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 Elektronics, Forschungszentrum Karlsruhe GmbH, Hermann-von-Helmholtz-Platz 1, 76344, Eggenstein-Leopoldshafen, Germany *email:* Thomas.Jejkal@ipe.fzk.de
phone: (+49 07247) 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 easily parallelized. GRAIL is a tool to simplify the access and the testing of Web Services for the Globus Grid middleware, to provide an easy to use framework for executing Web Services and to construct complex relationships between independent services to realize parallel execution. The underlying framework allows an easy integration of any Web Service or offline task without much additional effort for the developer. Existing technologies, shipped out with the Globus Toolkit, are seamlessly integrated into GRAIL. The following abstract describes the objectives of GRAIL and the benefit for developers.

1 Introduction

The development of Web Services for a Grid middleware like the 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. Nevertheless there are open questions after the main development work. How the new Web Service can be easily tested? In which way dependencies between services on the client side can be realized without many programming effort? The following scenario should point out the difficulties while dealing with these questions: A developer creates a Web Service using existing tools like the Grid Development Tools for Eclipse [1]. His work is reduced to the definition of interfaces and the actual implementation of the service functionality. For testing purpose a command line client is generated together with the Web Service code. The next steps are either creating a complex API for wrapping the simple client or creating a dedicated user interface to access solely one single service. Both approaches have in common that they imply a lot of work which will occur again for every Web Service. The complex API must be tested and validated against dependent services and libraries. The complexity can often be hardly reflected by command line interfaces. In most cases the developer will write scripts for automatized testing, test classes or graphical user interfaces where needed parameters can be adjusted dynamically. Dedicated graphical user interface for stand-alone services have the disadvantage that they have a lot in common but

they are often realized as single applications. It is unattractive to users to get a handful of single applications which can be used to access the same middleware. They want an environment with which they feel familiar and from which they can access every functionality of the Grid. These problems are addressed by some other projects. One of them is the Java CoG Kit Desktop [2]. This project wants to hide technical details of an underlaying Grid environment by offering a familiar desktop like user interface to access the Globus middleware. The currently available prototype implementation provides access to the Globus Resource Allocation Manager (GRAM) and GridFTP. Due to the prototypical implementation it is not possible to say what additional features will be integrated into the CoG Kit Desktop and if it would be feasible for a Web Services based architecture like Globus. Another comparable implementation is the Migrating Desktop (MD) [3] which was developed under the CrossGrid project. The idea behind is nearly the same as for the CoG Kit Desktop. The details of the Grid should be hidden from the user by an easy to use graphical, desktop like interface. The communication between the Grid and the MD is realized by a Roaming Access Server (RAS). This server offers well defined Web Services which are wrapping interfaces to e.g. job submission clients. The realization of the user interface to such Web Services can differ depending on the job which will be submitted. Nevertheless MD is intended to be used for LCG base Grids with a well defined number of provided services. Adapting the architecture of the MD to a dynamic Web Service environment would be hard to realize. Rather GRAIL wants to pick up ideas of the MD and extend them to be usable in an environment where the number of services can change all the time. The following chapter 2 will point out the objectives of GRAIL in terms of the addressed Grid architecture. Chapter 3 describes new results achieved by GRAIL. The last chapter will give a conclusion concerning the usability of GRAIL.

2 Objectives

The objectives of GRAIL are to give possible solutions for all questions mentioned in the introduction together with additional simplifications for the developer during his daily work with the Globus Toolkit. Furthermore GRAIL offers an uniform graphical interface which is applicable for presentations or to introduce novices to the work with Globus based Grids by an instant feeling of success. GRAIL itself should not provide much implemented functionality because it is not meant to be a Grid portal or a dedicated tool for a specific user group. In fact GRAIL should only enable access to the Globus core components like the Monitoring and Discovery System, GridFTP, the Globus Resource Allocation Manager (GRAM) and security handling based on X.509 certificates. Apart from these features GRAIL should be as much extensible as possible to enable the integration of almost any functionality the user wants to. The user should be supported by offering wizards for regularly recurring tasks especially while working with Web Services. There should be none or only manageable additional work while using GRAIL. Summarizing a dynamic Grid environment,

as it can be realized by using Web Services, should be focussed. Concerning the scenarios mentioned in the first chapter, GRAIL should enable fast and easy integration of new components regardless if they want to provide graphical access to a Web Services or the parameterized execution of an algorithm implemented as Web Service or on the local machine. No matter what task lays behind GRAIL should abstract its representation. It should be possible to combine these tasks which differ in the location of their execution and their complexity within a chain of single executions and the ability to transfer results between actually independent steps. Every component of such an execution chain should be easily exchangeable during runtime to test different combinations of components. Chapter 3 will show the results of realizing these objectives.

3 Results of GRAIL

As pointed out within the objectives GRAIL is a graphical user interface which hides Grid functionality from the user. Functionalities are predominantly realized as components which can be combined on the fly via drag and drop. Monitoring and file transfer are realized as separate dialogs to enable their usage independent from other tasks. The creation of a Grid proxy for authentication and authorization between Globus container and the user is realized as a kind of login dialog during GRAIL's startup. The creation of new components is supported by a wizard which is integrated into GRAIL. Components which are intended to access Web Services are automatically generated after providing the URL leading to the service WSDL document. There exists a set of basic components. For the GRAM service there are for example three components. One to create a job description, another to submit the job to a specific service container and a third component to monitor the progress of submitted jobs. So, concerning GRAM, GRAIL can be used to generate job descriptions, to submit generated or existing descriptions by reading them from a file and to monitor submitted jobs, regardless they where submitted by GRAIL or any other client. The components can be connected to each other to realize a task graph. This way of integrating GRAM should show in which ways GRAIL can be used by developers. They can decide whether they integrate their functionality within one component or if they split it up into different components to test combinations between them. Wether a component accesses a Web Service or executes another program or a complex algorithm does not influence its appearance or the way of using it. The last chapter will give a conclusion about GRAIL together with the significance of this work.

4 Conclusions

GRAIL is a tool which can simplify the developers life in many ways. Even only with the set of basic components GRAIL offers a lot of feasible ways to access Globus based Grids. The developer can feel free to extend GRAIL by own components to ease especially his work. Due to the general definition of a

GRAIL component it is also possible to access any other middleware by wrapping available clients inside one or more components. Altogether GRAIL brings feasible ideas of the Migrating Desktop to the architecture of Globus Toolkit extended by the ability of large-scale extensions.

References

1. "Grid Development Tools for Eclipse". http://mage.uni-marburg.de/trac/gdt. 2006.
2. "Java CoG Kit Desktop". http://wiki.cogkit.org/index.php/Java_CoG_Kit. 2006.
3. "Migrating Desktop - Environment for Grid Interactive Applications". http://desktop.psnc.pl/.