

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

Erik Elmroth¹, Sverker Holmgren², Jonas Lindemann³, Salman Toor², and Per-Olov Östberg¹

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. The complexity of simultaneously providing customized user interfaces and transparent Grid access has led to a situation where current Grid portals tend to either be tightly coupled to specific middlewares or only provide generic user interfaces. In this work, we build upon the methodology of the Grid Job Management Framework and propose a flexible and robust 3-tier integration architecture that decouples application interface customization from Grid job management. Furthermore, we illustrate the approach with a proof of concept integration of the Lunarc Application Portal, which here serves as both a framework for the creation of application-oriented user interfaces and a Grid portal, and the Grid Job Management Framework, a framework for transparent access to multiple Grid middlewares. The loosely coupled architecture facilitates creation of sophisticated user interfaces customized to end-user applications while preserving the middleware-independence of the job management framework. The integration architecture is presented together with brief introductions to the integrated systems, and a system evaluation is provided to demonstrate the flexibility of the architecture.

1 Introduction

The task of constructing complete, robust, and high-performing systems that simultaneously provide sophisticated user interfaces and transparent access to computational resources is inherently complex. The range of Grid middlewares available today combined with the amount of applications (potentially) running on Grids introduces additional complexity. Thus, current portal-oriented efforts towards this goal [8, 11, 13] typically yield solutions that provide application-oriented interfaces tightly coupled to specific Grid middlewares, or Grid middleware solutions accessible only through generic user interfaces.

In this work we explore an architectural design pattern for development of advanced end-user applications capable of middleware-agnostic Grid use. We extend the methodology of [6] to development of flexible Grid portals that combine

application-oriented user interfaces with transparent Grid access. A 3-tier integration architecture that abstracts Grid functionality and isolates user interfaces from job management is proposed, and the approach is illustrated by an integration of the Lunarc Application Portal, an application-oriented Grid portal, and the Grid Job Management Framework, a middleware-independent Grid job management system designed for this purpose. The integration of these systems provides a flexible architecture where user interfaces can be adapted to specific applications and abstracted beyond the details of the underlying middleware.

1.1 The Lunarc Application Portal

The Lunarc Application Portal (LAP) is a web-based portal for submitting jobs to Grid resources [16–18]. The portal is implemented in Python using the Webware for Python [21] application server, and utilizes (in the original design) ARC/arcLib [5] for submitting and controlling jobs. 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 an SSL-enabled Apache web server.

The LAP can be viewed both as a 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 modules that provide 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 use of customization points and plug-ins. An 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 user interfaces, LAP provides an object-oriented user interface module, `Web.Ui`, that renders HTML for web user interfaces and handles form submissions. LAP also provides functionality for automatic generation of xRSL [5] job descriptions.

1.2 The Grid Job Management Framework

Developed in the Grid Infrastructure Research & Development (GIRD) [19] project, the Grid Job Management Framework (GJMF) [6] is a toolkit for job management in Grid environments. The design of the framework is a product of research on service composition techniques [9] and exploration of software design principles for a healthy Grid ecosystem [7]. The framework is implemented as a Service-Oriented Architecture (SOA), using Java and the Globus Toolkit [10].

The GJMF is comprised of a hierarchical set of replaceable 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 and fault-tolerant processing 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 without regard to 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 end-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 can be provided and installed by third party developers. See [6] for details.

A full Java client API for the framework is provided and allows developers with limited experience of Web Service development to utilize the framework. This API, as demonstrated in this work, facilitates integration of the GJMF with other systems, e.g., application portals and Grid applications. All features of the framework are accessible through both the Web Service interfaces and the Java client API. The GJMF utilizes JSDL [2] for job descriptions, and provides a translation service for transformations to other job description formats.

2 Integration Architecture

In the proposed architecture, we employ a loosely coupled model where customized modules in portals (the LAP) dynamically discover and access computational resources via a Grid access layer (the GJMF services). 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, gather job submission parameters, create application file repositories, acquire user credentials, authenticate users in the Grid environment, and to render user interfaces. For example, the LAP provides application requirement metadata in job descriptions to indicate and detail the use of Matlab in applications.
- Grid job management: The GJMF is responsible for all matters pertaining to 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. For example, the GJMF uses the previously mentioned application requirement metadata to broker Matlab jobs to Matlab-equipped hosts.

As illustrated in Figure 1, the original 2-tier architecture of the LAP has been extended into a classical 3-tier architecture [4] where the GJMF services are accessed through bridge modules and the GJMF client API. As can be seen in the illustration, the GJMF job management coexists non-intrusively with the legacy ARC/arcLib-based job management modules of the original LAP.

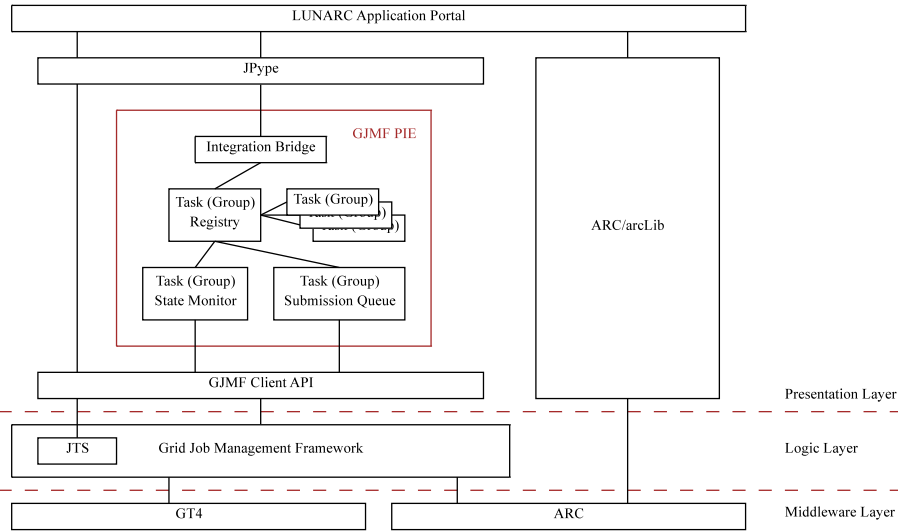


Fig. 1. Overview of the LAP-GJMF integration architecture. Integration components are presented without detailing the internal workings of the LAP or the GJMF.

The integration of the LAP and the GJMF has resulted in the development of the GJMF Portal Integration Extensions (PIE), a customization of the xRSL translation capabilities of the GJMF JSDL Translation Service (JTS), as well as the inclusion of a number of Java-Python bridge components in the LAP.

In the GJMF, it is the purview of the JTS to provide translations between job description formats and to ensure that job semantics are preserved in the process. In the case of the LAP-GJMF integration, there are two types of translations performed: an xRSL to JSDL translation is performed in the LAP upon job creation, and a translation from JSDL to the actual job description format used by the middleware (xRSL for ARC, and RSL [10] for GT4) is performed internally in the GJMF during the final stages of job submission. In the LAP-GJMF integration, the JTS uses job description annotations to provide semantically correct translations of application support parameters (e.g., preserving process environment information) for LAP applications. Naturally, the JTS also contains customization points for extending the translation capabilities to support other formats or alternative translation semantics.

The PIE is a Java-based software component consisting of an integration bridge, a task (group) registry, a submission queue, and a state monitor. These components provide an LAP interface, manage tasks and task groups, handle background GJMF submissions, and monitor GJMF state updates respectively. The PIE effectively wraps use of the GJMF client API and provides functionality for job submission, job control, notification-based state monitoring, and job brokering to the portal. PIE objects are deployed in authenticated sessions in the LAP, run inside the LAP process space, and help enforce user-level isolation

of job information in the portal (each user session is provided a unique PIE instance). In the LAP, bridge modules for job submission, job control, portal status updates, and job monitoring that interface with the PIE have been added. As the bridge modules are native to the LAP (i.e., developed in Python), JPype (version 0.5.3) has been employed to bridge the Java-Python barrier. JPype is a library that allows Python applications to access Java class libraries within the Python process space, connecting the virtual machines on native code level.

As also illustrated in Figure 1, the flexibility of the integration architecture allows existing legacy applications supported by the LAP to continue to function unaltered, including applications who have not yet been adapted to the new environment. This is achieved by the portal maintaining a concurrent legacy job management setup, which utilizes the ARC/arcLib [5] for job management.

When investigating the scalability and flexibility of the architecture, it should be noted that just as a single LAP can make use of multiple GJMFs, multiple LAPs can make use of the same GJMF. Similarly, just as a single GJMF can make use of multiple middleware installations concurrently, so can multiple GJMFs utilize the same middleware installation. Furthermore, as demonstrated by the test configurations of Section 3, the LAP, the GJMF, and the middleware(s) can all be hosted locally or distributed to dedicated servers over networks. The components of the architecture are designed to function non-intrusively [7] for seamless integration in production Grid environments.

3 System Evaluation

To evaluate and demonstrate the flexibility of the proposed architecture, a number of tests have been performed using a range of test configurations and a set of Grid applications that are in current production use.

Software Installations.

The test suites in the system evaluation have been run on nodes deploying different configurations of (at least) three software installations.

- LAP node: A front-end deploying an installation of the upgraded LAP. This node houses the PIE and the bridge components of the integration architecture as well as a fully functional legacy installation of the original LAP architecture. For file staging, the LAP node also deploys a GridFTP server.
- GJMF node: A node deploying a full installation of the GJMF. All GJMF services are run in the same GT4 ws-core 4.0.5 service container to enable inter-service local call optimizations [6, 9], and all communication is protected using GT4 Secure Conversation encryption.
- Middleware node(s): A (set of) middleware back-ends, running either the GT4 or the ARC middleware. The middleware node(s) also deploy middleware-specific GridFTP file staging solutions.

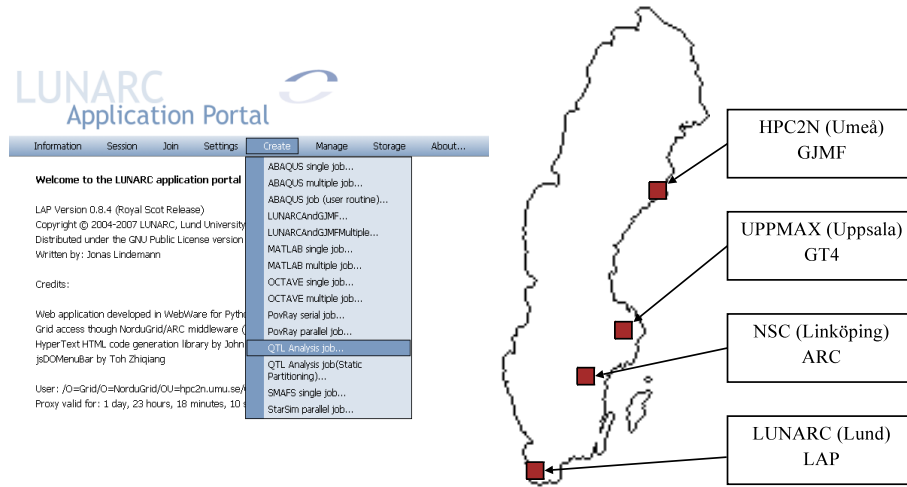


Fig. 2. The LAP and the production environment deployment configuration.

Deployment Configurations.

To illustrate the robustness and flexibility of the proposed architecture, the system is demonstrated in three deployment configurations. NorduGrid certificates are used for authentication of actors and security contexts in all tests.

- Local environment: All three software components are installed on the same machine, and each software component executes in a dedicated process.
- Distributed environment: Each software component is installed on a dedicated machine. All machines are located on the same network.
- Generic production environment: As illustrated in Figure 2, each software component is installed on geographically distributed production machines. The LAP node is located at LUNARC (Lund, Sweden), the GJMF node at HPC2N (Umeå, Sweden), GT4 middleware node(s) at UPPMAX (Uppsala, Sweden), and ARC middleware node(s) at NSC (Linköping, Sweden).

Test Applications.

Two applications for which the LAP is in production use today have been used to gauge the usability of the portal in a production environment.

- Matlab application: The LAP contains a bootstrapping module for initializing and executing Matlab code on Grid resources without use of the Matlab Compiler. This type of application requires a Matlab installation on the computational resource, executes a single job, and performs bidirectional file staging. The Matlab application module is here tested using an implementation of finite element code simulating stresses in straddling beams.
- Bioinformatics application (QTL mapping): This application searches for locations in the genome of an organism affecting a quantitative trait like body weight, crop yield, etc. The search is performed by solving a demanding

multidimensional global optimization problem, which is parallelized into a set of independent jobs. This type of application performs bidirectional file staging but does not require specific execution environment support libraries.

Usage Scenarios.

In the LAP, the main usage scenarios involve two user roles: the application expert and the end-user. Support for new applications is added to the LAP through the creation of application-specific plug-in modules that perform automatic creation of job environments, configuration of application workflows, and generation of application user interfaces. Creation and configuration of applications is the responsibility of the application expert (or system administrator). The process of creating, submitting, and managing jobs in the LAP is performed by the portal end-user, and includes four conceptual stages.

1. The portal end-user creates a job by instantiating a pre-configured application workflow, and supplies the job with required application parameters in the LAP. This results in the generation of an xRSL job description, which is later translated to a JSDL job description using the GJMF JTS.
2. The end-user submits the job from the LAP, an action resulting in the submission of a GJMF task (for single jobs) or a GJMF task group (for multiple jobs) to the PIE. The PIE places the task or task group in a background submission queue, and eventually submits it to the GJMF.
3. The GJMF processes the task or task group, submitting and resubmitting it to middlewares until the process has resulted in a successful job completion. Portal end-users can monitor task or task group progress in the LAP.
4. Once a task or task group has been processed by the GJMF, the end-user accesses resulting data files in the LAP, and removes the job from the LAP. All file stagings are performed by middlewares as GridFTP transfers between the LAP server and the computational resource used for job execution. The GJMF conveys file staging information, but is not actively involved in any file staging scenarios. The file staging semantics of the proposed architecture differ from the original architecture of the LAP, where end-users manually fetched job results from computational resources using HTTP requests. File transfer status is considered part of job execution status and a failed file staging attempt (in either direction) will result in a failed job. A multi-job task failure will not affect the status of other multi-job tasks.

4 Performance Observations

We briefly discuss the proposed architecture's impact on interface response times, job management overhead, and job execution makespans.

- Interface response times. Compared to the original 2-tier architecture of the LAP, the proposed integration architecture improves upon the system's user interface responsiveness, providing instantaneous response to user actions. This is due to the background submission queues of the PIE and the new

- architecture's use of the GJMF notification-based state monitoring, which improves scalability through a reduced need for middleware state polling.
- Job management overhead. The overhead associated with job management tasks performed by the GJMF (e.g., resource discovery, task brokering) sum to an average of less than one second per job and has previously been documented in [6]. As the GJMF job management overhead is masked by job execution times, the system-wide impact of this overhead is negligible.
 - Job execution makespan. The job execution makespan is made up by factors such as batch system queue times, middleware overhead, job execution time, and file staging times. These factors are independent of the proposed integration architecture and therefore out of the scope of this discussion.

The integration architecture has proven stable and provides enhanced functionality and middleware independence with comparable or improved performance.

5 Related Work

There exist a number of projects that implements web interfaces for Grid resources, such as Gridsphere [13], GridBlocks [11], and the P-GRADE portal [15]. The user interfaces of these portals are often designed as workflow editors and applications are viewed as building blocks in larger contexts. This differs from our work as the LAP focuses on creation of customized user interfaces for specific applications, and provides pre-configured workflows for target applications.

There also exist a number of projects related to the GJMF approach to job management, e.g., the Gridbus [20] middleware that employs a layered architecture and platform-independent approach to Grid job management; the GridWay Metascheduler [14] that provides reliable and autonomous execution of Grid jobs; the GridLab Grid Application Toolkit [1] that provides a service-oriented toolkit for Grid application development; GridSAM [12] that uses JSDL job descriptions and offers middleware-abstracted job submission through Web Services; and P-GRADE [15], which provides fault-tolerant Grid execution of parallel programs.

Related to the integration architecture, a kin project is the GEMLCA [3] integration with P-GRADE [15], where the layered architecture of GEMLCA is employed to run legacy applications as Grid services and P-GRADE provides interfaces for building execution environment, application monitoring, and results management. In comparison with other projects, the aim of the proposed architecture is to illustrate how to exploit the already available components of the LAP and the GJMF using the simplest possible integration model.

6 Conclusions

We have investigated integration techniques for user-friendly, robust, scalable, and flexible Grid portal architectures, proposed a layered approach to system integration, and demonstrated this in the integration of two existing systems; the LAP and the GJMF. The proposed integration architecture improves upon the

original LAP architecture in terms of scalability, support for multiple middlewares, performance, response times, and deployment flexibility. The user-friendly interfaces of the LAP abstract the use of the GJMF, allowing existing portal installations to be transparently upgraded to use the new integration architecture.

Use of the GJMF's automated brokering and job (re)submission capabilities improves the system's fault-tolerance and robustness, and introduces transparent middleware independence. As the multiple job submission mode of the LAP makes use of the GJMF Task Group Management Service (TGMS), the need for manual synchronization of jobs is eliminated and allows end-users to treat multiple jobs as a single management unit. Similarly, use of customized JTS job description translations facilitates middleware independence and automated job result retrieval. The middleware independence introduced by the GJMF allows for integration of new middlewares, facilitates transitions to new environments, and increases the expected lifetime of the LAP and LAP applications. Conversely, use of the LAP's ability to easily create customized application user interfaces empowers the GJMF with application support and usability features.

The proposed integration architecture is lightweight and non-intrusive, supports a representative range of Grid applications, and does not impede use of the original architecture's functionality in any way. In fact, the two versions are completely orthogonal in implementation and can co-exist in the same deployment environment. Use of the GJMF for job management in the portal contributes additional functionality in terms of resource brokering, failure handling, loose coupling of resources, and middleware independence. The PIE improves portal response times and scalability in state monitoring and job submission.

The GJMF-empowered LAP is currently available in a prototype version for SweGrid, supporting bioinformatics, computational chemistry, and astronomy applications. The Matlab extensions of the original architecture have been preserved and Matlab-based applications function unaltered in the new architecture.

7 Acknowledgments

This work has in part been supported by the Swedish Research Council (VR) under contract 621-2005-3667. For use of their resources, we acknowledge HPC2N, Umeå University, LUNARC, Lund University, NSC, Linköping University, and UPPMAX, Uppsala University. We also thank Daniel Henriksson, Johan Tordsson, and the anonymous referees for valuable feedback.

References

1. G. Allen, K. Davis, K. Dolkas, N. Doulamis, T. Goodale, T. Kielmann, A. Merzky, J. Nabrzyski, J. Pukacki, T. Radke, M. Russell, E. Seidel, J. Shalf, and I. Taylor. Enabling Applications on the Grid - A GridLab Overview. *International Journal of High Performance Computing Applications*, 2003.
2. A. Anjomshoaa, F. Brisard, M. Drescher, D. Fellows, A. Ly, A. S. McGough, D. Pulsipher, and A. Savva. Job Submission Description Language (JSDL) specification, version 1.0. <http://www.ogf.org/documents/GFD.56.pdf>, March 2007.

3. T. Delaittre, T. Kiss, A. Goyeneche, G. Terstyanszky, S. Winter, and P. Kacsuk. GEMLCA: Running legacy code applications as Grid services. *Journal of Grid Computing*, 3(1 - 2):75 – 90, June 2005. ISSN: 1570-7873.
4. E. Eckerson. Three Tier Client/Server Architecture: Achieving Scalability, Performance, and Efficiency in Client Server Applications. *Open Information Systems*, 10(1):3–22, 1995.
5. M. Ellert, M. Grønager, A. Konstantinov, B. Kónya, J. Lindemann, I. Livenson, J. L. Nielsen, M. Niinimäki, O. Smirnova, and A. Wäänänen. Advanced Resource Connector middleware for lightweight computational Grids. *Future Generation Computer Systems*, 27:219–240, 2007.
6. 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.
7. 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*, volume 4967, pages 259–270. Lecture Notes in Computer Science, Springer-Verlag, 2008.
8. E. Elmroth, M. Nylén, and R. Oscarsson. A User-Centric Cluster and Grid Computing Portal. *International Journal of Computational Science and Engineering*, 3(5), 2007 (to appear).
9. E. Elmroth and P-O. Östberg. Dynamic and Transparent Service Compositions Techniques for Service-Oriented Grid Architectures. In S. Gorlatch, P. Fragopoulou, and T. Priol, editors, *Integrated Research in Grid Computing*, pages 323–334. Crete University Press, 2008.
10. 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*, LNCS 3779, pages 2–13. Springer-Verlag, 2005.
11. GridBlocks. <http://gridblocks.hip.fi>, visited December 2008.
12. GridSAM. <http://gridsam.sourceforge.net>, visited December 2008.
13. Gridsphere Portal Framework. <http://www.gridisphere.org/gridisphere/gridisphere>, visited December 2008.
14. E. Huedo, R. S. Montero, and I. M. Llorente. A framework for adaptive execution on Grids. *Software - Practice and Experience*, 34(7):631–651, 2004.
15. P. Kacsuk and G. Sipos. Multi-grid and multi-user workflows in the P-GRADE Grid portal. *Journal of Grid Computing*, 3(3-4):221–238, 2006.
16. P. Linde and J. Lindemann. ELT Science Case Evaluation Using An HPC Portal. In *Astronomical Data Analysis Software and Systems XVII*, 2007.
17. J. Lindemann and G. Sandberg. An extendable GRID application portal. In *European Grid Conference (EGC)*. Springer Verlag, 2005.
18. J. Lindemann and G. Sandberg. A Lightweight Application Portal for the Grid. In *Nordic Seminar on Computational Mechanics NSCM 19*, 2006.
19. The Grid Infrastructure Research & Development (GIRD) project. Umeå University, Sweden. <http://www.gird.se>, visited December 2008.
20. S. Venugopal, R. Buyya, and L. Winton. A Grid service broker for scheduling e-science applications on global data Grids. *Concurrency and Computation: Practice & Experience*, 18(6):685–699, May 2006.
21. Webware, Python Web Application Toolkit. <http://www.webwareforpython.org>, visited December 2008.