Empowering a Flexible Application Portal with a SOA-based Grid Job Management Framework

Erik Elmroth[1], Sverker Holmgren[2], Jonas Lindemann[3], Salman Toor[2], and Per-Olov Östberg[1]

[1] Dept. Computing Science and HPC2N, Umeå University SE-901 87 Umeå, Sweden,
{elmroth, p-o}@cs.umu.se http://www.gird.se
[2] Dept. Information Technology, Uppsala University, Box 256,
SE-751 05 Uppsala, Sweden
{sverker.holmgren, salman.toor}@it.uu.se http://www.uu.se
[3] LUNARC, Lund University, Box 117,
SE-221 00, Sweden
jonas.lindemann@lunarc.lu.se http://www.lu.se

Abstract. There exists today a number of web-based portals and Grid middlewares that support the submission of jobs to Grid resources. A majority of these portals are tightly coupled with the underlying middleware, and rarely provide user interfaces customized to the end-user applications. The Lunarc Application Portal is a Grid portal that supports the development of customized user interfaces for Grid jobs. The Grid Job Management Framework is an application toolkit providing middleware-independent job management. This work describes an integration project combining these two architectures to provide a Grid portal that supports customization of application user interfaces and uses multiple middlewares in a flexible, robust, and loosely coupled way. The integration architecture is presented together with brief introductions to the two systems involved. Finally a performance analysis is performed.

1 Introduction

Today, there exist several web-based portals [5–7] and Grid middlewares for submitting and controlling jobs on Grid resources. These portals are often designed to offer alternative user interfaces to the functionality set offered by a specific middleware, and rarely provide more than limited support for the actual applications used. The main user groups for these portals are often experienced users with high levels of domain knowledge. In this work we study the integration of two systems: an application-oriented Grid portal, and a middleware-independent job management system.

The Lunarc Application Portal [8–10] is a Grid portal developed with focus on the end-users and their applications. The Grid Job Management Framework [1] is a middleware-independent job management toolkit that abstracts middleware functionality and offers multiple levels of job control granularity. The integration of these two systems provides a flexible architecture where user interfaces can be adapted to the specific nature of the applications and abstracted beyond the details of the underlying middleware. Combining the strengths of both systems, the resulting portal is designed to provide high levels of extensibility, installability, and customizability; and to facilitate the creation of customized user interfaces that support new applications.

1.1 The Lunarc Application Portal

The Lunarc Application Portal (LAP) is a web based portal for submitting jobs to Grid resources. The portal is implemented in Python using the WebWare for Python [11] application server. WebWare is a lightweight application server providing multi-user session handling, servlets, and page rendering. Although WebWare provides a built-in web server, most applications use the Apache web server and a special extension module, mod_webkit2, to forward HTTP requests to the WebWare application server. The recommended way of running LAP is through a SSL-enabled Apache web server.

The Lunarc Application Portal can be viewed both as a stand-alone web portal and as a Python-based framework for implementation of customized user interfaces for Grid-enabled applications. The core implementation includes a set of Python modules that provide services for management of users and job definitions, security, middleware integration, and user interface rendering. The portal also provides a set of servlets for non-application oriented tasks such as job definition creation, job monitoring, and job control.

Support for new applications in LAP is offered through the use of customization points and plug-ins. A LAP plug-in is comprised of a user interface generation servlet, a task class that defines job attributes, methods for generating job descriptions, and a set of bootstrap files required for Grid job submission.

In order to simplify the process of implementing a user interface, LAP provides an extensive object-oriented user interface module named Web.Ui. This module generates web user interface HTML, and also handles all types of form submission validations. LAP also provides functionality for automatic generation of xRSL job descriptions. This information is available to developers as properties in the Python task base classes.

1.2 The Grid Job Management Framework

The Grid Job Management Framework (GJMF) is an application toolkit for easy and robust job management in Grid environments. The framework is designed in accordance with the view of a healthy Grid ecosystem [2], and implemented using Java and the Globus Toolkit [3]. The GJMF is comprised of a hierarchical set of interchangeable Web Services that combined provide an infrastructure for virtualization of Grid middleware functionality and automatization of the repetitive tasks of job management.

The granularity of job management in the GJMF ranges from management of individual jobs to automatic brokering of sets of abstract task groups. The GJMF provides middleware virtualization by principle of abstraction and presents a common interface to Grid middleware functionality to developers and end-users. This is done without regard of the details of the underlying middleware. Functionality in the framework not supported by the underlying middleware, e.g., job state notifications in ARC, is emulated by the framework and presented to applications and users as native resources of the middleware. The GJMF also provides numerous structures for customization of the job management process. This customization ranges from individual configuration of the framework services to plug-in structures where, e.g., brokering algorithms, monitoring interfaces, failure handling, and job prioritization modules easily can be provided and installed by third party developers.

A full Java client API for the framework is provided and allow developers with limited experience of Web Service development to utilize the framework. This API, as demonstrated in this work, greatly facilitates the integration of the GJMF with other toolkits, e.g., application portals and customized Grid applications. Naturally, all features of the framework are accessible through both the Web Service and the Java client API interfaces. The GJMF utilizes JSDL for job descriptions, and provides a translation service for transformations to other job description formats.

2 Integration Architecture

In the integration of these systems, we employ a loosely coupled model where customized modules in the LAP dynamically access the GJMF services through the GJMF Java client API. For the Java-Python integration we use the JPype [4] (version 0.5.3). JPype is a library that allows Python applications to access Java class libraries within the Python process space. The LAP and GJMF are assigned the following responsibilities in the integration architecture.

– Application management: It is the responsibility of the LAP to provide application configuration parameters for the portal, gather job submission parameters, create application file repositories, acquire user credentials, authenticate users in the Grid environment, and to render user interfaces.
– Job management: The GJMF is responsible for all matters pertaining to actual Grid job management. This includes functionality for resource brokering, job submission, job monitoring, job control, and to provide robust handling of failures in job submission and execution.

The core of the integration architecture is based on Python threads, an approach that provides two primary benefits: At the front-end, user interface response times are reduced, while at the back-end, tasks are assigned dedicated execution threads. This allows the integration architecture to be simple, robust, and efficient. Roughly outlined, the integration architecture functions in the following way: The first time a job submission request is sent to the LAP the JPype connector creates a primary thread and instantiates a Java Virtual Machine (JVM) in it. Once a JVM is successfully instantiated, the connector creates a dedicated thread for each job. In order to access Java functionality from task threads, these threads are attached to the primary thread using JPypes native mechanisms. The creation of new threads is triggered by job submission, job monitoring, and result download requests.

The flexibility of the integration architecture also allows existing legacy applications supported by the LAP to continue to function unaltered. This is achieved by the portal maintaining the legacy job management setup, which utilizes the arcLib for job management.

As the GJMF utilizes JSDL, and the LAP xRSL, a translation between these two job description formats is necessary to maintain system interoperability. This is the responsibility of the GJMF JSDL Translation Service. This service has the capability of translating generic JSDL to and from the customized xRSL used in the LAP. Naturally, this service also contains customization points for extending the translation capabilities to support other formats.

The GJMF-empowered Lunarc Application Portal is currently available in a prototype version for SweGrid, supporting applications in computational chemistry and astrophysics. Demonstrations of its use and a performance analysis will be presented in an extended version of this contribution.

References

1. E. Elmroth, P. Gardfjäll, A. Norberg, J. Tordsson, and P-O. Östberg. Designing general, composable, and middleware-independent Grid infrastructure tools for multi-tiered job management. In T. Priol and M. Vaneschi, editors, *Towards Next Generation Grids*, pages 175–184. Springer-Verlag, 2007.
2. E. Elmroth, F. Hernández, J. Tordsson, and P-O. Östberg. Designing service-based resource management tools for a healthy Grid ecosystem. In R. Wyrzykowski et al., editors, *Parallel Processing and Applied Mathematics. 7th Int. Conference, PPAM 2007.* Lecture Notes in Computer Science, Springer-Verlag, 2007 (to appear).
3. I. Foster. Globus toolkit version 4: Software for service-oriented systems. In H. Jin et al., editors, *IFIP International Conference on Network and Parallel Computing,* Lecture Notes in Computer Science 3779, pages 2–13. Springer-Verlag, 2005.
4. JPype, http://jpype.sourceforge.net/, visited February 2008
5. Gridsphere Portal Framework, http://www.gridsphere.org/gridsphere/gridsphere, visited February 2008
6. Crossgrid, Developing new Grid components, http://www.crossgrid.org/main.html, visited February 2008
7. GridBlocks, http://gridblocks.hip.fi/, visited February 2008
8. J. Lindemann, G. Sandberg An extendable GRID application portal, European Grid Conference (EGC) Springer Verlag, 2005
9. P. Linde, J. Lindemann ELT Science Case Evaluation Using An HPC Portal *Astronomical Data Analysis Software and Systems XVII*, London 23-26 September, 2007
10. J. Lindemann, G. Sandberg A Lightweight Application Portal for the Grid *Nordic Seminar on Computational Mechanics NSCM 19*, Lund Sweden, 2006
11. Python Web Application Toolkit, http://www.webwareforpython.org, visited February 2008