Towards a Generic Framework for Deploying Applications as Grid Services

Simona Arustei, Mitica Craus and Alexandru Archip

Department of Computer Science and Engineering, "Gh. Asachi" Technical University
Bd. D. Mangeron, Iasi, Romania

Abstract. Exploiting the power of the grid very often involves transforming existing or new applications into grid services. In this paper we present a generic framework based on a service oriented architecture developed in order to simplify the task of deploying applications as grid services. Our work consists of a configurable grid service that provides application developers with a high level programming model, hiding the complexity of dealing with web services and grid technologies. The architectural design of the framework allows custom functionality to be plugged into an adaptive grid service in a simple manner, thus attracting more non-expert users to the grid. A prototype implementation of the framework has been built and a case study has been developed to illustrate the concept.

1 Introduction and Related Work

Functional grids across many universities represent the proof that grid technologies are making great advances and present a wide-spread acceptance since many initial installation and configuration issues have been resolved. Still, the difficulty to develop and/or use already existing applications causes an evident lack of users on those grids as most scientists do not concentrate their research effort on gaining a specialized insight on grid technologies although their access to grid computing should be facilitated as it not only provides maximum available data/computing resources, but also shows powerful ability on some critical issues, such as security, load balancing and fault tolerance. Grid services provide several key features such as statefull services, notifications, and uniform authentication and authorization across different administrative domains. The focus of this paper is to describe the development, as well as possible uses and extensions, of a generic framework based on a service oriented architecture that can be used to provide grid access at a higher level, abstracting the details of the grid middleware from the end user. The proposed framework is based on a configurable grid service and has an architectural design that allows custom functionality to be plugged into this adaptable grid service in a simple manner.

There has been done work trying to provide the scientists with more attractive alternatives to grid interaction. For example, [1] presents a generic job submission service that provides a standardized, high-level set of functions allowing for easy job submissions and access to grid resources, but the user is still communicating directly

Arustei S., Craus M. and Archip A. (2008).
Towards a Generic Framework for Deploying Applications as Grid Services.
In *Proceedings of the 2nd International Workshop on Architectures, Concepts and Technologies for Service Oriented Computing*, pages 17-26
DOI: 10.5220/0001898800170026

with individual resources. Other efforts concentrated in migrating different technologies to the grid, but in the spectrum of specific narrow applicative domains. In [4], for example, the problem of automatic deployment of CCM (CORBA Component Model) and MPI (Message Passing Interface) applications is addressed, while the authors in [6] describe the migration of Web Services to a core grid middleware by converting them to OGSI (Open Grid Services Infrastructure) compliant grid services.

Other considerable efforts [2], [3] were concentrated on abstracting the grid middleware from the user/developer. The Grid Application Toolkit[2], for example, defines a platform-independent API to grid resources and services, focusing on resource management (job submission and migration), data management (access to files and pipes), event management (application monitoring and control), and information management (application-specific meta data). The GEMSS project [3] provided an interoperable grid middleware for medical services applications built on common Grid standards. For this, a generic grid service provision framework was developed trying to hide the complexity of transforming existing medical applications into grid services.

The remainder of the paper is organized as follows: section 3 presents the architecture of the proposed framework and gives a detailed description of its components, section 4 describes a case study developed as proof of concept, followed in section 5 by conclusions and issues that we will address in future development of the framework.

2 Architecture of the Generic Grid Service Framework

The architecture of our framework is based on three components: the client component, the service component and the work execution unit (Fig. 1). Each component is a configurable one such that the user is able to customize the functionality of the framework. The whole grid execution process is driven through the client component. This module allows the user to interact with the framework in two ways: first, by specifying or providing the components of the application needed to be deployed and the input data for it and second, by driving a set of dynamic parameters for the application through the interaction with a result visualization area. The generic service component controls the execution of the provided application components on the working units and can also be responsible with executing some pre- and post-processing of the work.

The collaboration between the three components is notification-based allowing the communication of input and output data in a service-oriented fashion. If needed it can also be achieved through the use of file staging.

The application to be deployed as a grid service is specified in a modular fashion and can be composed of up to three modules: the data partitioning module, the job execution module and the result composition module. For the development of each module we provide an easy to use Java interface and each module can extend a built-in class that hides the internal workings and details of the grid.

The input for the data partitioning module is represented by the number of units that will execute the actual work and should provide the input data for these components.

The job execution module is responsible for launching the execution of the actual work on the working nodes. The working nodes can all execute the same code or can each have a different functionality. There can be one or more such work execution units and the hosts on which they run can be specified through the client component or can be determined by the service component.

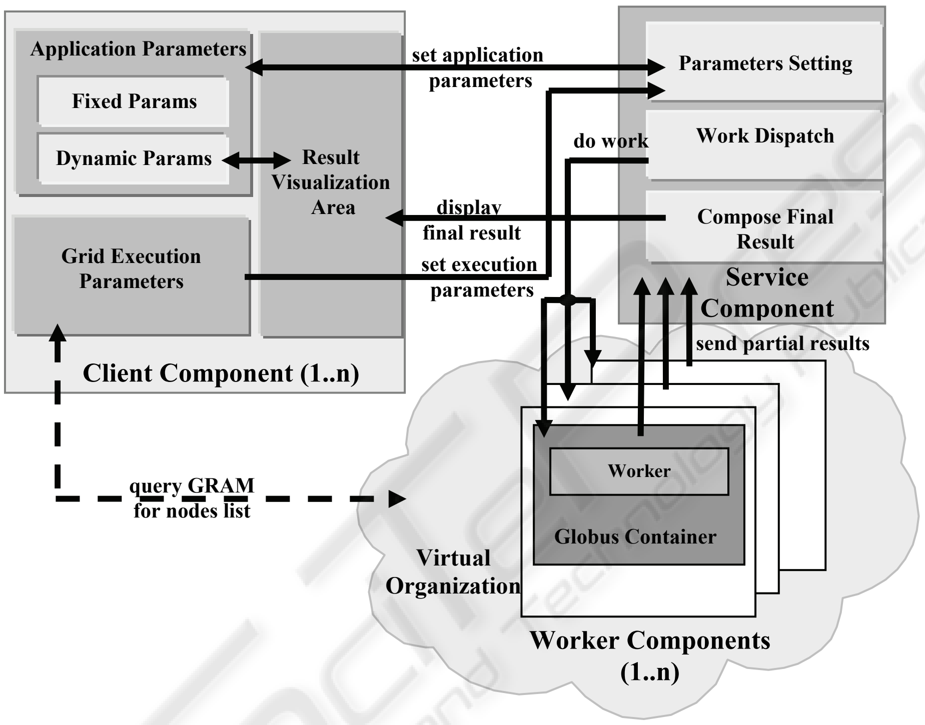


Fig. 1. Architecture of the generic framework.

If the partial results produced by the work execution units need a final stage of refinement or composition, this can be done through the means of the result composition module.

In the following we will discuss in more detail the three components of the framework, their interaction and customization.

2.1 The Client Component

This component is a Java application that allows the user to interact with the framework and to explore the results of the specified processing. The interaction between the client component and the service is described by two types of parameters:

- parameters that drive the execution of the framework on the grid

- application specific parameters

The former parameters express the information related to the grid execution of the requested processing and consist in:

- input data set that needs to be processed - the user specifies the location of the data set that needs to be transferred to the host executing the partitioning module;

- the data partitioning module - represents a custom Java module that divides the input data set between work execution units, employing either a static or a dynamic load balancing scheme;

- the job execution module(s) - custom Java module(s) that will be executed by the workers. This module represents the actual functionality of the custom grid service and states the processing that each execution unit must perform on the allocated sub-set of data.

- the result composition module - custom Java modules that builds a final result of the parallel processing based on the partial results produced by the work execution units.

- number and type of work execution units - this is where the user can request the number of working nodes and can specify whether the working nodes will all execute the same job or will each have distinct functionality.

- work execution units hosts - optionally it can be specified through the client component the exact hosts that will execute the custom job modules.

There are two types of application specific parameters: fixed input parameters (these are set only once at first use of the service instance) and dynamic parameters (used to drive a loop in the jobs execution on the working nodes - any change in the values of these parameters triggers a new grid execution of the customized framework with the new values).

The results of the processing on the input data can be visualized in a result visualization area which can also be used to trigger new executions by changing the values of dynamic application parameters.

2.2 The Service Component

The generic service component is developed using Globus Toolkit 4 (GT4) [7] and runs inside a secured grid service container. In the grid system there can be only one such component but there might be a need for multiple client components to access it at the same time and either use the same or different customized functionalities. In order to solve this situation, the generic service component was designed using the factory-instance pattern. Using this pattern, a client can either create a new customized instance of the service or connect to an already running one. When the client needs the creation of a new instance, it will contact a fabric service that will manage the instantiation and initialization of a new resource. Because multiple resources need to be managed at the same time they are assigned a unique key needed for their identification. Thus, the fabric service will return an *endpoint reference* information (EPR) associated to a WS-resource (Web Services resource). The EPR will contain the URI (Uniform Resource Identifier) of the service as well as the resource key so that the client can invoke the service operations through the means of a customized instance service.

Resource Properties and Operations. Because the main requirement of the service is that it is reconfigurable and adaptive, the application related resource properties published generically express all the settings that need to be done in order to customize the functionality of the service:

Table 1. Application related resource properties exposed by the generic service component.

`inputDataSet`	An array of strings representing the location(s) of the input data set in the form of `host:pathToFile`
`partitionModule`	A string representing the location of the .jar file containing the classes that implement the partitioning module
`jobExecutionModule`	A string representing the location of the .jar file containing the classes that implement the job execution module(s)
`compositionModule`	A string representing the location of the .jar file containing the classes that implement the result composition module
`jobInputParameters`	An array of input parameters for the job execution module expressed generically in the form of `<paramName, paramType, paramValue, paramLength>`. These parameters need to comply with the input parameters specified in the classes implementing the job execution module.
`jobOutputParameters`	An array of output parameters filled in by the job execution module with the results of the processing. The parameters are specified in the same manner as above and each work execution unit produces a set of such parameters. They will be used to create the final output parameters of the service.
`serviceOutputParameters`	A set of output parameters specified in the same manner as above. These represent the final result sent to the client and are filled by the composition module based on the sets of `jobOutputParameters` produced by each worker.

In order to ensure the generality of the service the input and output parameters for the application to be deployed as a grid service need to be specified by the client component and can be of any type depending on the functionality of the application. Thus, our generic grid service allows the parameters to be specified in the form of parameter name (string) - parameter type (string) - parameter value (array of bytes) - parameter length (integer). All manipulated types of data are streamed to a byte array and vice versa.

The generic service component provides two types of operations:

- parameter related operations (`setJobInputParameter`, `setJobOutputParameter`, `getServiceOutputParameter`) and
- workflow related operations (`partitionData`, `launchJobs`, `composeResult`).

Launching Execution of Workers. The latter type of operations provided by our service use GRAM (Globus Resource Allocation Manager) [8] to launch jobs that actually trigger the execution of the provided modules for partitioning, job execution and result composition. The work is sent to the processing nodes via asynchronous threads. For this we have developed a `Dispatcher` class that implements the `GramJobListener` interface in order for the launcher to be notified about the status of the Gram job.

The job execution hosts can be specified by the client or can be chosen by the service component based on some minimal criteria requested by the client. Either way, the discovery of the nodes that can execute the jobs is done through the means of a `GramLocator` class that finds the Globus nodes defined for a specified virtual organization.

Transferring Results. The linking between the client, service and worker components is implemented as subscriptions and notifications. The partial results of the processing can either be written to files and pushed on the node performing the composition step or can be transferred by calling a service operation. In the first case the files are transferred to the composition node as part of the job launched on the work execution units. In the second case, the results of the processing are written by the worker by accessing an operation exposed by the generic service (`setJobOutput-Parameter`). Either way a notification is produced when the partial results have reached destination.

When there is a composition step performed by the service component, the finalization of this stage needs to be notified to the client component and this is implemented by publishing a resource property as a notification topic. The final result can then be transferred to the client either as a file or by accessing the `getOutput-Parameter` operation published by the generic service.

Security Issues. The framework is built to exploit the security mechanisms available in Globus Toolkit 4. These mechanisms allow restricted access to the generic service, including both authentication and authorization. The Grid Security Infrastructure (GSI) [9] was used to enable message-level security (GSI Secure Conversation) and full delegation was transferred to the service in order to allow it to act on the user's behalf. Also a proxy was transferred to the workers such that they are able to contact the service in order to transfer back the results of the processing.

2.3 The Worker Component

This component represents the application launched on a working node by the rendering service through GT4 GRAM. It was developed as a Java application as it acts as a client for the grid service. The generic service pushes the files needed for the host machines to act as clients for the grid service and prepares the environment by executing a script on the host. Once launched, the worker application processes its corresponding subset of the input data and either writes the results to a file which it transfers back to the service component or directly to the client (if there is no need for post-processing or it was chosen to be done by the client) or sends the result to the service by accessing the generic operations available.

3 Case Study: Sort-Last Parallel Rendering

A case study has been taken from the visualization domain, specifically Object Space Parallel Rendering, and has been used to demonstrate and test the framework. The need for real-time visualization of large data sets leads the way towards developing a Grid implementation of a parallel rendering pipeline.

Within Object Space parallel visualization, each node (individual unit of the parallel system, typically a single machine or processor) is responsible for the rendering of its block of data, irrespective of whether it may actually be visible at that precise moment. Object Space parallelization is also known as Sort-Last [5], reflecting the late stage in the graphics pipeline at which the graphics primitives are sorted from object-space into the resultant image-space (Fig. 2). Each node computes the values of the pixels for its associated sub-set and sends them to the compositing node which solves for the visibility of the pixels received from all processing nodes.

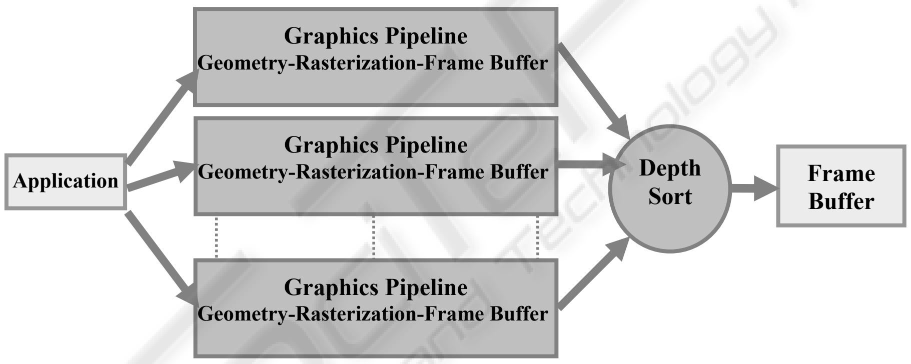

Application
Graphics Pipeline
Geometry-Rasterization-Frame Buffer
Graphics Pipeline
Geometry-Rasterization-Frame Buffer
Graphics Pipeline
Geometry-Rasterization-Frame Buffer
Depth Sort
Frame Buffer

Fig. 2. Sort-Last parallel rendering pipeline [5].

We developed our visualization service by customizing the generic framework previously presented and we depict the architecture of the customized framework in Fig. 3. The rendering process is driven and displayed through the client component. This module allows the user to interact with the visualization system in two ways: first, by specifying or providing the input data to be rendered and second, by driving the visualization parameters through the interaction with an OpenGL visualization area. The grid execution parameters and the visualization parameters are provided to the Render Service which controls the execution of the Render Workers that produce the images of the associated sub-sets of graphics data. The Render Service is also responsible with compositing the final image to be sent to the client, based on the partial results received from the rendering units. Both the client and the worker components act like clients for the Render Service. The three components of the visualization service will be discussed in more detail in the following.

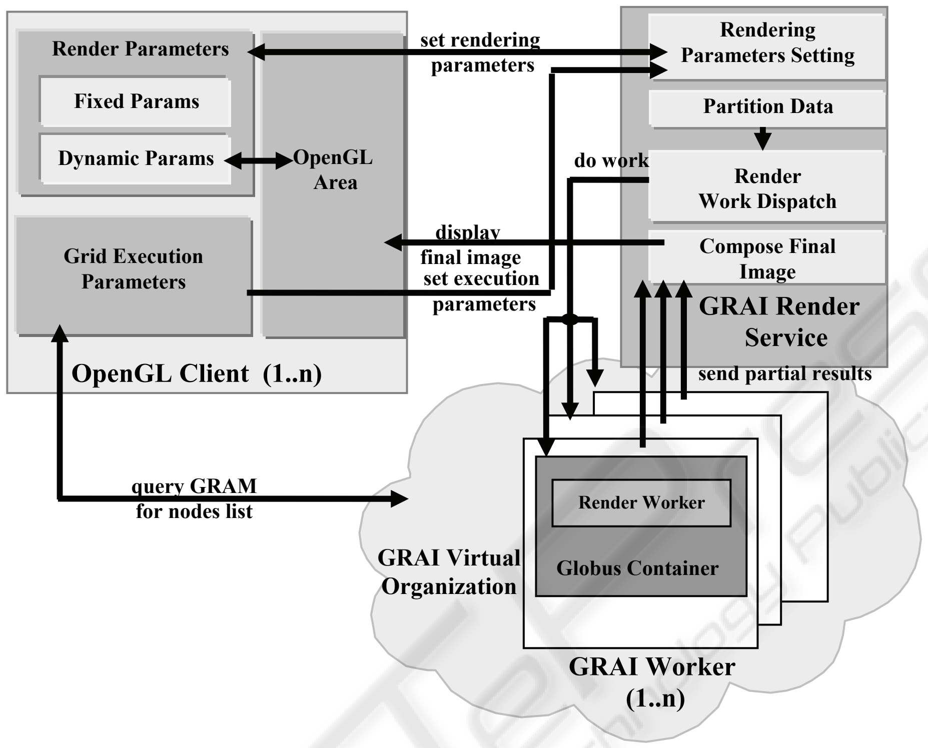
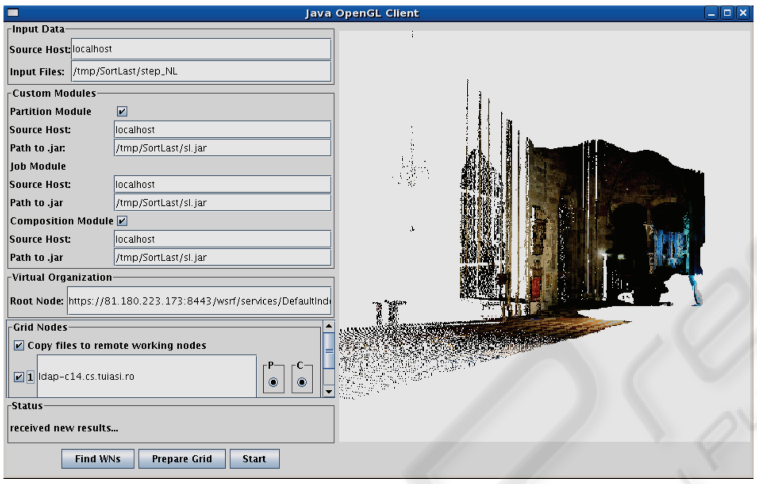


Fig. 3. Custom Visualization Service Architecture.

The three modules used to configure the generic service into a visualization service express the work that needs to be done to implement the Sort-Last parallel rendering scheme:
Data partitioning module - the input for this module is the path to the graphics primitives data set and it outputs a set of files with the primitives equally allocated to each working node.
Job execution module - this component effectuates an off-screen rendering of its allocated geometry and reads back the pixels along with depth information. The pixel and depth information are sent to the Render Service for compositing the final image.
Result composition module - this module is executed by the service component and sorts the pixels received from each worker based on the depth information, thus creating the image to be sent to the client.

The client component includes a Java application that uses OpenGL through the means of a wrapper library - JOGL (Java Binding for the OpenGL API) [10] - and allows the user to interact with the visualization system and to explore the results of the rendering. The interaction between the client component and the Render Service is described by the two types of parameters presented in section 3.1.

The grid execution parameters express the information related to the input data (location of the files containing the graphics primitives to be rendered), to the resolution and aspect of the resulting image and to the configuration of the parallel rendering pipeline (the user can specify the number of rendering nodes needed and can specify or choose the exact nodes that will execute the rendering).



Input Data
Source Host: localhost
Input Files: /tmp/SortLast/step_NL

Custom Modules
Partition Module ✓
Source Host: localhost
Path to .jar: /tmp/SortLast/sl.jar
Job Module
Source Host: localhost
Path to .jar /tmp/SortLast/sl.jar
Composition Module ✓
Source Host: localhost
Path to .jar /tmp/SortLast/sl.jar

Virtual Organization
Root Node: https://81.180.223.173:8443/wsrf/services/DefaultInd

Grid Nodes
✓ Copy files to remote working nodes
✓ 1 ldap-c14.cs.tuiasi.ro P ● C ●

Status
received new results...

Fig. 4. Visualizing the St. Stephan's Cathedral point cloud using Sort-Last parallel rendering executed on the customized framework.

The application specific parameters are needed to drive the rendering process by specifying modeling and visualization (rotation, translation, scaling etc) and projection transformations to be applied to the input data. These are dynamic parameters determined by the client application based on the user interaction with the OpenGL area. Any update on the rendering parameters triggers a new execution of the rendering pipeline on the working nodes.

The initialization of the visualization framework is triggered by the client application which causes the service component to partition the data set according to the number of working nodes specified through the client application and using the provided partitioning module. The Render Service sends the rendering jobs to the working nodes through the worker dispatcher. When the working nodes finish their rendering tasks, the Render Service begins the compositing stage of the final image by depth sorting the resulting pixels.

We tested our visualization service by rendering a point cloud acquired with a range scanning device (Fig. 4). The range scanners are capable of producing highly detailed point clouds, so, even though point primitives can be rendered simply and relatively fast, problems arise due to the often huge size of the datasets. Our test data is represented by a 3 million points scan (with color information) from the interior of St. Stephen's Cathedral in Vienna[1].

[1] Data provided by the Institute for Computer Graphics and Algorithms, TU Vienna, Austria.

4 Conclusions and Future Work

We have presented the vision, design and prototype implementation of a generic framework for deploying applications as grid services. Our approach simplifies the task of developing a grid service based on an existing or a new application, providing application developers with a high level programming model, hiding the complexity of dealing with web services and grid technologies. To test the proposed framework we developed a simple visualization service by customizing the generic components and we illustrated how an existing parallel rendering application can be deployed as a grid service. Whilst the current implementation of the framework has demonstrated the basic principles behind the architectural design, it represents a work in progress. Future research and development will address issues concerning resource monitoring and (re)scheduling, deployment of MPI applications, enabling inter-process communication during processing and providing Quality of Service support. Further development will also include adding built-in functionality to our service and creating an extension dedicated to grid visualization.

Acknowledgements

The research for this paper was supported by the Grant 74 CEEX-II03/31.07.2006.

References

1. Afgan, E., Jones, W. T.: Design, development and usage of a generic job submission grid service. In: Proc. of the 44th Annual Southeast Regional Conference, (Melbourne, Florida, March 10 - 12, 2006) 738-739 .
2. Allen G., Davis K., Goodale T., et. al.: The Grid Application Toolkit: Towards generic and easy application programming interfaces for the grid. In: Proc. of the IEEE, vol. 93, no. 3 (2005) 534-550.
3. Benkner, S., Berti, G., Engelbrecht, G., Fingberg, J., Kohring, G., Middleton, S. E., Schmidt, R.: GEMSS: Grid-infrastructure for Medical Service Provision. In: Methods of Information in Medicine, vol. 44, part 2 (2005) 177-181.
4. Lacour, S., Perez, C., Priol, T.: Generic Application Description Model: Toward Automatic Deployment of Applications on Computational Grids. In: Proc. of the 6th IEEE/ACM Int. Workshop on Grid Computing (Nov. 13 - 14, 2005). Int. Conf. on Grid Computing, IEEE Computer Society, Washington, DC, 284-287.
5. Molnar S., Cox M., Ellsworth D., Fuchs H.: A Sorting Classification of Parallel Rendering. In: IEEE Computer Graphics & Applications, vol. 14, no. 4 (1994) 23-32.
6. Parastatidis S., Watson P.: Experiences with Migrating myGrid Web Services to Grid Services, Global Grid Forum Workshop on Designing and Building Grid Services, (Chicago, Illinois, Oct. 8, 2003).
7. http://www.globus.org/toolkit/
8. http://www.globus.org/toolkit/docs/3.2/gram/ws/
9. http://www.globus.org/toolkit/docs/4.0/security/
10. https://jogl.dev.java.net/