

**A GRID PORTAL FOR ENVIRONMENTAL MONITORING
APPLICATIONS**

By

Mariana Mendoza-Botero

A project report submitted in partial fulfillment of the requirements for the degree
of

MASTER OF ENGINEERING

in

COMPUTER ENGINEERING

UNIVERSITY OF PUERTO RICO
MAYAGÜEZ CAMPUS

2007

Approved by:

Domingo Rodríguez, Ph.D
Member, Graduate Committee

Date

Nayda Santiago, Ph.D
Member, Graduate Committee

Date

Wilson Rivera, Ph.D
President, Graduate Committee

Date

Silvestre Colón, Ph.D
Representative of Graduate Studies

Date

Isidoro Couvertier, Ph.D
Chairperson of the Department

Date

Abstract of Project Presented to the Graduate School
of the University of Puerto Rico in Partial Fulfillment of the
Requirements for the Degree of Master of Engineering

**A GRID PORTAL FOR ENVIRONMENTAL MONITORING
APPLICATIONS**

By

Mariana Mendoza-Botero

2007

Chair: Wilson Rivera

Major Department: Electrical and Computer Engineering

Grid portals hide cyber-infrastructure complexity via easy-to-use interfaces, creating gateways to computing resources and data. This project report describes the development of a grid portal for a image processing application. In particular, this portal provides services to access environmental data, and apply signal processing operators on signals.

Resumen de Proyecto Presentado a Escuela Graduada
de la Universidad de Puerto Rico como requisito parcial de los
Requerimientos para el grado de Maestría en Ingeniería

**UN PORTAL GRID PARA APLICACIONES DE MONITOREO
AMBIENTAL**

Por

Mariana Mendoza-Botero

2007

Consejero: Wilson Rivera

Departamento: Ingeniería Eléctrica y Computadoras

Los portales grid ocultan la complejidad de ciber-infraestructuras por medio de interfaces de fácil uso, brindando acceso a recursos y datos de cómputo. Este reporte de proyecto describe el desarrollo de un portal grid para una aplicación de procesamiento de señales. En particular, éste portal provee servicios para acceder a datos ambientales, y ejecución de operadores de procesamiento de señales en un ambiente de trabajo distribuído.

Copyright © 2007

by

Mariana Mendoza-Botero

Dedicated to
my mother María Beatríz,
my father Carlos Arturo,
my sister Beatríz Helena,
my grandmother Rubiela.

ACKNOWLEDGMENTS

I would like to thank to my advisor Wilson Rivera for giving me the opportunity to study my master. Also, I would like to thank to my graduate committee, Dr. Nayda Santiago and Domingo Rodríguez for their valuable help and advice. Thanks to WALSAIP which allowed the development of this project.

Thanks to my expanding family for always give me support. Thanks to Sandra Montalvo (Sandy) an angel for the graduate students. Thanks to Andres, Arianna, David, Edith, Isabel, John, Juan, Juddy, Kennie, Nadia, Omar, Pablo, and all my special friends and coworkers for their company and advices. Thanks to God, Your hands always were on me.

This material is based upon work supported by the National Science Foundation under Grant No. 0424546.

TABLE OF CONTENTS

	<u>page</u>
ABSTRACT ENGLISH	ii
ABSTRACT SPANISH	iii
ACKNOWLEDGMENTS	vi
LIST OF FIGURES	ix
LIST OF ABBREVIATIONS	x
1 INTRODUCTION	1
2 CONCEPTS AND TOOLS	2
2.1 The WALSAIP Project Framework	2
2.1.1 Digital Image Processing Operators	3
2.2 Grid Computing	8
2.2.1 Grid Computing Architecture	9
2.2.1.1 Grid Architecture Model	9
2.2.1.2 Service Oriented Architecture (SOA Model)	10
2.2.1.3 Open Grid Services Architecture (OGSA)	10
2.2.1.4 WS-Resource Framework (WSRF)	12
2.2.2 Globus Toolkit	13
2.3 Grid Portals	14
2.3.1 Grid Portal Requirements	14
2.3.2 Grid Portal Standards	15
2.3.2.1 Java Portlet Specification (JSR 168)	15
2.3.2.2 Web Services for Remote Portlets (WSRP)	16
2.3.3 Grid Portal Development Frameworks	16
2.3.3.1 Commodity Grid (CoG) Kits	16
2.3.3.2 GridSphere Portal Framework	16
2.3.3.3 GridPort Toolkit	17
2.3.4 Grid Portal Applications	17
3 GRID PORTAL IMPLEMENTATION	20
3.1 Software Interfaces	21
3.2 Communication Interfaces	23
3.3 Grid Services Implementation and Deployment	23
3.4 Portlets and Services Integration	25

3.5	User Interfaces	26
3.6	About the Appendices	27
4	CONCLUSION AND FUTURE WORK	28
	APPENDICES	29
A	CREATING AND DEPLOYING A GRID SERVICE	30
B	PORTLET INTEGRATION	38
C	USER'S GUIDE	43
D	PROGRAMMING THE PORTLET	47
	BIOGRAPHICAL SKETCH	69

LIST OF FIGURES

<u>Figure</u>		<u>page</u>
2.1	CIP Environment	3
2.2	Digital Image Processing Operator Representation	5
2.3	Discrete Image Processing Operator Representation	6
2.4	WS-Resource	12
3.1	Grid Portal Application Architecture	21
3.2	Grid Portal Software Infrastructure	22
3.3	Five Steps to Create and Deploy a Grid Service	23
3.4	Portlet Architecture	26
3.5	Grid Portal Main Page	27
A.1	New Java Project Wizard	31
A.2	New Grid Service Wizard	33
A.3	Invert Operator Service Deployed in Globus Container	37
C.1	Portlet Web Application	43
C.2	Invert Portlet	44
C.3	Operator Grid Service Performed	46

LIST OF ABBREVIATIONS

CIP	Computational and Information Processing.
GoC	Commodity Grid.
DTS	Data Replication Service.
FTP	File Transfer Protocol.
GGF	Global Grid Forum.
GRAM	Grid Resource Allocation and Management.
GridFTP	Grid File Transfer Protocol.
GSI	Grid Security Infrastructure.
JSR	Java Specification Request.
MDS	Monitoring and Discovery System.
OGSA	Open Grid Services Architecture.
OGSA-DAI	Globus Data Access and Integration.
RFT	Reliable File Transfer.
RLS	Replica Location Service.
SAS	Sensor Array System.
SOA	Service Oriented Architecture.
VO	Virtual Organizations.
WALSAIP	Wide Area Large Scale Automated Information Processing.
WSDL	Web Services Description Language.

CHAPTER 1

INTRODUCTION

The availability of powerful computers and high speed network technologies, as low cost commodity components has changed the way we approach large scale problems. These technology opportunities have led to the possibility of using geographically distributed resources as a single unified computing resource, while increasing the possibilities for collaborative developments. Grid portals hide the complexity of these systems by providing easy to use interfaces to computing resources and data.

The Wide-Area Large-Scale Automated Information Processing (WALSAIP) project is developing an infrastructure for the automated processing of information arriving from physical sensors focused on environmental applications. The specific objective of the project presented in this report is to provide transparent access to a set of signal processing operators related to environmental applications that may be available on distributed resources. To achieve this objective we leverage on existing grid technologies including the Globus toolkit and Gridsphere. The contribution of this project is a prototype of the WALSAIP grid portal, which provides access via portlets to diverse signal processing operators in a distributed framework.

This project report is organized as follows: Chapter 2 introduces the concepts and tools related to signal processing operators and grid computing. Chapter 3 discusses the architecture and implementation issues of the WALSAIP Grid Portal. Finally, Chapter 4 presents conclusions and future work.

CHAPTER 2

CONCEPTS AND TOOLS

2.1 The WALSAIP Project Framework

The *Wide Area Large Scale Automated Information Processing (WALSAIP)*¹ project is developing a new conceptual framework for the automated processing of information arriving from physical sensors in a generalized wide-area, large-scale distributed network infrastructure. The WALSAIP project is focusing on water-related ecological and environmental applications, and it is addressing issues such as scalability, modularity, signal representation, data coherence, data integration, distributed query processing, scheduling, computer performance, network performance, and usability. This new framework treats signals as elements in prescribed sets and their associated structures. The project is constructing a Computational and Information Processing (CIP) environment to deal with the algorithmic treatment of signal-based large scale content in order to extract information relevant and important to a user (see Figure 2.1). New theories and algorithms for computational signal processing to gather, process, and represent data obtained from physical sensors are also under development. Further, it is also developing new concepts in middleware integration, distributed query optimization, distributed query processing, and distributed scheduling algorithms to adapt to an ever changing network infrastructure and to provide a pathway between a physical world sensory reality, with its

¹ <http://walsaip.uprm.edu/>

associated physical sensors, and a user, with network and database infrastructure applications.

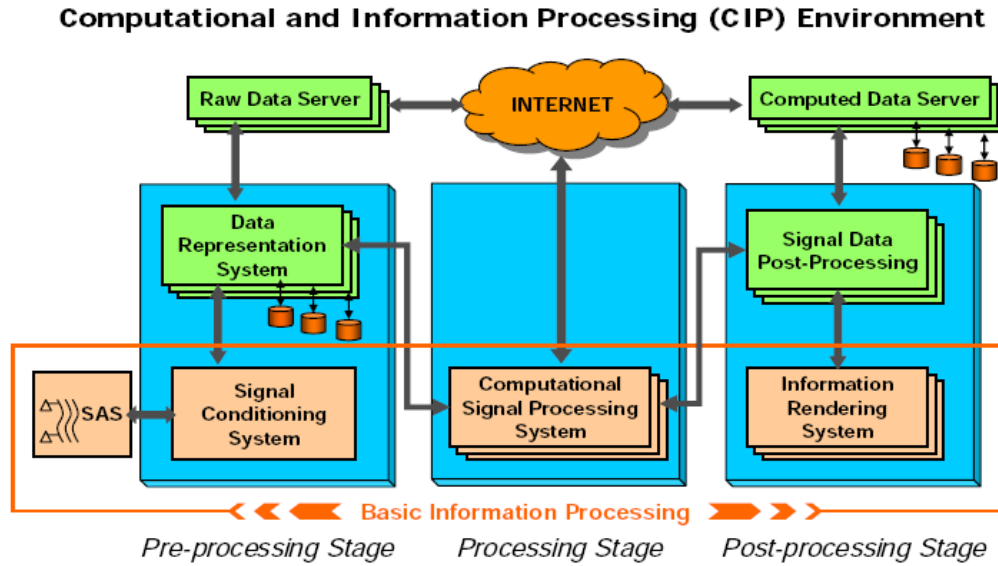


Figure 2.1: CIP Environment

As illustrated in Figure 2.1, the core of the processing stage is the computational signal processing system, which includes a set of generalized signal processing operators.

2.1.1 Digital Image Processing Operators

The concept of the digital image processing operators is very useful when dealing with the algorithmic treatment of two-dimensional arrays of data of finite order [1]. The algorithmic treatment of two-dimensional array of data finite order has as a main objective to extract relevant information important to users. The *algorithmic treatment* is a well defined procedure to accomplish a task in a finite number of step. This section describes the concept of a digital image processing operator and presents a formulation of the linear operator mostly used in image processing, namely, the *linear convolution operator*. This section also presents the other linear operator commonly used in image processing, the Fourier transform operator.

In fact, these two linear operators and its parametric variants represent the main bulk of all operators used in scientific and engineering applications. To introduce the concept of digital image processing operator, we first present some preliminary concepts dealing with array data indexing sets and how they define the structure of the associate data.

Let $Z_N = a[0], a[1], a[2], \dots, a[N - 1]$ be the standard indexing set of order N . This set is useful to index a set of one dimension of data of N elements, say, $A = \{a[0], a[1], a[2], \dots, a[N - 1]\}$.

$$\begin{aligned}
 \text{Let } Z_M \times Z_N = & \{(0, 0), (0, 1), (0, 2), \dots, (0, N - 1), \\
 & (1, 0), (1, 1), (1, 2), \dots, (1, N - 1), \\
 & \dots \\
 & (M - 1, 0), (M - 1, 1), (M - 2, 2), \dots, (M - 1, N - 1)\}
 \end{aligned} \tag{2.1}$$

be the two-dimensional standard indexing set of order $M \times N$. A two-dimensional standard indexing set determines the structure of the data array to be indexed.

A two-dimensional array data set of order $M \times N$ is defined as a set of MN real numbers or values which are indexed by the standard indexing set $Z_M \times Z_N$. We then say that this array data set has M rows and N columns. We call this array data set a *real image* of M rows by N columns if all the MN real numbers are non-negative. In the same manner, we may call this array of M rows by N columns a *digital image* if all of the MN real numbers, in addition of being non-negative, are also integer values. The set of integer values $0, 1, \dots, 255$ is commonly used to represent the MN values of a digital image, where '0' represents zero intensity and '255' represents

maximum intensity.

A digital image processing operator, say O_g (general operator), is a mathematical function which has as input a digital image, say (d_i) , and it has as output another digital image, say (d_o) . The following block diagram representation is normally used to describe a given digital image processing operator (O_g)



Figure 2.2: Digital Image Processing Operator Representation

Where $Z_P = 0, 1, 2, 3, \dots, P - 1$ and $Z_Q = 0, 1, 2, 3, \dots, Q - 1$.

$$\begin{aligned}
 \text{and } Z_P \times Z_Q = & \{(0, 0), (0, 1), (0, 2), \dots, (0, Q - 1), \\
 & (1, 0), (1, 1), (1, 2), \dots, (1, Q - 1), \\
 & \dots \\
 & (P - 1, 0), (P - 1, 1), \\
 & (P - 1, Q - 1)\}.
 \end{aligned} \tag{2.2}$$

Even though in practice we implement digital image processing operators on computational structures, it is more natural to deal with discrete image processing operators to obtain a better generalization of algorithms results from the mathematical point of view:

The input to the discrete image processing operator (O_g) in this case is a discrete image instead of a digital image. A discrete image is able to admit any non-negative real numeric value. This is in contrast to the digital image which only admit a finite

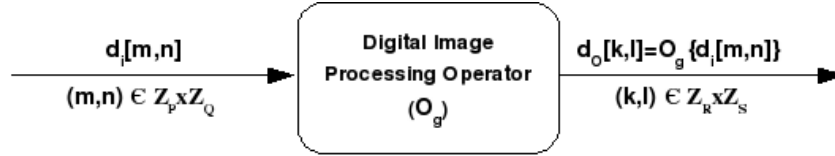


Figure 2.3: Discrete Image Processing Operator Representation

number of integer values.

This work present two examples of discrete image processing operators, namely, the discrete two dimensional linear convolution operator and the discrete two-dimensional Fourier transform.

The discrete two-dimensional linear convolution of and input image, say $d_i[m, n]$, $m, n \in Z_P \times Z_Q$, with respect to a filtering function $h_i[m, n]$, $m, n \in Z_M \times Z_N$, produces an output image, say $d_o[m, n] \in Z_R \times Z_S$, whose individual values are given by the following expression.

$$d_o[m, n] = \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} h_i[k, l] \cdot d_i[m - k, n - l], m \in Z_R, n \in Z_S.$$

where $R = P + M - 1$, and $S = Q + N - 1$.

The two-dimensional discrete Fourier transform of a discrete image, say $X[m, n]$, $m, n \in Z_P \times Z_Q$, is defined as a complex image, say $K[k, l]$, through the following expression:

$$X[k, l] = \sum_{m=0}^{P-1} \sum_{n=0}^{Q-1} X[m, n] \cdot e^{(-J2\pi mk)/P} \cdot e^{(-J2\pi nl)/Q}, J = \sqrt{-1}, k, l \in Z_Q \times Z_P.$$

The complex $X[k, l]$, $k, l \in Z_P \times Z_Q$ is composed of two real images $X_R[k, l]$ and $X_I[k, l]$ defined by the following expression:

$$X[k, l] = X_R[k, l] + j \cdot X_I[k, l], k, l \in Z_P \times Z_Q$$

where $R = P + M - 1$, and $S = Q + N - 1$.

Several computational environments for image processing have been developed. Bautista [2] implements a Java-based environment for the treatment of remote sensing imaging information which implements a set of image processing operators along with a mechanisms to compose subset of operators.

Del Fabbro [3] presents a framework for the distributed manipulation of large georeferenced images. He developed a visual programming language (VPL) that allows users to create programs to be executed in a distributed environment. He used the Java Advanced Imaging (JAI), also considered in our project to process the images.

Tsou et. al. [4] introduce a web-based remote sensing application that can provide advanced image comparison and processing functions for natural habitat conservation and environmental monitoring. This project adopted Java programming tools to create a series of Java applets that can perform multiple image analysis and change detection functions. These Java applets were organized into an image analysis toolbox, where users can select different buttons to launch different applets (image processing functions) in separate windows.

Smith [5] presents the Nano-Toolbox project that is a data and image analysis package, specifically developed to be integrated into nano-systems. It provides methods to import and export data as well as direct intercommunication with a

nano-systems e-learning environment.

Our project follows a different approach as it leverages existing grid computing technologies to provide access to distributed signal processing operators in a transparent way to end users. In the next section we introduce these technologies.

2.2 Grid Computing

Grid computing deals with the computing power that is supplied by a set of resources in a distributed environment. More formally the Open Grid Forum [6] defines a *grid* as a system concerned with the integration, virtualization, and management of services and resources in a distributed, heterogeneous environment that supports collections of users and resources (*virtual organizations*) across administrative and organizational domains (*real organizations*). A grid system can be characterized by three important aspects [7]:

1. *A system that coordinates resources that are not subject to centralized control:* Sharing implies direct access on behalf of *virtual organizations (VOs)*[8] to computers, software, data, and other resources for collaborative problem-solving. VOs associate users, their requests, and the set of interacting resources. Sharing resources in a VO should be highly controlled, with resource providers and consumers defining clearly and carefully what is shared, who is allowed to share, and the conditions under which sharing occurs.
2. *A system using standard, open, general-purpose protocols and interfaces:* To achieve desirable communication among those heterogeneous VOs, it is very important for both users and resources, to offer authentication, authorization, resource discovery, and resource access through standard protocols. The Open Grid Forum contributes to this grid computing aspect by developing such protocols. A *standard protocol* allows resource-sharing arrangements to be established with any interested party

and thus creating a compatible and interoperable distributed system. A *general-purpose protocol* enables the implementation of general-purpose services and tools by means of this standardization.

3. *A system to deliver nontrivial qualities of service:* To supply the application's requirements the system must be able to coordinate the allocation of resources according to nontrivial quality of service criteria.

2.2.1 Grid Computing Architecture

The Open Grid Services Architecture (OGSA) is commonly used as an open-standard architecture, aligning the best aspects of both the Grid Architecture model and the Service Oriented Architecture (SOA) model. In this section we discuss briefly the architectural models, the services capabilities defined in OGSA as well as the implementation specifications given by the Web-Services Resource Framework (WS-RF).

2.2.1.1 Grid Architecture Model

The *Grid Architecture Model* defines the layers to provide interoperability. Interoperability allows the grid system to offer a sharing environment between resources providers and consumers via a set of common protocols. This model has five layers briefly described as follows [9].

- *Fabric layer:* Defines the interfaces to access local resources to be shared.
- *Connectivity layer:* Defines the communication and authentication protocols of network transactions among resources.
- *Resource layer:* Uses the communication and authentication protocols of the connectivity layer to manage the sharing of individual resources. This management controls secure negotiations, instantiation, monitoring, accounting, and payment. It also uses the fabric layer function to access and control local resources.

- *Collective layer*: Manages the interactions among collections of resources. It uses protocols from both connectivity and resource layers. It also implements information protocols to access the structure and state of resources, as well as manages protocols to negotiate access to resources.
- *Application layer*: Includes all different user applications, portals and development toolkits supporting the applications. Applications rely on all layers below to run on the grid.

2.2.1.2 Service Oriented Architecture (SOA Model)

SOA [10] is a paradigm for organizing distributed capabilities that may be under the control of different ownership domains. A *service* in SOA is a mechanism to enable access to one or more capabilities. Services in SOA have three fundamental properties:

- Access is provided using a prescribed interface consistent with the constraints and policies specified by the service description. This property is used to describe and publish services.
- Localization and invocation can be dynamic. This property is used to discovery services.
- Maintain its own state. This property permits services be stateful to consumption or interaction.

2.2.1.3 Open Grid Services Architecture (OGSA)

The Global Grid Forum (GGF) has embraced *OGSA* as the industry blueprint for standards-based grid computing. In OGSA, a *grid service* [11] is a web service that provides a set of well defined interfaces and follows specific conventions. The services implemented under OGSA fall into seven areas defined in terms of capabilities that are frequently required in grid scenarios:

- *Infrastructure Services*: Refers to a set of common functionalities typically required by higher level services. As OGSA builds on Web services technologies,

service interfaces are defined by the Web Services Description Languages (WSDL). Infrastructure includes emerging standards such as the Web Services Resource Framework (WSRF), WS-Notification (WSN), Web Services Distributed Management (WSDM), and Naming (RNS and WS-Naming).

- *Execution Management Services*: Concern with issues such as starting and managing tasks, including placement, provisioning, and lifecycle management. Tasks may range from simple jobs to complex workflows or composite services.
- *Data Services*: Provide functionality to move data, manage replicated copies, run queries and updates, and transform data into new formats. Data consistency, persistence and integrity are key requirements satisfied by these services.
- *Resource Management Services*: Provide management capabilities for grid resources such as: management of the resources themselves, management of the resources as grid components, and management of the OGSA infrastructure. For example, resources can be monitored, reserved, deployed, and configured as needed to meet application quality-of-service requirements.
- *Security Services*: Facilitate the enforcement of security-related policy within a (virtual) organization, and supports safe resource-sharing. Authentication, authorization and integrity-assurance are essential functionalities provided by these services.
- *Self-Management Services*: Support service-level attainment for a set of services (or resources), to reduce the costs and complexity of managing the system. These services are essential in addressing the increasing complexity of owning and operating an information technology infrastructure.
- *Information Services*: Provide efficient production of, and access to, information about the Grid and its constituent resources. The term information refers to dynamic data or events used for status monitoring. Troubleshooting is just one of the possible uses for information provided by these services.

2.2.1.4 WS-Resource Framework (WSRF)

Grid service is a web service that is able to maintain state. WS-Resource Framework provides state to stateless web services. WSRF describes the specifications on which the OGSA architecture is built. In order to minimize the complex problem of adding state to web services [12], it has been proposed that a web service and its state information be kept completely separate. The entity that stores this information state is called the *resource*, identified by a unique key. Pairing of a Web service with a resource is called a *WS-Resource*. The address of a particular WS-Resource is called an *endpoint reference* (this is WS-Addressing) (see Figure 2.4).

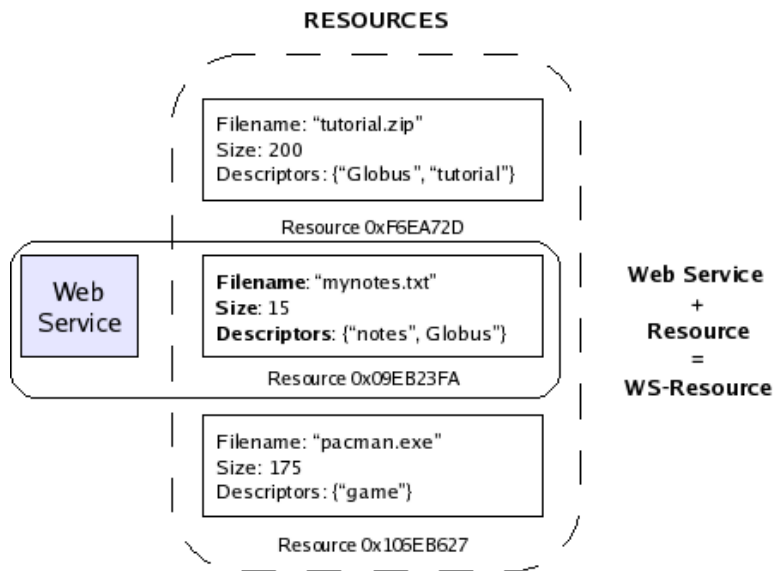


Figure 2.4: WS-Resource

WSRF integrates the following collection of specifications in order to manage the WS-Resources:

- *WS-ResourceProperties*: Specifies how resource properties are defined and accessed. A resource is composed of zero or more resource properties.
- *WS-ResourceLifetime*: Supplies some basic mechanisms to manage the lifecycle of the resources. Resources can be created and destroyed at any time.

- *WS-ServiceGroup*: Specifies exactly how grouping services or WS-Resources together.
- *WS-BaseFaults*: Provides a standard way of reporting faults when something goes wrong during a WS-Service invocation.
- *WS-Notification*: It is not a part of WSRF, but is closely related to it. It allows a Web service to be configured as a *notification producer*, and certain clients to be *notification consumers* (or subscribers). This means that if a change occurs in the Web service (or, more specifically, in one of the WS-Resources), that change is *notified* to all the subscribers.
- *WS-Addressing*: Provides a mechanism to address Web services which is much more versatile than plain URIs.

2.2.2 Globus Toolkit

The *Globus Toolkit (GT)* middleware is composed of a set of modules, each one implementing a specific functionality. Those modules are conceptualized mainly as services implemented on top of WSRF. The Globus modules include [13]:

- *Task Execution Management*: The *Grid Resource Allocation and Management (GRAM)* service provides a Web services interface to initiate, monitor, and manage the execution of arbitrary tasks on remote computers.
- *Data Management*: A set of services to manage data transport and management.
 - *Grid File Transfer Protocol (GridFTP)* provides libraries and tools for reliable, secure, high-performance memory-to-memory and disk-to-disk data movement.
 - *Reliable File Transfer (RFT)* service provides reliable management of multiple GridFTP transfers.
 - *Replica Location Service (RLS)* provides information on the location of replicated files and datasets.

- *Data Replication Service* combines RLS and GridFTP to allow for the management of data replication.
- *Globus Data Access and Integration (OGSA-DAI)* tools provide access to relational and XML data.
- *Resource Monitoring and Discovery*: The *Monitoring and Discovery Service (MDS)* facilitates monitoring and the discovering services and resources in a distributed system.

Grid portals provide a solution to overcome the complexity of managing and accessing grid infrastructures. The next section describes the grid portal requirements, standards, and development frameworks.

2.3 Grid Portals

According to the Portlet Specification, a *grid portal* is a Web application that commonly provides personalization, single sign on, content aggregation from different sources, and seamless access to grid heterogeneous resources and services.

2.3.1 Grid Portal Requirements

The requirements for a grid portal can be listed as follows [14][15]:

- *Security Services*: The user authentication process is one of the distinguishing characteristics of grid portal. The Open Grid Forum, promotes secure access to grid resources through *GSI (Grid Security Infrastructure)*, in which each user needs an authenticated key and certificate to access resources and grid services. GSI supports delegation, meaning that a temporary user certificate (*proxy*) is created with its private key to provide unique authentication. MyProxy [16] is a certificate repository developed to support delegation.
- *Remote File Management*: Grid portals must provide access to files, collections, and metadata for local and remote files, and also support third party file transfer.

In the Globus toolkit this source is provided by GridFTP with authentication based on GSI.

- *Remote Job Management:* Grid portals must provide mechanisms for job execution and monitoring. Such monitory tasks include capabilities to observe job queues follow job execution, and consult logs when jobs fail. In the Globus toolkit this is provided by GRAM.
- *Access to Information Services:* Grid portals must provide access to directories and status tools, that indicate the state of executions and possible faults.
- *Application Interfaces:* Grid portals must hide grid details behind useful application interfaces.
- *Access to Collaboration:* Grid portals must serve as gateways to virtual organizations (VO) to share resources.

2.3.2 Grid Portal Standards

Portlets standards guide the implementation of portable portlets to allow interoperability and exchange among portlet containers. A *portlet* [17] is a piece of code that runs on the portal server and allows content to be embedded into portal pages. A *portlet container* [17] is the server-side run-time environment. It calls the component and provides component-specific services (such as user information and persistence).

2.3.2.1 Java Portlet Specification (JSR 168)

The JSR 168 [18] standard establishes a standard API for creating portlets. Portlet specification is required to achieve interoperability between portlets and Java-based portal servers or other web applications that implement the specification. The goal is to allow portlets to be packaged into Web Application aRchive (WAR) files and deployed in a standard way on any server implementing the specification.

2.3.2.2 Web Services for Remote Portlets (WSRP)

The Web Services for Remote Portlets (WSRP) [19] standard specifies the interfaces to allow applications (typically a portal) to consume its portlets, regardless of its technology.

2.3.3 Grid Portal Development Frameworks

Currently, a large variety of grid-enable portals technologies are available.

2.3.3.1 Commodity Grid (CoG) Kits

The CoG kit [20] allows grid users, grid application developers, and grid administrators to use, program, and administer grids from a higher-level framework. It is used with Globus Toolkit and provides Java-based GSI, GridFTP, MyProxy, and GRAM client implementations. The Architecture of the Java CoG kit is based on a layered module concept that allows easier maintenance and bridges the gap between applications and the grid middleware. It allows the easy integration of enhancements developed by the community.

2.3.3.2 GridSphere Portal Framework

The primary goal of GridSphere portal framework [21] project is to develop a standards based portlet framework for building web portals, and a set of portlet web applications that work seamlessly with the GridSphere framework to provide a complete Grid portal development solution. The integration of the GridSphere portal framework with the collection of gridportlets provided as an add-on module forms a cohesive grid portal end-user environment for managing users, supporting remote job execution, and providing access to information services. The GridSphere portal framework provides an implementation of the JSR 168 portlet API standard. It supports the development of re-usable portlets and portlet services. It includes a set of core portlets and portlet services that provide the basic infrastructure required for developing and administering Web portals. A key feature of their design is that it builds upon the web application repository (WAR) deployment model to support

third-party portlet web applications. In this way, portlet developers can easily distribute and share their work with other portal projects that use GridSphere to support their portal development.

2.3.3.3 GridPort Toolkit

The GridPort Toolkit (GridPort) [22] enables the rapid development of highly functional grid portals that simplify the use of underlying grid services for the end-user. It is comprised of a set of portlet interfaces and services in the portal layer that provide access to a wide range of backend grid and information services provided by lower-level grid technologies including the Globus Toolkit, Grid Portal Information Repository (GPIR), and Condor. Portlets expose the backend services via customizable web interfaces in order to enable personalization of grid portal user interfaces. Portal services support the portlets inside the portal layer by augmenting their capabilities in an extensible and reusable way while tying the portlets together in order to make them more cohesive.

In summary, CoG is a Java toolkit that provides a Globus toolkit-independent abstraction layer to access Globus services programmatically. Both GridSphere and GridPort are based on CoG, but while GridSphere does not require CoG programming experience, GridPort does.

2.3.4 Grid Portal Applications

The Geosciences Network (GEON) [23] project is a collaboration among a dozen institutions and a number of other partner projects, institutions, and agencies to develop cyberinfrastructure in support of an environment for integrative geoscience research. GEON's purpose is to enable scientists to access, synthesize, and model geoscience data from a wide variety of sources. GEON consists of digital libraries containing freely available data from many geoscience disciplines, along with an integrated set of software tools for access, analysis, visualization, and modeling.

Integrating, analyzing, and modeling geoscience data is a major challenge because of the extreme heterogeneity of data formats, storage and computing systems, and conventions, terminologies, and ontological frameworks that are found in the disciplines that comprise the earth sciences. GEON has adopted a service-oriented approach and a portlet-based approach. The GEON architecture consists of a single GEON Portal and a number of distributed GEON sites and GEON service providers. The Portal (using GridSphere framework) provides a single point of entry into this distributed environment. Remote sites and service providers operate in the service-oriented architecture (SOA) environment of GEON.

The Linked Environments for Atmospheric Discovery (LEAD) [24] project was created to develop an infrastructure for solving problems of mesoscale weather events that deal with disparate, high volume data sets and streams, and the computational demands and logistical complexity of its numerical models and data assimilation systems. Currently, the portal is identifying, accessing, preparing, assimilating, predicting, managing, analyzing, mining, and visualizing a broad array of meteorological data and model output, independent of format and physical location. The LEAD Portal is, essentially, a set of Java portlets complying with the JSR 168 Java Portlet 1.0 specification hosted within a GridSphere framework. For the user, the LEAD Portal takes care of loading the user's security credential and making it available to the LEAD portlets. The portlets in turn use the user's security credentials to interact with LEAD web services on the user's behalf.

The Oceans and Climate Digital Library portal [25] offers an integrated web-based user interface to a wide range of online oceanographic and climate data sets. The portal provides a single point of entry to the data sets, and displays them in a common and easy to navigate format. Moreover, it provides a means of searching

the meta-data and other characteristics of the data sets. A key component of the portal design is the search crawler, which periodically checks specified locations (local directories or external websites) for updated or modified data sets. The crawler updates the portal with the most current state of every data set, and allows new data sets to be discovered and automatically added to the system. The portal facilitates the sharing of earth systems science data sets, and enable scientists to more efficiently manage their data holdings and data analysis work flows. The portal front-end is build with Gridsphere framework and its portlets interact with the environmental data sets.

The SURA Coastal Ocean Observing and Prediction (SCOOP) portal [\[26\]](#) is engaged in real time prediction of severe weather events, including tropical storms and hurricanes, and provides operational information including wind, storm surge and resulting inundation, important for emergency management. The SCOOP portal, built with the GridSphere Framework, currently integrates customized Grid portlet components for data access, job submission, resource management and notification.

CHAPTER 3

GRID PORTAL IMPLEMENTATION

This project focuses on the development of a set of grid services related to signal processing operators. Each operator may be implemented as a grid service on a geographically distributed domain. Such grid services can be accessed through portlets allocated into the grid portal.

The physical infrastructure supporting the portal is the PDCLab grid testbed deployed by the Parallel and Distributed Computing Laboratory at the University of Puerto Rico. The PDCLab grid-testbed is an experimental grid designed to address research issues, such as the effective integration of sensor networks to grid infrastructures. The PDCLab grid test-bed components run CentOS 4.2 and the Globus Toolkit 4.0.1. The computational resources available on the grid include:

- An IBM xSeries Linux cluster with 64 nodes, dualprocessor at 1.2GHz, 53GB of memory and 1TB of storage.
- Eight (8) IA-64 Itanium servers, dual processor at 900 MHz, each with 8GB of memory and 140GB of SCSI Ultra 320 storage.
- Two (2) IA-32 Pentium IV servers, dual processor at 3 GHz, each with 1GB of memory and 120GB of ATA-100 storage.
- One (1) IA-32 Pentium III server, dual processor at 1.2 GHz with 2GB of memory and 140Gb of SCSI Ultra 160 storage.

- One (1) IA-32 Xeon server, dual processor at 2.8 GHz, L2 Cache 1MB with 1GB of memory and one 230 GB RAID of storage (STB Server).
- Two (2) PowerVault storage with 8TB.

As general idea, a portlet invokes the corresponding grid service; the grid service processes the request; it returns the response to the portlet, and finally it renders it into the user interface. Figure 3.1 illustrates the main components of our application architecture. Each grid service is implemented to offer an image processing operator functionality. There are twelve grid services distributed in two host environment nodes. There is a grid portal containing a component interface (portlet) for each grid service. The grid portal is running on an independent node from the host environment nodes. The data image transference is made through GridFTP. Portlet grid service communication is established through SOAP messages.

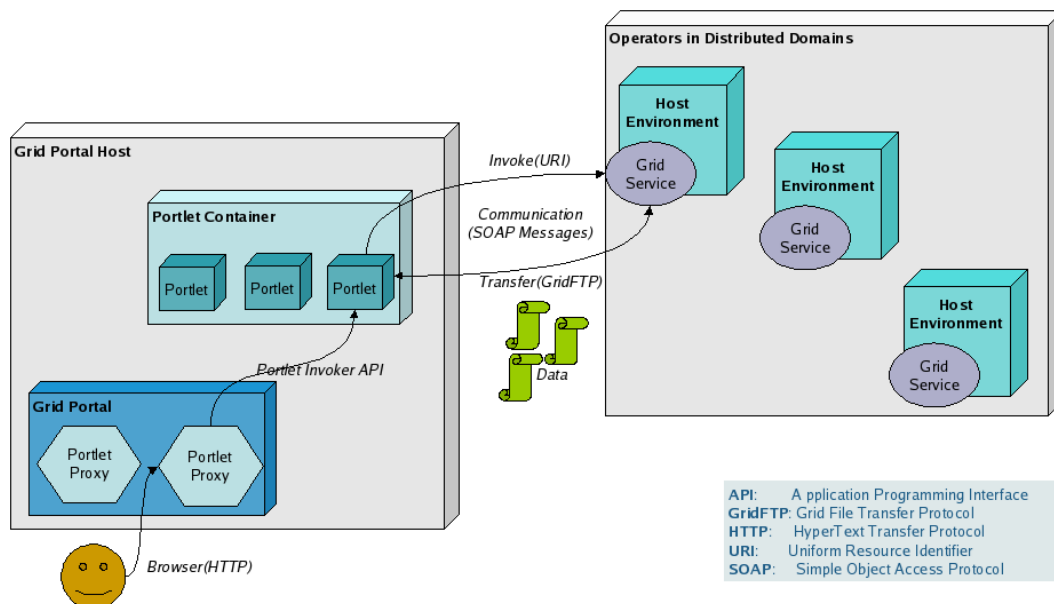


Figure 3.1: Grid Portal Application Architecture

3.1 Software Interfaces

The application infrastructure is described as follows (see Figure 3.2):

- *Grid portal*: GridSphere portal framework version 2.2.7 is used to implement the grid portal interface. Apache Tomcat server version 5.5.20 is the Web server and supports the GridSphere portlet container.
- *Middleware*: Globus Toolkit version 4.0.1 is the middleware used to support grid service transactions. It is installed only on the portal node.
- *Grid services*: Eclipse 3.2 is used as integrated development environment. Java language version 1.5 is used to implement the logic of grid services. Java Service Pages (JSP) language and GridSphere Tag Library User's are used to implement the user interface to portlets. Grid Development Tools (GDT-MAGE) plug-in on Eclipse is used to facilitate the grid services creation.
- *API's*: The Java Advanced Image (JAI) library version 1.1.3 is used to execute the image operations. JFreeChart library version 1.0.4 is used to generate the plots. Document Object Model (JDOM) version 1.1 is used to manage XML files.
- The system runs on Linux. End-users can access the application with Linux or Windows system operative through Firefox and Opera web browsers.

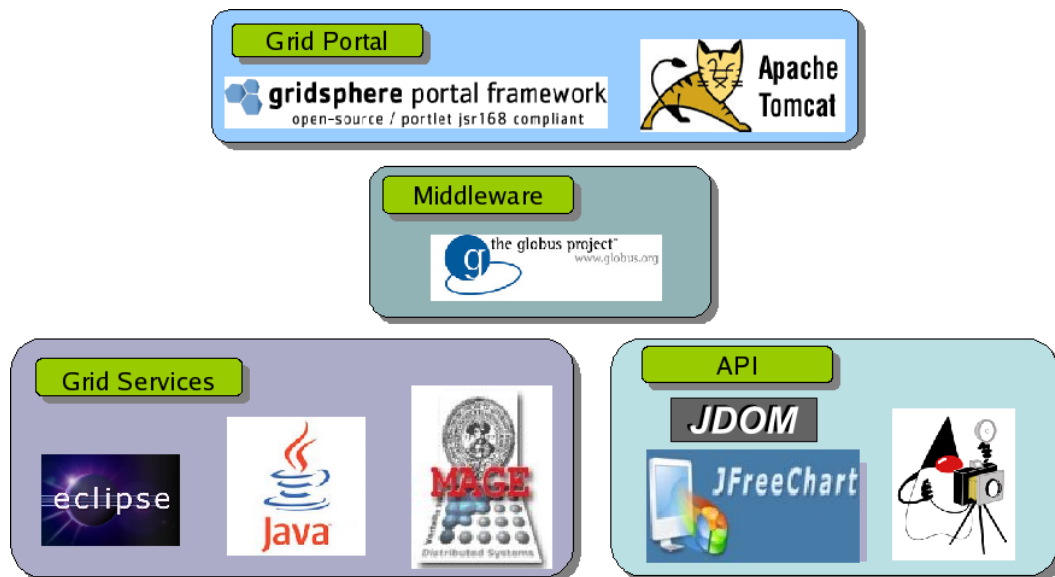


Figure 3.2: Grid Portal Software Infrastructure

3.2 Communication Interfaces

The communications interfaces to accomplish the grid portal functionality are described as follows:

- The end-user through a web browser accesses the grid portal via HTTP (Hypertext Transfer Protocol).
- Communication between portlet and grid service is made through SOAP (Simple Access Object Protocol) messages.
- The data image transference is managed via GridFTP (Grid File Transfer Protocol), the client FTP of Globus.

3.3 Grid Services Implementation and Deployment

There are five steps to create and put in operation a grid service (see Figure 3.3).

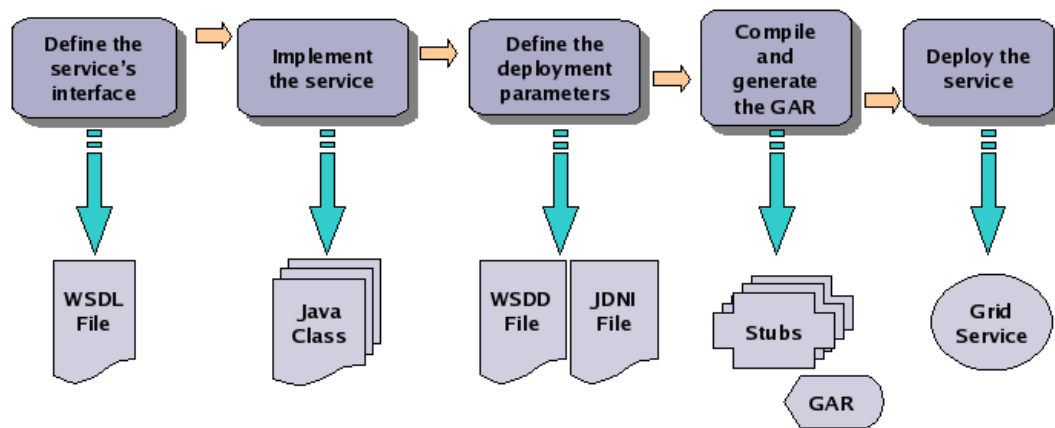


Figure 3.3: Five Steps to Create and Deploy a Grid Service

- *Define the Service Interface:* The service interface specifies what the service does. This service specification is built using the Web Service Description Language (WSDL), and XML-based language. This WSDL file contains a list of grid attributes and methods that would characterize a grid service. Specifically a WSDL file must have four elements: definitions, port-types, messages and types.

- *Definitions*: Provide the name of the service and the target name-space, which is a URI (Uniform Resource Identifier) that groups similar service functionalities.
 - *Port-types*: Embraces the specification of attributes and operations that the service will provide. Each operation specification has its name as well as the input and output messages, indicating the messages to pass at invocation and returning, respectively.
 - *Messages*: Once the port-types have been defined the operations messages must be defined as well.
 - *Types*: The messages are defined by elements that represent parameters of the operations. The types of these parameters must be also defined. The input and output parameters of the operations, are wrapped into complex type elements that define the sequence of types corresponding to each one of them.
- *Implement the Service*: The service implementation specifies how the service does, what it says it does. The operation functionalities are implemented in programming language, such as Java or C++.
 - *Define the Deployment Parameters*: To set the grid service deployment parameters in the host environment server, two files are used: the Web Service Deployment Descriptor (WSDD) and Java Naming and Directory Interface (JDNI).
 - *Compile and Generate the GAR File*: Grid Archive (GAR) is a packed file that contains all the structure needed in order to deploy the service into the service container. This structure contains the previously mentioned files: service interface (in WSDL), service implementation (in Java), deployment descriptors (in WSDD and JDNI), and other complementary files. These complementary files can be automatically generated using Ant and involves the following steps.
 - Processing the WSDL file to add missing pieces (such as bindings).

- Creating the stubs classes from the WSDL.
 - Compiling the stubs classes.
 - Compiling the service implementation.
 - Organizing all the files into a very specific directory structure.
- *Deploy the Service*: Deploying the service into the service container enables its remote invocation. For this case, during the deployment process is used Globus toolkit and Ant. With these tools the GAR file is unpacked and the WSDL, stubs, compiled implementation and descriptors are copied into the Globus directory structure.

3.4 Portlets and Services Integration

Deployed grid services can be executed by end-users through its URI. A command line approach implies previous knowledge about the service localization and its nontrivial invocation. The GridSphere portal framework offers a transparent way for the execution of our grid services. For each grid service we have implemented one portlet, which can be seen as the client application that invokes that service. Through this portlet the user interacts submitting their request (for example an image to operate), and once the service processes it, the response is rendered on the portlet interface.

In Figure 3.4 shows the portlet architecture that help us clarify the portlet functionality within a portlet web application and its user interaction. In our project a set of image processing portlets belong together to a portlet web application. The portlets configuration is determined by a set of important files: *group.xml*, that indicates the grouping of portlets that would be part of a portlet web application; *portlet.xml*, that indicates how each portlet would be deployed in the portlet container, giving its portlet class name and class localization; *layout.xml*, indicates the portlet organization in the portlet web application visual interface. A portlet implements the *model-view-controller (MVC)* pattern design, which separates the source

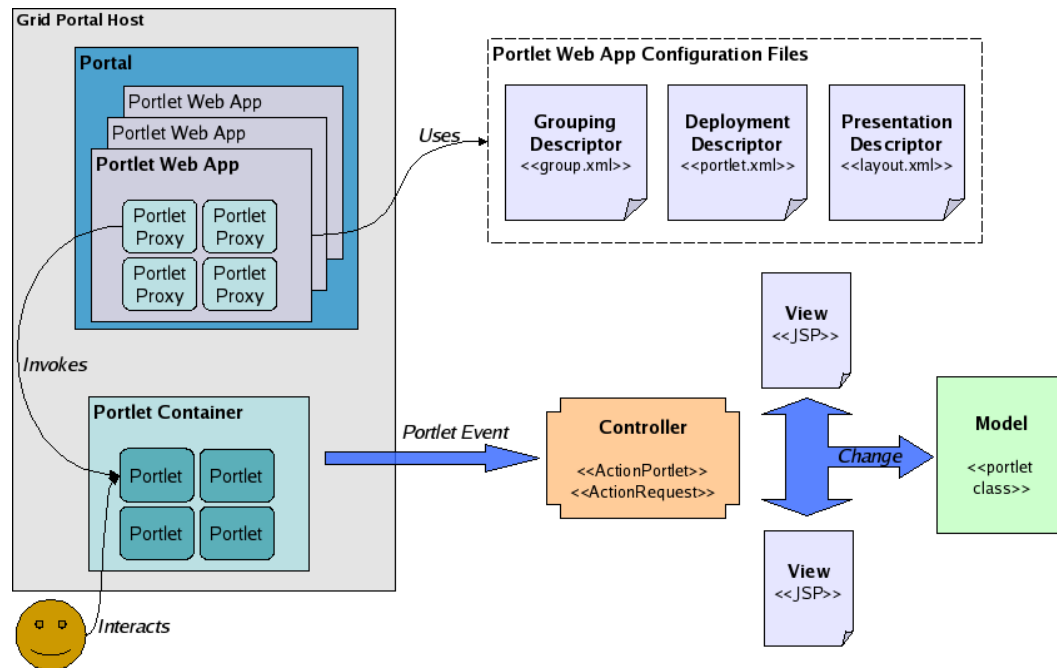


Figure 3.4: Portlet Architecture

code among functionality (model), interface (view) and its connection (controller), in order to make it modular and facilitating its reuse. The portlet functionality was implemented in Java and the visual interface in JSP.

3.5 User Interfaces

Either grid portal administrators or end-user may access the grid portal to the interface illustrated in Figure 3.5. There are a set of steps that an end-user must follow to execute an image processing operator from a portlet:

- From a web browser enter the URL of the grid portal.
- The user must enter their username and password to be authorized.
- Once the user has entered to their session, she/he will be able to choose the image processing operators web portlet application. A list of operators portles is displayed.
- The user chooses the portlet required, upload their image from local host through a file browser button provided in the portlet.

The image shows the main page of the WALSAIP Portal. At the top, there is a banner with an aerial view of a coastal area and the text 'WALSAIP Portal'. Below the banner are navigation links: 'PDC Lab Home', 'WALSAIP Home', and 'UPRM Home'. A secondary navigation bar contains 'Grid Portal', 'PDC Lab', 'Applications', 'Data', and 'Documents'. The main content area is divided into several sections. On the left, there is a 'WALSAIP Project' section with a description and two bullet points. Below that is the 'WALSAIP Grid Portal' section with a description. On the right, there is a 'Login' form with fields for 'User Name' and 'Password', a 'Remember my login' checkbox, and buttons for 'Login', 'Create new account', and 'Forgot your password?'. The footer contains logos for PDC Lab, WALSAIP, and Universidad of Puerto Rico.

Figure 3.5: Grid Portal Main Page

- The user triggers the action to apply the operator. The data image is transferred from grid portal host to grid service host via GridFTP. This is made by the URI service invocation. The grid service takes the original data image and executes its operator function, resulting an operated data image. The operated data image is transferred from grid service host to grid portal host via GridFTP.
- The resulting operated data image is rendered in the portlet visual interface.
- The operated data image can be downloaded to the local host through a download button in the portlet.

3.6 About the Appendices

To clarify the issues related to configuration and implementation we have included four appendices in this report. Appendix A provides guidelines to create and deploy a grid service. Appendix B describes the portlets integration step by step. Appendix C provides a user's guide for the portal interface. Finally, Appendix D provides the source code for the implementation of the portlets.

CHAPTER 4

CONCLUSION AND FUTURE WORK

In this project report, we have presented the development issues of a *grid portal* that provides access to a framework of distributed signal processing operators. The current grid portal prototype provides services to access environmental data and perform signal processing operators over these data sets.

Future work include:

1. Fully automation of the system where developers may register their version of signal processing operators and make then available to end users in a transparent way.
2. A methodology to compose signal processing operators efficiently according to resource constraints and the metadata associated to operators.
3. A fault tolerance mechanism to make available a service in the case that the host environment crashed. A solution can create replicas in several nodes having a index services system. Another solution could be send and deploy the GAR service file when a fail is detected.
4. A performance evaluation mechanism to assess the possible transfered rate to the host environment in order to decide if the data image must have packed to its size reduction. Similary, a mechanism to evaluate the resource availability of the host environment and transfer the data image to a optimal host is needed.

APPENDICES

APPENDIX A

CREATING AND DEPLOYING A GRID SERVICE

This appendix gives the guidelines to create and deploy a grid service. The image processing operator that we use as example is the *invert operator*. The grid service creation will be implemented with the help of the MAGE plug-in for Eclipse. This plug-in generates all its support files from a Java service definition. The following steps must be carried out into the Eclipse environment.

1. Configure MAGE plug-in for Eclipse

- Go to *Help—Software Updates—Find and Install—Search for New Features to Install*.
- *New Remote Site*. *Set Name*: GDT-MAGE and *URL*: <http://mage.unimarburg>.
- Once the \$SERVICES_PROJECT has been created: right-click over the project directory structure and go to *Properties* menu following: *Java Build Path—Libraries—Add library—User Library—User Libraries...—New... de/eclipse/*.
- Set the name of the library and add the jars.

2. Define Methods and Attributes

Define methods and attributes.

- private String inImage → Image path to operate.
- public String getDescription () → Get the description of the operator.
- public void setInImage (String inImage) → Set the image to operate.
- public void applyOp () → Apply the operator to the image.

- `public String getOutImage ()` → Get the path of the operated image.

3. Java Project Creation

- Go to File menu and follow *New Project—Java Project*, a wizard window will be deployed (see Figure A.1).

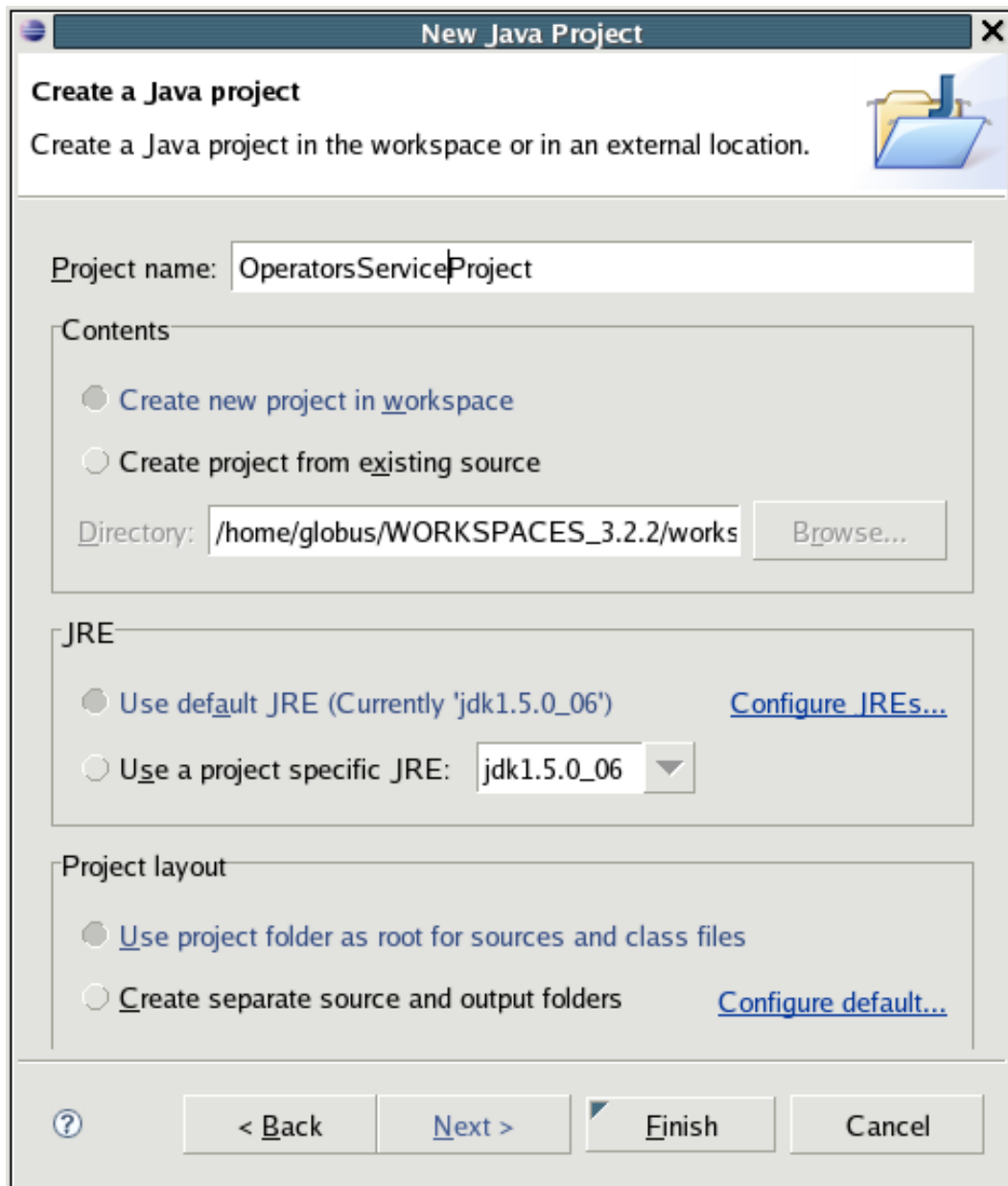


Figure A.1: New Java Project Wizard

- As *Project name*: OperatorServicesProject (\$SERVICES_PROJECT), that will be the directory container for our grid services (this name will not affect any settings about the services).
- Set Java compiler compliance to 5.0.
- Set Project layout to Create separate source and output folders.

4. Grid Service Creation

- Into the \$SERVICES_PROJECT structure, go to File menu and follow *New—Other—MAGE-GDT Grid Service*. A wizard window will be deployed.
- Set Select platform to GT4 (Figure A.2).

5. Grid Service Programming

Now we must program the code for our grid service, according to the attributes and methods previously defined (step 1). A distinction between grid methods and nongrid methods must be specified. Similarly for grid attributes. A grid method will have the @GridMethod prefix and a grid attribute the @GridAttribute.

- Edit the Java class

\$SERVICES_PROJECT/edu/uprm/grid/operators/invertop/InvertOp.java,
according to step one.

```
package edu.uprm.grid.operators.invertop;

import java.io.File;
import javax.media.jai.JAI;
import javax.media.jai.PlanarImage;
import de.fb12.gdt.GridService;
import de.fb12.gdt.GridAttribute;
import de.fb12.gdt.GridMethod;

@GridService (name = "InvertOp",
    namespace = "http://uprm.edu/ns/grid/operators/invertop",
    targetPackage = "edu.uprm.grid.operators.invertop",
    serviceStyle = "SSTYLE_FACTORY",
    resourceStyle = "RSTYLE_MAGE",
```

Create a GT4 Service
Create a GT4 Service in the current project.

Service name

GT4 Service Settings

Service Namespace

Annotated Package

Annotated Class

Service Style:

Resource Style:

Load Service on container start

Package Settings

Use default service package
 Use custom service package

Service Package

Figure A.2: New Grid Service Wizard

```

operationProvider = "GetRPPProvider",
loadOnStartup = false,
hotLoadable = false,
securityDesc = "[]")

public class InvertOp {
    @GridAttribute private String inImage;
    private String outImage;
    private String desc;
    private String nameImage;
    private static final String TEMP_DIR = "/tmp/";
    private static final String PREFIX = "invert_";

    public void setDescription(){
        desc = "\n Operator Invert"
            +"\n *****"
            +"\n Description : This operator allow invert an image"
            +"\n "
            +"\n\n Structure:"
            +"\n     operatorImage( \"Op_Invert\", \" \");"
            +"\n\n where:"
            +"\n     InvertOp : is the name of the class.;"
        }//end setDescription

    @GridMethod public String getDescription(){
        this.setDescription();
        return desc;
    }//end method

    public String getInImage() {
        return inImage;
    }//end getDescription

    @GridMethod public void setInImage(String inImage) {
        this.inImage = inImage;
    }//end setInImage

    @GridMethod public String getOutImage() {
        return outImage;
    }//end getOutImage

```

```

public void setOutImage(String outImage) {
    this.outImage = outImage;
}

}

@GridMethod public void applyOp(){
    //get the name of image
    File imageFile = new File(this.getInImage());
    nameImage = imageFile.getName();
    //Read the image.
    String inImage = this.getInImage();
    PlanarImage input = JAI.create("fileload", inImage);
    //Invert the image.
    PlanarImage output = JAI.create("invert", input);
    //Saves the image to the given path and filename as a JPEG image
    String filepath = TEMP_DIR + PREFIX + nameImage;
    saveImage(output, filepath, "JPEG");
}

}

/**
 *Saves the image to the given path and filename using the given codec
 *
 *@param filepath the path and filename to save the image to.
 *@param type The JAI-defined codec type to save as.
 */
public void saveImage(PlanarImage output, String filepath, String type) {
    //Saves the image to the given path and filename
    JAI.create("filestore", output, filepath, type, null);
    this.setOutImage(filepath);
}

}

}

```

We can see within the structure of \$SERVICES_PROJECT that several directories and files have been automatically generated. Such files include the service client into the *client* directory, and the classes to manage several instances into the *impl* directory.

6. Stubs and GAR File Generation

Stubs are the classes that represent the service in the client side, and GAR is the packed file that contains all the files that compose a grid service. In our

`$$SERVICES_PROJECT` we see that some errors are reported. This is due to missing services files. To solve this problem.

- Go to task bar and press the *Generate Stubs* button.
- Go to task bar and press the *Package Services* to generate the GAR file. Our generated GAR would be *edu_uprm_grid_operators_invertop_InvertOp.gar* located at `$$SERVICES_PROJECT/InvertOp/`.

7. Deploy the Service

Now the Gar file is ready to be deployed in the service container of Globus. This deployment allow to copy and configure a set of files into strategic directories of Globus. There are two ways to deploying:

- From Eclipse: right-click and go to *Deploy GAR* over gar file located in the `$$SERVICES_PROJECT` structure.
- From a shell console: `cd $$SERVICES_PROJECT/InvertOp/` and execute the command: *globus-deploy-gar edu_uprm_grid_operators_invertop_InvertOp.gar*.

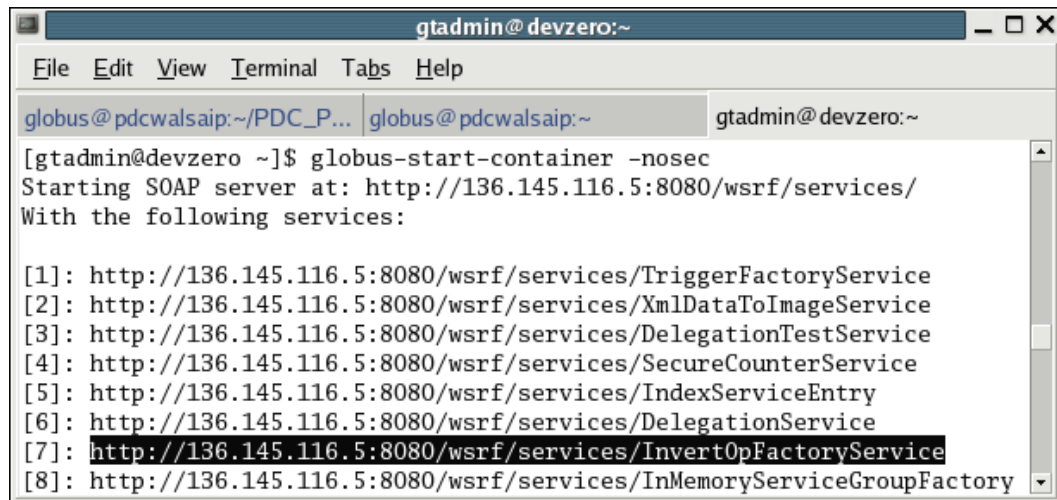
8. Make a Grid Service Client

The client class invokes the grid service functionality. This class contains the stubs, the URI's of the service, and the remote invocation service methods. This class is automatically generated by the MAGE-GDT plug-in, and is located at `$$SERVICES_PROJECT/InvertOp/src/edu/uprm/grid/operators/invertop/client/InvertOpClient.java`.

9. Test the Grid Service

- Before running the service client class,copy `$$GLOBUS_LOCATION/clientconfig.wsdd` to our `$$SERVICES_PROJECT` root.
- Next, start the Globus services container from shell console: *globus-start-container -nosec* (see Figure [A.3](#)).

- From Eclipse into the client class: (*right-click*)—*Run As...*—*Run...*—*Java Application*—(*double-click*). A wizard window is deployed.
- Go to *Classpath*—*User Entries*—*Advanced*—*Add External Folder*—(*browse \$GLOBUS_LOCATION*)—*OK*—*Apply*—*Run*.



```
gtadmin@ devzero:~  
File Edit View Terminal Tabs Help  
globus@pdcwalsaip:~/PDC_P... globus@pdcwalsaip:~ gtadmin@ devzero:~  
[gtadmin@devzero ~]$ globus-start-container -nosec  
Starting SOAP server at: http://136.145.116.5:8080/wsrf/services/  
With the following services:  
[1]: http://136.145.116.5:8080/wsrf/services/TriggerFactoryService  
[2]: http://136.145.116.5:8080/wsrf/services/XmlDataToImageService  
[3]: http://136.145.116.5:8080/wsrf/services/DelegationTestService  
[4]: http://136.145.116.5:8080/wsrf/services/SecureCounterService  
[5]: http://136.145.116.5:8080/wsrf/services/IndexServiceEntry  
[6]: http://136.145.116.5:8080/wsrf/services/DelegationService  
[7]: http://136.145.116.5:8080/wsrf/services/InvertOpFactoryService  
[8]: http://136.145.116.5:8080/wsrf/services/InMemoryServiceGroupFactory
```

Figure A.3: Invert Operator Service Deployed in Globus Container

APPENDIX B

PORTLET INTEGRATION

This Appendix describes the steps required to integrate grid services and portlets once the service has been implemented.

1. Create the Portlet Project in Gridsphere

- From a shell console into the $\$GRIDSPHERE_HOME$ execute the command line *ant new-project*. A set of steps configuration will be deployed.
- Set *Project Title* to *Operators Portlets*.
- Set *Project Name* to *operatorsportlets*.
- Indicate that the project follows the *jsr standard*.
- Finally, into the $\$GRIDSPHERE_HOME/projects$, an *operatorsportlets* directory, which will contain the portlet structure will be created. We will denominate *operatorsportlets* as $\$PORTLET_DIRECTORY$.

2. Integrate the Portlet Project to Eclipse

- Go to *File—New—Project—Java Project*, choose *Create project from existing source* pointing to $\$GRIDSPHERE_HOME/projects/\$PORTLET_DIRECTORY$.

3. Programming the Portlet

- Copy
 $\$SERVICES_PROJECT/InvertOp/src/edu/uprm/grid/operators/invertop/client/$
InvertOpClient

to

`$PORTLET_DIRECTORY/src/edu/uprm/operatorsportlets/portlets/`

- In `$PORTLET_DIRECTORY/src/edu/uprm/operatorsportlets/portlets/` create the class `InvertOpPortlet.java` (see Appendix D).
- In `$PORTLET_DIRECTORY/webapp/jsp/` create the JSP file: `index.jsp` (see Appendix D)

4. Edit the Portlet Web Application Files

- Go to `$PORTLET_DIRECTORY`, and edit the files: `portlet.xml`, `layout.xml` and `group.xml` located in `$PORTLET_DIRECTORY/webapp/WEB-INF` directory.
- `portlet.xml`:

```
<portlet>
    <!-- place portlet description here -->
    <description xml:lang="en">Invert Portlet</description>
    <!-- place unique portlet name here -->
    <portlet-name>InvertOpPortlet</portlet-name>
    <display-name xml:lang="en">A Invert Portlet</display-name>
    <!-- place your portlet class name here -->
    <portlet-class>edu.uprm.operatorsportlets.portlets.InvertOpPortlet</portlet-class>
    <expiration-cache>1</expiration-cache>
    <!-- place supported modes here -->
    <supports>
        <mime-type>text/html</mime-type>
        <portlet-mode>config</portlet-mode>
        <portlet-mode>edit</portlet-mode>
        <portlet-mode>help</portlet-mode>
    </supports>
    <supports>
        <mime-type>text/wml</mime-type>
        <portlet-mode>edit</portlet-mode>
        <portlet-mode>help</portlet-mode>
    </supports>
    <supported-locale>en</supported-locale>
    <portlet-info>
```

```

        <title>A Invert Portlet</title>
        <short-title>Invert</short-title>
        <keywords>invert</keywords>
    </portlet-info>
    <!-- place portlet preferences here -->
    <portlet-preferences>
        <preference>
            <name>myPref</name>
            <value>avalue</value>
            <read-only>true</read-only>
        </preference>
    </portlet-preferences>
</portlet>

```

- layout.xml:

```

<portlet-tab label="inverttab">
    <title lang="en">::Invert::</title>
    <table-layout>
        <row-layout>
            <column-layout>
                <portlet-frame label="invert">
                    <portlet-class>edu.uprm.operatorsportlets.portlets.InvertOpPortlet</portlet-class>
                </portlet-frame>
            </column-layout>
        </row-layout>
    </table-layout>
</portlet-tab>

```

- group.xml:

```

<portlet-role-info>
    <portlet-class>edu.uprm.operatorsportlets.portlets.InvertOpPortlet</portlet-class>
    <required-role>USER</required-role>
</portlet-role-info>

```

5. Create the Portlet Structure

- In `$PORTLET_DIRECTORY/src/` create the package: `edu.uprm.operatorsportlets.portlets`.
- In this package create the class: `InvertOpPortlet.java`.

6. Set the Grid Service Libraries

- Copy
`$GLOBUS_LOCATION/lib/edu_uprm_grid_operators_invertop_InvertOp*` to
`$GRIDSPHERE_HOME/projects/operatorsportlets/lib/`.
- Copy `$GLOBUS_LOCATION/lib/*.jar` to `$PORTLET_DIRECTORY/lib/`.

7. Deploy the Portlet

- In a shell console go to: `$PORTLET_DIRECTORY/`.
- Execute the command line: `ant deploy`.

8. Deploy Globus in Tomcat Server

This step is needed for posterior portlets deployment.

- From a shell console execute the command:
`ant -f $GLOBUS_LOCATION/share/globus_wsrp-common/tomcat/tomcat`
`-Dtomcat.dir=$CATALINA_HOME/ deployTomcat`

9. Set the Portlet to Work with Globus

- Copy
`$CATALINA_HOME/webapps/wsrp/WEB-INF/lib/` to `$PORTLET_DIRECTORY/lib/`.
- Copy
`$GLOBUS_LOCATION/client-config.wsdd` to
`$CATALINA_HOME/webapps/operatorportlets/lib/WEB-INF/classes/`.

10. Set the External Libraries

- In `$JAVA_HOME/jre/ext/lib/` put the other external libraries required for the portlet functionality. In our example these are the JDOM jars, JFreeChart jars, and JAI jars.

11. Test the Portlet

- In a shell console execute the command: `$CATALINA_HOME/bin/startup.sh`.
- From a web browser go to: `$SERVER_NAME:port/gridsphere/`.

APPENDIX C USER'S GUIDE

This section presents a user's guide for the grid portal.

- From a web browser go to: *http://proc.ece.uprm.edu:8080/gridsphere/*, or the *\$GRIDPORTAL_HOST* where the grid portal has been installed. The grid portal main page is deployed.
- To be identified enter the *username* and *password* from the grid portal main page. The user session is activated and the grid portal environment is deployed.
- The portlet web application of the set of image processing operators are deployed. Choose among these operators portlets to apply a required operator (see Figure C.1).

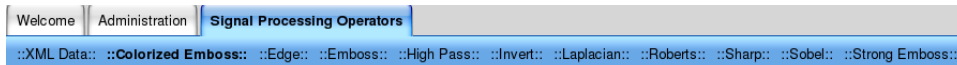


Figure C.1: Portlet Web Application

- Once a operator portlet has been selected (see Figure C.2) a data image from local host can be uploaded following the steps:
 - *Browse* button, a browse file window is shown, select the data image.
 - *Upload Image* button to upload the data image to grid portal server.
- Apply the operator over the data image through *Apply Operator* button. The apply operator process is activated. Then it is rendered (see Figure C.3). There are other services associated to the service invocation. These services are:

The screenshot displays the WALSaip Portal interface. At the top, there is a banner with an aerial view of a port and the text "WALSaip Portal". To the right of the banner, a "Logout" link is visible, along with a welcome message: "Welcome, Administrator User". Below the banner, there are navigation links for "PDCLab Home", "WALSaip Home", and "UPRM Home".

The main content area features a navigation bar with "Welcome", "Administration", and "Signal Processing Operators". Below this, there are tabs for "XML Data", "Edge", and "Invert". The "Invert" tab is active, displaying a window titled "A Invert Portlet".

The "A Invert Portlet" window contains the following elements:

- Image Selection:** A section with an "Image:" label, a text input field, "Browse..." and "Upload Image" buttons, and an "Apply Operator" button. A "Choose Another Image" button is located below this section.
- Operator Description:** A text area containing the following information:
 - Operator Invert
 - *****
 - Description : This operator allow invert an image
 - Structure:
operatorImage("Op_Invert", " ");
 - where:
InvertOp : is the name of the class.
- Image Properties:** A section with a message: "The operator has not been applied".
- Image Histogram:** A section with a message: "The operator has not been applied".
- Image Result:** A section with a message: "The operator has not been applied".

Figure C.2: Invert Portlet

- The Image Properties that shows the original and operated data image properties which are its name, dimension and size.
- The Image Histogram that presents the original and operated data image histograms. This indicates the number of image intensity values by each intensity values range.
- The Image Result that displays the original and operated images. It is possible to save the operated image through *Save Image* button.
- Choose other data image to new operator application through *Choose Another Image* button.

WALSAP Portal

Logout
Welcome,
Administrator User

PDCLab Home | WALSAP Home | UPRM Home

Welcome Administration **Signal Processing Operators**

XML Data Edge Invert

A Invert Portlet

THE OPERATOR WAS SUCCESSFUL

Image Selection

Image: Browse... Upload Image

Apply Operator

Preview:

Choose Another Image

Operator Description:

Operator Invert
.....
Description : This operator allow invert an image

Structure:
operatorimage("Op_Invert", " ");

where:
InvertOp : is the name of the class.

Image Properties

Original Image:

Name: oe_puerto_rico_hi-res.jpg
Size: 695279
Width: 827
Height: 1169

Operated Image:

Name: invert_oe_puerto_rico_hi-res.jpg
Size: 213462
Width: 827
Height: 1169

Image Histogram

Each Bin stores the number of pixels samples of a image whose values lie within a given range.

Operated Image:

Pixels per Bin

Number of Pixels Samples

256 Bins

Pixels per Bin

Number of Pixels Samples

256 Bins

Image Result

Original image:

Operated image:

Opening invert_oe_puerto_rico_hi-res.jpg

You have chosen to open
invert_oe_puerto_rico_hi-res.jpg
which is a: JPEG image
from: http://proc.ece.uprm.edu:8080

What should Firefox do with this file?

Open with Image Viewer (default)

Save to Disk

Do this automatically for files like this from now on.

Cancel OK

Download Image

Figure C.3: Operator Grid Service Performed

APPENDIX D

PROGRAMMING THE PORTLET

- InvertOpPortlet.java

```
package edu.uprm.operatorsportlets.portlets;

import java.io.File;
import java.rmi.RemoteException;

import javax.portlet.ActionRequest;
import javax.portlet.PortletConfig;
import javax.portlet.PortletContext;
import javax.portlet.PortletException;
import javax.portlet.RenderRequest;
import javax.xml.rpc.ServiceException;

import org.apache.axis.types.URI.MalformedURIException;
import org.gridlab.gridisphere.provider.event.jsr.ActionFormEvent;
import org.gridlab.gridisphere.provider.event.jsr.FormEvent;
import org.gridlab.gridisphere.provider.event.jsr.RenderFormEvent;
import org.gridlab.gridisphere.provider.portlet.jsr.ActionPortlet;
import org.gridlab.gridisphere.provider.portletui.beans.ActionSubmitBean;
import org.gridlab.gridisphere.provider.portletui.beans.FileInputBean;
import org.gridlab.gridisphere.provider.portletui.beans.ImageBean;
import org.gridlab.gridisphere.provider.portletui.beans.MessageBoxBean;
import org.gridlab.gridisphere.provider.portletui.beans.MessageStyle;
import org.gridlab.gridisphere.provider.portletui.beans.TableCellBean;
import org.gridlab.gridisphere.provider.portletui.beans.TextAreaBean;
import org.gridlab.gridisphere.provider.portletui.beans.TextBean;

public class InvertOpPortlet extends ActionPortlet {

    // for Invert operator service
    private static final String OP_SERVICE_HOST = "devzero.ece.uprm.edu";
```

```

private static final String OP_INSTANCE_URI = "http://136.145.116.5:8080/wsrf/services/InvertOpService";
private static final String OP_FACTORY_URI = "http://136.145.116.5:8080/wsrf/services/InvertOpFactoryService";
//for ImageInfo service
private static final String IMGINFO_SERVICE_HOST = "pdcgrid-32-01.ece.uprm.edu";
private static final String IMGINFO_INSTANCE_URI = "http://136.145.116.6:8080/wsrf/services/ImageInfoService";
private static final String IMGINFO_FACTORY_URI =
    "http://136.145.116.6:8080/wsrf/services/ImageInfoFactoryService";
// for portal host
private static final String PORTAL_HOST = "proc.ece.uprm.edu";
private static final String PUBLIC_IMG_DIR = "/home/mmendoza/opt/tomcat/webapps/images/";
private static final String WEB_IMG_DIR = "http://proc.ece.uprm.edu:8080/images/";
private static final String DISPLAY_PAGE = "index.jsp";
private static final String TEMP_DIR = "/tmp/";
private static final String PREFIX = "invert_";
private static final int MSG_ALERT = 0;
private static final int MSG_ERROR = 1;
private static final int MSG_INFO = 2;
private static final int MSG_SUCCESS = 3;
private boolean isAppliedOperator = false;
private boolean showPreview = false;
private boolean isUploadImage = false;
private ImageBean imgbPreview;
private ImageBean imgbOriginal;
private ImageBean imgbOperated;
private String nameImageOriginal;
private String nameImageOperated;
private InvertOpClient opClient;
private ImageInfoClient imgInfoClient;

public void init(PortletConfig config) throws javax.portlet.PortletException {
    super.init(config);
        PortletContext context = config.getPortletContext();
        context.log("****In display portlet****");
        DEFAULT_VIEW_PAGE = "prepare";
    }//end init method

public void prepare(RenderFormEvent event) throws PortletException {
    RenderRequest request = event.getRenderRequest();
        System.err.println("context path = " + request.getContextPath());
        //invoke the service
        this.invokeServices();

```

```

        //render the fields
this.renderFields(event);
// calls the jsp page
        setNextState(request, DISPLAY_PAGE);
} //end prepare

public void invokeServices(){
try {
opClient = new InvertOpClient(OP_FACTORY_URI, OP_INSTANCE_URI, false);
//imgInfoClient = new ImageInfoClient(IMGINFO_FACTORY_URI, IMGINFO_INSTANCE_URI, false);
} catch (Exception e) {
        e.printStackTrace();
}
} //end invokeService()

private void displayMessage(FormEvent event, String beanId, int style, String message){
    MessageBoxBean msgbox = event.getMessageBoxBean(beanId);
    switch(style){
    case MSG_ALERT:    msgbox.clearMessage();
        msgbox.deleteCssStyle();
        msgbox.setMessageType(MessageStyle.MSG_ALERT);
        break;
    case MSG_ERROR:    msgbox.clearMessage();
        msgbox.deleteCssStyle();
        msgbox.setMessageType(MessageStyle.MSG_ERROR);
        break;
    case MSG_INFO:     msgbox.clearMessage();
        msgbox.deleteCssStyle();
        msgbox.setMessageType(MessageStyle.MSG_INFO);
        break;
    case MSG_SUCCESS:  msgbox.clearMessage();
        msgbox.deleteCssStyle();
        msgbox.setMessageType(MessageStyle.MSG_SUCCESS);
        break;
    } //end switch
    msgbox.setValue(message);
} //end displayMessage

//***** Render Methods *****/

private void renderFields(FormEvent event){

```

```
this.renderDescription(event);
this.renderPreview(event);
this.renderProperties(event);
this.renderHistogram(event);
this.renderOriginalImage(event);
this.renderOperatedImage(event);
} //end renderFields

private void renderDescription(FormEvent event){
String descriptionOp = "";
TextAreaBean textarea = null;
try {
descriptionOp = opClient.getDescription();
} catch (RemoteException e) {
e.printStackTrace();
}
textarea = event.getTextAreaBean("textDescription");
textarea.setCols(50);
    textarea.setRows(10);
    textarea.setCssStyle("background:#FFFFFF;");
    textarea.setValue(descriptionOp.trim());
    textarea.setReadOnly(true);
} // end renderDescription

private void renderPreview(FormEvent event){
TextBean textPreview = event.getTextBean("textPreview");
if(showPreview==true){
if(isUploadImage==true){
textPreview.setVisible(true);
textPreview.setValue("Preview:");
imgbPreview = event.getImageBean("imagePreview");
imgbPreview.setVisible(true);
imgbPreview.setSrc(WEB_IMG_DIR + nameImageOriginal);
imgbPreview.setHeight("175");
imgbPreview.setWidth("350");
} //end if
else{
textPreview.setVisible(false);
imgbPreview = event.getImageBean("imagePreview");
imgbPreview.setVisible(false);
} //end else
}
```

```

} //end if
} //end renderPreview method

private void renderProperties(FormEvent event){
String image = "";
TableCellBean tcell;
TextAreaBean textarea;
String properties = "";
String message = "";
TextBean textOr = event.getTextBean("textPropertiesOr");
TextBean textOp = event.getTextBean("textPropertiesOp");
if(isAppliedOperator == true){
if(isUploadImage == true){
try{
// ---- Original image ----
image = PUBLIC_IMG_DIR+nameImageOriginal;
// invoke ImageInfo
imgInfoClient = new ImageInfoClient(IMGINFO_FACTORY_URI, IMGINFO_INSTANCE_URI, false);
imgInfoClient.setInImage(image);
textOr.setValue("Original Image:");
tcell = event.getTableCellBean("cellPropertiesOr");
textarea = new TextAreaBean();
textarea.setCols(50);
        textarea.setRows(6);
        textarea.setCssStyle("background:#FFFFFF;");
        properties = "Name:  "+nameImageOriginal+"\n"+
                "Size:  "+imgInfoClient.getSize()+"\n"+
                        "Width: "+imgInfoClient.getWidth()+"\n"+
                        "Height: "+imgInfoClient.getHeight()+"\n";
        textarea.setValue(properties);
        textarea.setReadOnly(true);
        tcell.addBean(textarea);
        properties = "";
// ---- Operated image ----
image = PUBLIC_IMG_DIR+nameImageOperated;
// invoke ImageInfo
imgInfoClient = new ImageInfoClient(IMGINFO_FACTORY_URI, IMGINFO_INSTANCE_URI, false);
imgInfoClient.setInImage(image);
textOp.setValue("Operated Image:");
tcell = event.getTableCellBean("cellPropertiesOp");
textarea = new TextAreaBean();

```

```

textarea.setCols(50);
    textarea.setRows(6);
    textarea.setCssStyle("background:#FFFFFF;");
    properties = "Name:    "+nameImageOperated+"\n"+
        "Size:    "+imgInfoClient.getSize()+"\n"+
            "Width: "+imgInfoClient.getWidth()+"\n"+
            "Height: "+imgInfoClient.getHeight()+"\n";
    textarea.setValue(properties);
    textarea.setReadOnly(true);
    tcell.addBean(textarea);
    properties = "";
} catch (RemoteException e) {
e.printStackTrace();
} catch (MalformedURLException e) {
// TODO Auto-generated catch block
e.printStackTrace();
} catch (ServiceException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
}
} //end if
} //end if
else{
textOr.setVisible(false);
textOp.setVisible(false);
message = "The operator has not been applied";
this.displayMessage(event, "warningProperties", MSG_INFO, message);
} //end else
} //end renderProperties method

private void renderHistogram(FormEvent event){
ImageBean imgb;
String histogram = "";
String inImage = "";
String outImage;
String command;
String message = "";
TextBean textOr = event.getTextBean("textHistogramOr");
TextBean textOp = event.getTextBean("textHistogramOp");
if(isAppliedOperator == true){
if(isUploadImage == true){

```

```

try{
// ---- Original image ----
// image is yet in portal host in temp dir
inImage = TEMP_DIR+nameImageOriginal;
// transfer the original image form portal host to service host
command = "globus-url-copy "
+ "gsiftp://" + PORTAL_HOST + ":2811/" + inImage + " "
+ "gsiftp://" + IMGINFO_SERVICE_HOST + ":2811/" + inImage;
ExecuteProcess.exec(command, true);
// set the original image on the service host
imgInfoClient = new ImageInfoClient(IMGINFO_FACTORY_URI, IMGINFO_INSTANCE_URI, false);
imgInfoClient.setInImage(inImage);
// apply the operator on the service host
imgInfoClient.createHistogram();
// get the path operated image from service host
outImage = imgInfoClient.getOutImage();
// transfer the operated image from service host to portal host
command = "globus-url-copy "
+ "gsiftp://" + IMGINFO_SERVICE_HOST + ":2811/" + outImage + " "
+ "gsiftp://" + PORTAL_HOST + ":2811/" + outImage;
ExecuteProcess.exec(command, true);
    //copy the operated image from temp to web dir
    command = "cp " + outImage + " " + PUBLIC_IMG_DIR;
ExecuteProcess.exec(command, true);
// set the jsp
textOr.setValue("Original Image:");
imgb = event.getImageBean("imgHistogramOr");
imgb.setSrc(WEB_IMG_DIR+(new File(outImage)).getName().trim());
imgb.setWidth("600");
imgb.setHeight("400");
imgb.setVisible(true);
// ---- Operated image ----
// image is yet in portal host in temp dir
inImage = TEMP_DIR+nameImageOperated;
// transfer the original image form portal host to service host
command = "globus-url-copy "
+ "gsiftp://" + PORTAL_HOST + ":2811/" + inImage + " "
+ "gsiftp://" + IMGINFO_SERVICE_HOST + ":2811/" + inImage;
ExecuteProcess.exec(command, true);
// set the original image on the service host
imgInfoClient = new ImageInfoClient(IMGINFO_FACTORY_URI, IMGINFO_INSTANCE_URI, false);

```

```

imgInfoClient.setInImage(inImage);
// apply the operator on the service host
imgInfoClient.createHistogram();
// get the path operated image from service host
outImage = imgInfoClient.getOutImage();
// transfer the operated image from service host to portal host
command = "globus-url-copy "
+ "gsiftp://" + IMGINFO_SERVICE_HOST + ":2811/" + outImage + " "
+ "gsiftp://" + PORTAL_HOST + ":2811/" + outImage;
ExecuteProcess.exec(command, true);
    //copy the operated image from temp to web dir
    command = "cp " + outImage + " " + PUBLIC_IMG_DIR;
ExecuteProcess.exec(command, true);
// set the jsp
textOr.setValue("Operated Image:");
imgb = event.getImageBean("imgHistogramOp");
imgb.setSrc(WEB_IMG_DIR + (new File(outImage)).getName().trim());
imgb.setWidth("600");
imgb.setHeight("400");
imgb.setVisible(true);
//set the messagebox
message = "Each Bin stores the number of pixels samples of a image whose values lie within a given range.";
this.displayMessage(event, "infoHistogram", MSG_INFO, message);
} catch (RemoteException e) {
e.printStackTrace();
} catch (MalformedURLException e) {
// TODO Auto-generated catch block
e.printStackTrace();
} catch (ServiceException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
}
} //end if
} //end if
else{
textOr.setVisible(false);
textOp.setVisible(false);
message = "The operator has not been applied";
this.displayMessage(event, "warningHistogram", MSG_INFO, message);
} //end else
} //end renderHistogram method

```



```
private void renderOriginalImage(FormEvent event){
    TextBean text = event.getTextBean("textOriginalImg");
    String message = "";
    if(isAppliedOperator == true){
        if(isUploadImage == true){
            text.setVisible(true);
            text.setValue("Original image:");
            imgbOriginal = event.getImageBean("imageOriginal");
            imgbOriginal.setVisible(true);
            imgbOriginal.setSrc(WEB_IMG_DIR + nameImageOriginal);
            imgbOriginal.setHeight("500");
            imgbOriginal.setWidth("500");
        }//end if
        else{
            text.setVisible(false);
            imgbOriginal = event.getImageBean("imageOriginal");
            imgbOriginal.setVisible(false);
        }//end else
    }//end if
    else{
        message = "The operator has not been applied";
        this.displayMessage(event, "warningResult", MSG_INFO, message);
    }//end else
 }//end renderOriginalImage method
```

```
private void renderOperatedImage(FormEvent event){
    TextBean text = event.getTextBean("textOperatedImg");
    if(isAppliedOperator == true){
        if(isUploadImage == true){
            //set the download form
            ActionSubmitBean submit = event.getActionSubmitBean("download");
            submit.setAction("doActionDownloadImage");
            submit.setValue("Download Image");
            submit.setVisible(true);
            text.setVisible(true);
            text.setValue("Operated image:");
            imgbOperated = event.getImageBean("imageOperated");
            imgbOperated.setVisible(true);
            imgbOperated.setSrc(WEB_IMG_DIR + nameImageOperated);
            imgbOperated.setHeight("500");
```

```

imgbOperated.setWidth("500");
} //end if
else{
text.setVisible(false);
imgbOperated = event.getImageBean("imageOperated");
imgbOperated.setVisible(false);
} //end else
} //end if
} //end renderOperatedImage method

//***** doAction Methods *****

public void doActionPreviewImage(ActionFormEvent event) throws PortletException{
ActionRequest request = event.getActionRequest();
showPreview = true;
isAppliedOperator = false;
ActionSubmitBean submit = event.getActionSubmitBean("download");
    submit.setVisible(false);
this.renderFields(event);
setNextState(request, DISPLAY_PAGE);
} //end doActionPreviewImage method

public void doActionUploadImage(ActionFormEvent event){
ActionRequest request = event.getActionRequest();
boolean thereIsImage = false;
String message;
    try{
        //int fsize;
        FileInputBean finput = event.getFileInputBean("fileImage");
        String fname = finput.getFileName();
        if (fname.trim() != ""){
            thereIsImage = true;
            String nameImgUpload = (new File (fname)).getName().trim();
            nameImageOriginal = nameImgUpload;
                fname = fname.substring(fname.lastIndexOf("/") + 1);
                fname = fname.substring(fname.lastIndexOf("\\") + 1);
                //save the image upload in the temp dir
            finput.saveFile(TEMP_DIR + nameImageOriginal);
            String path = TEMP_DIR + nameImageOriginal;
            //copy the image upload from temp to web dir

```

```

        String command = "cp "+path+" "+PUBLIC_IMG_DIR;
ExecuteProcess.exec(command, true);
    }//end if
} catch (Exception e) {
e.printStackTrace();
}
if(thereIsImage==true){
    isUploadImage = true;
showPreview = true;
isAppliedOperator = false;
}
else{
isUploadImage = false;
showPreview = false;
isAppliedOperator = false;
message = "Please upload an image";
this.displayMessage(event, "message", MSG_ERROR, message);
};//end else
    this.renderFields(event);
    setNextState(request, DISPLAY_PAGE);
};//end doActionUploadImage method

public void doActionDownloadImage(ActionFormEvent event){
ActionRequest request = event.getActionRequest();
ImageBean imgb = event.getImageBean("imageOperated");
String image = imgb.getSrc();
// pathImage: PUBLIC_IMG_DIR + nameImageOperated;
String path = PUBLIC_IMG_DIR;
//setFileDownloadEvent(request, image, path);
setFileDownloadEvent(request, nameImageOperated, path);
this.renderFields(event);
setNextState(request, DISPLAY_PAGE);
};//end doActionDownloadImage method

public void doActionAnotherImage(ActionFormEvent event){
ActionRequest request = event.getActionRequest();
String command = "";

command = "rm " + PUBLIC_IMG_DIR+nameImageOriginal + "";
ExecuteProcess.exec(command, true);
command = "rm " + PUBLIC_IMG_DIR+nameImageOperated + "";

```

```

ExecuteProcess.exec(command, true);
command = "rm "+TEMP_DIR+nameImageOriginal + " ";
ExecuteProcess.exec(command, true);
command = "rm "+TEMP_DIR+nameImageOperated + " ";
ExecuteProcess.exec(command, true);

imgbPreview.setSrc("");
if (isAppliedOperator==true){
imgbOriginal.setSrc("");
imgbOperated.setSrc("");
}
isAppliedOperator = false;
showPreview = false;
isUploadImage = false;
nameImageOriginal = "";
nameImageOperated = "";
ActionSubmitBean submit = event.getActionSubmitBean("download");
    submit.setVisible(false);
this.renderFields(event);
setNextState(request, DISPLAY_PAGE);
} //end doActionAnotherImage method

public void doActionApplyOperator(ActionFormEvent event){
ActionRequest request = event.getActionRequest();
String inImage = TEMP_DIR+nameImageOriginal;
String outImage = "";
String command = "";
String message = "";
if(isUploadImage == true){
//transfer the original image form portal host to service host
command = "globus-url-copy "
+ "gsiftp://" +PORTAL_HOST+":2811/" +inImage+ " "
+ "gsiftp://" +OP_SERVICE_HOST+":2811/" +inImage;
ExecuteProcess.exec(command, true);
try {
//set the original image on the service host
opClient.setInImage(inImage);
//apply the operator on the service host
opClient.applyOp();
//get the path operated image from service host
outImage = opClient.getOutImage();

```

```

//transfer the operated image from service host to portal host
command = "globus-url-copy "
+ "gsiftp://" + OP_SERVICE_HOST + ":2811/" + outImage + " "
+ "gsiftp://" + PORTAL_HOST + ":2811/" + outImage;
ExecuteProcess.exec(command, true);

//copy the operated image from temp to web dir
command = "cp " + outImage + " " + PUBLIC_IMG_DIR;
ExecuteProcess.exec(command, true);
nameImageOperated = (new File(outImage)).getName().trim();
} catch (RemoteException e) {
e.printStackTrace();
}
isUploadImage = true;
showPreview = true;
isAppliedOperator = true;
message = "The Operator was successful";
this.displayMessage(event, "message", MSG_SUCCESS, message);
} //end if
else{
isUploadImage = false;
showPreview = false;
isAppliedOperator = false;
message = "Please upload an image";
this.displayMessage(event, "message", MSG_ERROR, message);
} //end else
this.renderFields(event);
setNextState(request, DISPLAY_PAGE);
} //end doActionApplyOperator

} //end class

```

- index.jsp

```

<%@ taglib uri="/portletUI" prefix="ui" %>
<%@ taglib uri="http://java.sun.com/portlet" prefix="portlet" %>
<%@ taglib uri="/portletAPI" prefix="portletAPI" %>

<portletAPI:init/>
<portlet:defineObjects/>

<ui:messagebox beanId="message"/>

```

```

<!-- ////////////////////////////////// Image Selection ////////////////////////////////// -->
<ui:group label="Image Selection">
  <table>
    <tr>
      <td>
        <ui:group>
          <ui:frame>
            <ui:tablerow>
              <ui:tablecell align="center">
                <ui:text cssStyle="font-style:bold">Image:</ui:text>
                <ui:fileform action="doActionUploadImage">
<ui:fileinput beanId="fileImage" size="50"/>
<ui:actionsubmit action="doActionUploadImage" key="Upload Image"/>
                </ui:fileform>
              </ui:tablecell>
            </ui:tablerow>
            <ui:tablerow>
              <ui:tablecell align="center">
                <ui:form>
                <ui:actionsubmit action="doActionApplyOperator" value="Apply Operator"/>
                </ui:form>
              </ui:tablecell>
            </ui:tablerow>
            <ui:tablerow>
              <ui:tablecell align="center">
                <ui:text beanId="textPreview"/>
              </ui:tablecell>
            </ui:tablerow>
            <ui:tablerow >
              <ui:tablecell align="center">
                <ui:image beanId="imagePreview"/>
              </ui:tablecell>
            </ui:tablerow>
            <ui:tablerow>
              <ui:tablecell align="center">
                <ui:form>
                <ui:actionsubmit action="doActionAnotherImage" value="Choose Another Image"/>
                </ui:form>
              </ui:tablecell>
            </ui:tablerow>
          </ui:frame>
        </ui:group>
      </td>
    </tr>
  </table>

```

```

    </ui:group>
</td>

<td>
    <ui:group>
<ui:frame>
    <ui:table>
    <ui:tablecell>
    <ui:text cssStyle="font-style:bold">Operator Description:</ui:text>
    </ui:tablecell>
    </ui:table>
    <ui:table>
    <ui:tablecell>
    <ui:textarea beanId="textDescription"/>
    </ui:tablecell>
    </ui:table>
    </ui:frame>
</ui:group>
</td>
</tr>
</table>
</ui:group>
<!-- //////////////// Image Properties //////////////// -->
<ui:group label="Image Properties">
    <ui:messagebox beanId="warningProperties"/>
    <ui:panel cols="50%, 50%" align="center">
    <ui:frame>
    <ui:table>
    <ui:tablecell>
    <ui:text beanId="textProperties0r" cssStyle="font-style:bold"/>
    </ui:tablecell>
    </ui:table>
    <ui:table>
    <ui:tablecell beanId="cellProperties0r"/>
    </ui:table>
    </ui:frame>
    <ui:frame>
    <ui:table>
    <ui:tablecell>
    <ui:text beanId="textProperties0p" cssStyle="font-style:bold"/>
    </ui:tablecell>

```



```

<!-- ////////////////////////////////// Image Result ////////////////////////////////// -->
<ui:group label="Image Result">
  <ui:messagebox beanId="warningResult"/>
  <ui:panel cols="50%, 50%" align="center">
    <ui:frame>
      <ui:tablerow>
        <ui:tablecell>
<ui:text beanId="textOriginalImg" cssStyle="font-style:bold"/>
          </ui:tablecell>
        </ui:tablerow>
        <ui:tablerow>
          <ui:tablecell>
<ui:image beanId="imageOriginal" />
            </ui:tablecell>
          </ui:tablerow>
        </ui:frame>
      <ui:frame>
        <ui:tablerow>
          <ui:tablecell>
<ui:text beanId="textOperatedImg" cssStyle="font-style:bold"/>
            </ui:tablecell>
          </ui:tablerow>
          <ui:tablerow>
            <ui:tablecell>
<ui:form>
              <ui:image beanId="imageOperated"/>
              <ui:actionssubmit beanId="download"/>
            </ui:form>
          </ui:tablecell>
        </ui:tablerow>
      </ui:frame>
    </ui:panel>
  </ui:group>

```

REFERENCE LIST

- [1] J. Villamizar, M. Paredes, and D. Rodríguez. La estructura algebraica del espacio de seales unidimensionales. *Revista Integración. Escuela de Matemáticas Universidad Industrial de Santander*, 23(2):15–39, 2005.
- [2] L. Bautista. Web-base data processing for environmental surveillance monitoring applications. Master’s thesis, University of Puerto Rico, 2006.
- [3] S. Del Fabbro. Developing a distributed image processing and management framework. Technical report, University of Adelaide, 2000.
- [4] D. Stow, M. Tsou, and L. Guo. Web-based remote sensing applications and java tools for environmental monitoring. Technical report, Online Journal of Space Communication, No. 3, 2003. <http://satjournal.tcom.ohiou.edu/Issue03/applications.html>.
- [5] A. Smith. A data and image processing toolbox for nano-world: The computer supported cooperative learning environment on nanophysics. Technical report, Department of Micro Engineering, University of Applied Sciences Biel, 2003.
- [6] Open grid forum. <http://www.ogf.org>.
- [7] I. Foster. What is the grid? a three point checklist. *Grid Today*, 1(6):22, 2002. <http://www.globus.org/alliance/publications/papers.php>.
- [8] J. Treadwell. Open grid services architecture – glossary of terms. Technical report, Global Grid Forum, 2005.
- [9] C. Fellenstein, J. Joseph, and M. Ernest. Evolution of grid computing architecture and grid adoption models. *IBM Systems Journal*, 43(4):624–645, 2004. <http://www.research.ibm.com/journal/sj/434/josepaut.html>.

- [10] C. MacKenzie, K. Laskey, F. McCabe, P. Brown, and R. Metz. Reference model for service oriented architecture 1.0. Technical report, OASIS Standards, 2006. <http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf>.
- [11] D. Talia. The open grid services architecture: Where the grid meets the web. *IEEE Internet Computing*, 06(6):67–71, 2002. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1067739.
- [12] B. Sotomayor. The globus toolkit 4 programmer’s tutorial, 2004. <http://gdp.globus.org/gt4-tutorial/>.
- [13] I. Foster. Globus toolkit version 4: Software for service-oriented systems. In *IFIP International Conference on Network and Parallel Computing*, volume 3779 of *Lecture Notes in Computer Science*, pages 2–13. Springer-Verlag, 2005.
- [14] Y. Cai, J. Cao, M. Li, and L. Chen. Portlet-based portal design for grid systems. In *International Workshop on Collaborative Virtual Research Environments (CVRE 2006)*, pages 571–575, 2006.
- [15] M. Thomas, J. Burruss, L. Cinquini, G. Fox, D. Gannon, L. Gilbert, G. von Laszewski, K. Jackson, D. Middleton, R. Moore, M. Pierce, B. Plale, A. Rajasekar, R. Regno, E. Roberts, D. Schissel, A. Seth, and W. Schroeder. Grid portal architectures for scientific applications. *Journal of Physics: Conference Series*, page 5, 2005. <http://grids.ucs.indiana.edu/ptliupages/publications/>.
- [16] Myproxy: Credential management service.
- [17] C. Wege. Portal server technology. *IEEE Internet Computing*, 06(3):73–77, 2002.
- [18] Jsr 168 portlet specification. <http://jcp.org/en/jsr/detail?id=168>.
- [19] Oasis web services for remote portlets (wsrp).
- [20] Commodity grid (cog) kits. <http://wiki.cogkit.org>.
- [21] Gridsphere portal framework. <http://www.gridsphere.org>.

- [22] Gridport toolkit. <http://gridport.net>.
- [23] Geosciences network (geon). <http://www.geongrid.org>.
- [24] Linked environments for atmospheric discovery (lead).
- [25] Oceans and climate digital library portal.
- [26] Sura coastal ocean observing and prediction (scoop).
- [27] E. Kourpas. Grid computing: Past, present and future. Technical report, IBM Corporation, 2006. http://www-03.ibm.com/grid/grid_literature.shtml.
- [28] S. Hashimi. Service-oriented architecture explained, 2004.
- [29] I. Foster, H. Kishimoto, A. Savva, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, J. Treadwel, and J. Von Reich. The open grid services architecture, version 1.5. Technical report, Open Grid Forum, July 24 2006. <http://www.ogf.org/gf/docs/?final>.
- [30] K. Amin, M. Hategan, G. Laszewski, and N. Zaluzec. Abstracting the grid. In *Proceedings of the 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP 2004)*, pages 250–257, A Coruña, Spain, February 2004.
- [31] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard. Web services architecture, w3c working group note 11 february 2004. *World Wide Web Consortium*, 2004.
- [32] I. Foster. A globus primer (draft v0.6). Technical report, Globus Alliance, 2006.
- [33] M. Aldinucci, M. Coppola, M. Danelutto, M. Vanneschi, and C. Zoccolo. ASSIST as a research framework for high-performance Grid programming environments. In O. Rana J. Cunha, editor, *Grid Computing: Software environments and Tools*. Springer, January 2006. To appear.
- [34] G. Laszewski, J. Gawor, S. Krishnan, and K. Jackson. *Grid Computing: Making the Global Infrastructure a Reality*, chapter Commodity Grid Kits - Middleware for Building Grid Computing Environments, pages 639–656. Communications

- Networking and Distributed Systems. Wiley, 2003.
- [35] S. Vickers. Portal standards: The answer to portal interoperability?, 2005. <http://java.sys-con.com/read/47686.htm>.
- [36] A. Akram, D. Chohan, X. Wang, X. Yang, and R. Allan. A service oriented architecture for portals using portlets. In *Uk e-Science All Hands Conference 2005*, September 2005. <http://pubs.doc.ic.ac.uk/portal-portlets-soa/>.
- [37] J. Novotny, M. Russell, and O. Wehrens. Gridsphere: An advanced portal framework. In *EUROMICRO '04: Proceedings of the 30th EUROMICRO Conference (EUROMICRO'04)*, pages 412–419. IEEE Computer Society, 2004.
- [38] M. Pierce. Open grid computing environments (ogce) annual report. Technical report, The OGCE Portal Toolkit, 2006.
- [39] C. Zhang, I. Kelley, and G. Allen. Grid portal solutions: A comparison of gridportlets and ogce. *Workshop on Grid Computing Portals (GCE 2005) of Concurrency and Computation: Practice and Experience*, 19(12):1739 – 1748, 2006. www.cct.lsu.edu/~gallen/Preprints/CS_Zhang05a.pre.pdf.
- [40] G. von Laszewski. Grid computing: Enabling a vision for collaborative research. In *Conference on Applied Parallel Computing, 3rd CSC Scientific Meeting*, Espoo, Finland, 2002. Springer. citeseer.ist.psu.edu/vonlaszewski02grid.html.
- [41] I. Foster. The anatomy of the grid: Enabling scalable virtual organizations. In *CCGRID '01: Proceedings of the 1st International Symposium on Cluster Computing and the Grid*, page 6, Washington, DC, USA, 2001. IEEE Computer Society.
- [42] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, and C. Kesselman. Grid service specification, 2002.
- [43] F. Patin. An introduction to digital image processing, 2003.

- [44] Hypermedia image processing reference (university of edinburgh).
http://www.cee.hw.ac.uk/hipr/html/hipr_top.html.
- [45] Clarens grid-enabled web services framework.
- [46] Apache ant. <http://ant.apache.org/>.
- [47] Apache ant. <http://ant.apache.org/>.

BIOGRAPHICAL SKETCH

Mariana Mendoza-Botero was born in October 2, 1980, in Cali, Colombia. Mariana is the daughter of María Beatriz Botero-Duque and Carlos Arturo Mendoza-Lenis. In May 2005 she received her Bachelor in Systems Engineering from University of Valley at Cali Campus, Colombia. Since August 2005 she is pursuing a master degree in Computer Engineering at University of Puerto Rico at Mayagüez Campus.