is available as a beta version and is tested internally to check which additional features or components could be integrated into a first release to improve the usability.

It was shown that there are a lot of ambitions to integrate the access to WSRF–compliant Web services into existing tools, as well as to develop dedicated tools for this purpose. Nevertheless the additional effort is, depending on the chosen tool, quite considerable. GRAIL automates as much as possible to minimize this additional effort. At the moment it provides support for the basic functionalities while working with the Globus middleware. It offers a flexible groundwork for future extensions.

References


