Execution and Composition of E-Science Applications using the WS-Resource Construct

Evangelos Floros and Yannis Cotronis

National and Kapodistrian University of Athens
Dept. of Informatics and Telecommunications
Panepistimiopolis, Athens 15784, GREECE
{floros, cotronis}@di.uoa.gr

Abstract

Service Oriented Architectures are emerging as the recommended paradigm for developing dispersed e-science environments. In this paper we analyze the characteristics and requirements of a common class of scientific applications, namely Computational Simulation Models, and define a generic service-oriented framework for their execution and composition. Finally we present the work done so far towards the implementation of such framework based on the WSRF set of specifications and the Globus Toolkit.

1. Introduction

The current trend in Grid computing is to architect the basic infrastructure using a Service Oriented approach [8, 1]. The term *Service Oriented Architecture (SOA)* refers to systems structured as networks of loosely coupled services, which communicate asynchronously by exchanging messages [2]. Respectively *Service Oriented Science* is the term coined to describe scientific research enabled by distributed networks of disparate interoperating services [7]. SOAs appear to be the natural choice for developing interoperable open Grid environments. Web Services, the most popular implementation of SOA architecture, provide a wealth of standard protocols and tools for building loosely coupled, distributed, cross-domain applications.

The Grid Computing community, having early realized the potential of Web Services, shifted its focus towards the exploitation of SOA technologies. Thereafter, the *Web Services Resource Framework (WSRF)* [21] has been introduced with the intention to support common characteristics of Grid applications. Such characteristics are the explicit representation and handling of a resource's state, the explicit manipulation of resource lifecycle, the support for pull and push event notification models, and the grouping of resources. WSRF comprises five specifications which coupled with *WS-Notification* [20] and *WS-Addressing* [19] specifications render these basic required capabilities. Hence, WSRF is broadly considered an important step towards the adoption of Web Services in Grid computing.

By exploiting WSRF's fundamental capabilities we can build higher level domain specific frameworks that will satisfy relevant application requirements. In this paper we outline such framework in the domain of high-performance scientific applications, mainly simulation models, that have specific patterns of execution and interaction. Our goals are manifold:

- The exploitation of e-science (potentially parallel high-performance) applications within Service Oriented Architectures (Web Services) using the capabilities provided by the Web Services Resource Framework.
- To enable the composition of applications into data-based workflows.
- To define a domain-specific Web Services internal architecture which facilitates the execution, monitoring and life-cycle management of legacy High Performance Applications (e.g. MPI parallel applications).
- To provide support for dynamic driven applications e.g. by facilitating the dynamic evolution of application workflows based on real-time data.
- To facilitate the sharing of legacy scientific applications among different, possibly interdisciplinary,

1-4244-0054-6/06/$20.00

teams and across organizational boundaries. To facilitate the collaboration of these teams and promote the establishment of Virtual Labs.

In our previous work [6] we have focused on Web Service based solutions for the execution and composition of Simulation Models following a Provides/Uses Quantities approach. In this former work we utilized the now obsolete *Open Grid Services Infrastructure (OGSI)* [17] specification to describe open interfaces for mesh-based parallel MPI [14] applications. Grid Services wrapped around legacy MPI applications and using *Service Data Elements*, programmers declared Data Quantities that the application provided and required for execution. Meanwhile, OGSI was criticized by the Web Service purists as being too object-oriented, overloaded and incompatible with rest Web Services tooling [4]. This criticism led to the re-factoring of OGSI and the introduction of WSRF. In this paper we have accordingly redesigned our framework, to be inline with the recent developments, and we have also further abstracted the target application model in order to support a potentially broader class of applications.

The rest of the paper is organized as follows: Setting out with Section 2 we outline the class of applications that our framework aims to support. The details of the framework are presented in Section 3 and an implementation strategy is analyzed in Section 4. In Section 5 we present related work in the area of Web Service-enabled e-Science. The paper is summarized in Section 6 where we also present our plans for future work.

2. Defining the Application Domain

In this section we identify the basic characteristics of the target application domain, and derive a primary list of requirements that we want to satisfy. Based on this requirements we will move on to the architectural design of the framework.

2.1. Application Characteristics

Our primary goal is to expose *Computational Simulation Models* through Web Services. Such models (e.g Meteorological, Hydrological, Pollution, Fire Propagation) are usually developed as isolated MPI applications, typically applying the SPMD programming paradigm. These applications exhibit specific patterns of execution with the following characteristics:

1. They read a set of input data, perform some computation and produce a set of output data.
2. Data have the form of *persistent "flat" files*. By flat we don't imply lack of structure. On the contrary this files have well known standard format which can be made publicly available in the form of meta-information (e.g MPI Derived Datatypes).
3. Computation is performed in *discrete time steps*. In each timestep a new set of output files is produced.
4. Applications are developed as *autonomous*, *self-contained* programs. They comprise the legacy applications of a research team, which they use as stand-alone programs and also want to expose through open Web Service interfaces.
5. They are executed as *long lived* jobs. Their execution may take arbitrary amount time. Typically such applications may run for days or even weeks.
6. They *don't require any interactivity* with the users. In most cases users start the applications and wait for the results to be produced.

Apparently, the described application model is general enough to incorporate also other type of applications that display similar execution patterns. Typically, these applications consume data from an initial set of input files, execute and produce simulation results in the form of output flat-files.

2.2. Workflow Composition

Simulation models most of the times can benefit from their interoperation (for instance a Hydrological model can interact with a Meteorological model) to produce more accurate results. The composition of these applications is not a straightforward process since source codes are developed by separate teams which are difficult or unwilling to cooperate. In principle workflows can be constructed by connecting the output files of one application with the input files of the other application. The application composition semantics define how applications are composed and orchestrated into data workflows, and how the workflows are executed.

Our framework adopts a straightforward *Application-Data-Application* (ADA) pattern. Consider for example the simple workflow in Fig. 1. Application A executes and produces output file D. D has been declared to be input for application B, thus when it becomes available application B can start executing. As mentioned the execution of A is not one-shot but typically A produces output D in continuous *time steps*. Thus application B has to be aware of the availability of D_t where $0 \leq t \leq T$ and T

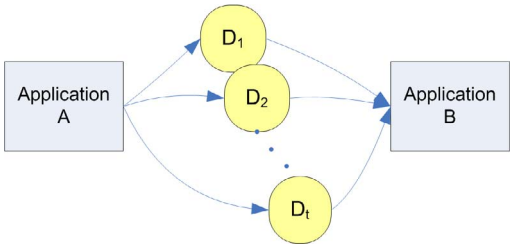

Figure 1. The ADA Composition Pattern

being the time that application A terminates. Obviously this scheme can be further extended for more than two applications and for *1-to-many* or *many-to-1* dependencies between files and applications, giving the ability to compose arbitrary graphs.

3. Framework Design

One of the fundamental ideas introduced by WSRF is the notion of the WS-Resource. A WS-Resource is the implicit coupling (usually referenced as *implied resource pattern*) of a Web Service with a Grid Resource [16]. Technically it defines the binding of a Web Service *portType* with a specific XML document (Resource Properties document), which contains state information for the resource managed by this service. Each WS-Resource exposes a set of operations (comprising the portType of WS-Resource) and state information (known as Resource Properties of the WS-Resource). In practice by utilizing the WS-Resource approach one can expose Resource state from Web Services in a standard way and clients can retrieve and optionally set state information through standard interfaces. Every WS-Resource is identified by an *End-point Reference (EPR)* defined by the WS-Addressing specification.

Our proposed framework is built around WS-Resource constructs. WS-Resources are used to wrap around the two basic components of our architecture, namely Applications and Files. Thus we define two respective classes of WS-Resources: *Application WS-Resources* and *File WS-Resources*.

3.1. Application WS-Resources

An *Application WS-Resource* abstracts a specific application, exposing its state and implementing basic functionality for the execution and handling of the application. In principle, we acknowledge that a service implements a single core operation: the execution of the application. This simplifies significantly the way a given service is perceived and handled by a client. In

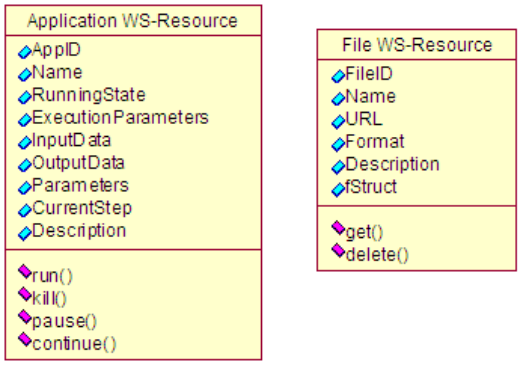

Figure 2. Logical view of Application and File WS-Resources

practice we define that the interface of the Application WS-Resource exposes the following operations:

run() Executes the application

kill() Causes the application to terminate prematurely

pause() Causes the application to pause execution. The semantics of this operation are application dependent. Usually an application that executes in a separate process cannot be suspended by another application. In this case pause() would mean that the application will complete its current step of execution and should not proceed to the next step before the service explicitly orders to do so.

continue() Resumes the execution of an application that is in the READY state (see below). Typically the operation will be invoked in order to start the next processing step after all input data have been prepared for the application.

And the following Resource Properties:

AppID A (potentially Globally Unique) Identifier of the Application.

Name Abstract name of the application.

RunningState Execution state of the application (see below).

ExecutionParameters A set of name/value pairs denoting the optional and mandatory parameters that are passed to the application upon execution.

InputData List of files (EPRs to File WS-Resources) required for input.

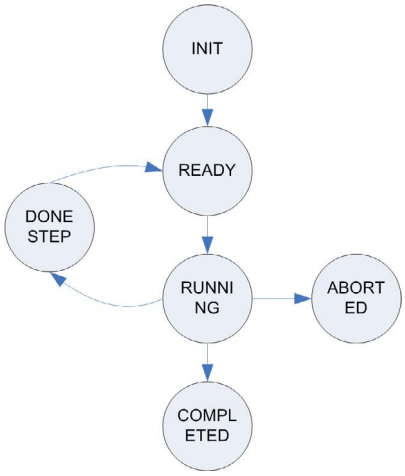

Figure 3. Running states of an Application WS-Resource

OutputData List of files (EPRs to File WS Resources) produced as output.

Description Informal/freetext description of the application

CurrentStep Current timestep of execution.

Within our framework an application may exist in one of the following states (see state transition diagram in Fig. 3):

INIT - The service is initializing (e.g. preparing the initial set of input files) and the application is not yet available to execute

READY - The service completed initialization and the applications is ready for execution or, the application has finished the execution of the current step and is ready to proceed with the next step by consuming the next set of input files.

RUNNING - The application is being executed

DONESTEP - The application has completed the current step of execution and the service is initializing the next step of processing (e.g. preparing the next set of input data)

ABORTED - The application was terminated prematurely either by the client or because of an internal runtime error

COMPLETED - The application has successfully completed all the timesteps of computation and has terminated.

As it can be seen the Application WS-Resource exposes virtually only one operation, namely *run()*. Formally this operation takes the application from the READY state, executes it (conveys it into RUNNING state) and implicitly guides it into the COMPLETED state when the computation finishes. Other operations are defined to provide elementary lifecycle management functionality (like kill() or pause() operations). Monitoring and remote administration capabilities are provided through the Resource Properties.

3.2. File WS-Resources and Data Transformation Services

A *File WS-Resource* is an abstraction of a flat file that has a well known publicly described structure, is addressed by a URL, can be accessed by a well known protocol (e.g. GridFTP) and is embedded with a set of meta-information. The interface of a File WS-Resource comprises the following operations:

get() Transfer a copy of all or part of the file, from one network location to another.

delete() Delete the file and discard the WS-Resource

We also define the following Resource Properties:

FileID A (potentially Globally Unique) Identifier of the File.

Name Abstract, human-readable, name of the file

URL Network wide pointer to the location of the file. The URL also declares the protocol that can be used to access the file. Typically for Grid environments this protocol is GridFTP (denoted as gsiftp:// at the beginning of the URL string), but other protocols can also be used (e.g. http, ftp etc)

Format Class of files that this file belongs to (e.g. HDF-EOS, Grib, DES etc)

Description Freetext description of the file contents.

fStruct Formal description of the file structure. XML Schema can be used for this reason.

File WS-Resources have been deliberately designed simple in terms of provided functionality. For advanced data handling capabilities we introduce the notion of *Data Transformation Services (DTS)*. Data Transformation Services are framework-defined and possibly user-extended portTypes, that implement enhanced file handling capabilities. Example of such capability is

the transformation of one file type to another or the remeshing of a file in order to compose two different mesh-based simulation models. Using DTS we can also implement the Quantify logic that we have presented in our previous work. GetQuantity and SetQuantity operations can be implemented in a DTS to facilitate access to derived data of a File WS-Resource.

3.3. Notifications and Workflow Composition

Notifications are an integral part of the framework. They provide the basic mechanism that enables applications to coordinate and workflows to execute. We follow the topic-based publish/subscribe pattern defined by the WS-Notification [20] family of standards, to implement the dispatching of notifications regarding the changes in the execution state of an application and the availability of input data.

Consider for example the simple workflow of Fig. 1; application B is interested to know when D_t is available, so it registers its interest for this event. Whenever D_t becomes available a notification is sent to B triggering the execution of this application with input file D_t.

We identify three application composition schemes we may follow, based on the notification and service instantiation strategies that can be applied:

- **All notifications are handled via a central coordinator** (Fig. 4a). The coordinator is responsible for instantiating all services in the workflow. Upon instantiation of WS-Resource A the coordinator registers its interest to receive notifications about the availability of D_t. Application A sends a notification to the coordinator announcing that it just finished the current step of execution and that D_t is available. The Coordinator notifies B about the event in order to retrieve the file and execute its step.

- **Application-to-application notifications** (Fig. 4b). WS-Resource B registers to WS-Resource A its interest to receive notifications about the availability of D_t in every time step t. When A finishes the current step of execution it checks who is interested about the results and sends notifications to service B.

- **File-to-application notifications** (Fig. 4c). WS-Resource B contacts File WS-Resource D requesting to be notified whenever a new snapshot of D_t is available. Whenever a new file in the series of D_t is available, D sends a notification to B.

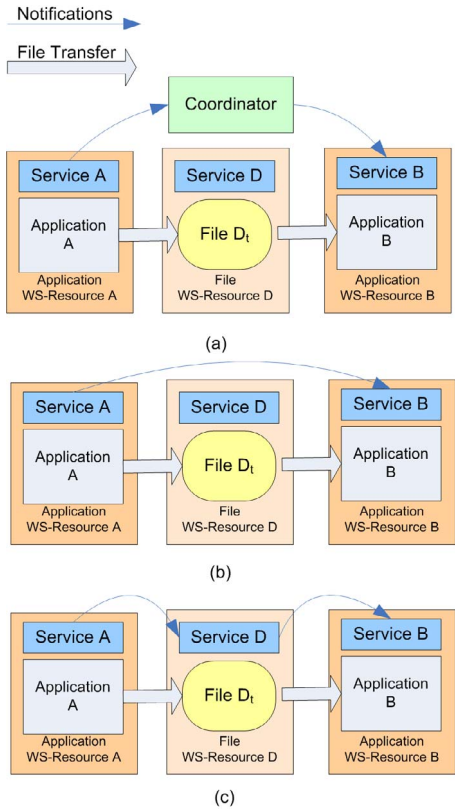

Figure 4. The three application composition schemes

We believe that all of the above approaches are of value and should be supported in an integrated framework, providing the required versatility and flexibility to the programmer.

3.4. Other Services

As part of our architecture, albeit not in the form of WS-Resources but rather as regular Web Services, we identify the following base services, which are essential for the development of a sound Grid infrastructure:

Information Services. Programmers can utilize UDDI [15] compliant registries, to store and discover information about WS-Resources, either Applications or Files, as they would do for any regular Web Service. Moreover, being in the WSRF context, we have the ability to index services according to their Resource Properties. Within our framework, clients can search for services that expose particular RPs or for RPs with

specific value, obtain the EPRs to the relevant WS-Resources and use their functionality.

Dynamic binding can also be supported. Clients can search for models that are already executing, bind and retrieve current or past results. Files can also be queried and retrieved to be used either for initial input data or in order to reconstruct the time line of the simulation execution. File WS-Resources give also the ability to build libraries of simulation results. In many situations a user may not need to execute a new simulation rather search the registry for the desired data and retrieve them.

Security Services. As with any other Grid framework, security is essential for the establishment of a trusted execution environment. Standard Grid security mechanisms (like X.509v3 digital certificates) are needed to protect the integrity of communication and data. Access control has also to be enforced in terms of who owns a WS-Resource and who can exploit it.

4. Implementation Details

In this section we provide a brief insight to the implementation details of the framework starting with the target Web Services platform.

4.1. Web Services Platform

The dilemma for choosing a target platform for developing the framework is actually non-existent, since currently there are only a few implementations of the WSRF specifications. Thus the *Globus Toolkit 4* [12] is the natural choice, since not only it is the de-facto platform for developing Grid infrastructures, but also is one of the few software stacks that provide a Web Services container and APIs (comprising the *WS-Core* component of GT4) that support WSRF, WS-Notification and WS-Addressing. Moreover GT4 renders the required infrastructure services like *Meta Directory Services (MDS4)* for indexing WS-Resources, *Security* and *Data Management Services* (GridFTP, Data Replication etc).

4.2. Internal Architecture

Although much time and effort has been invested in defining standards for the External Architecture of Web Services, namely the protocols and vocabulary for service-to-service interactions, little has been done regarding the important issue of the Internal Architecture, that is, how the service interacts with the wrapped application. One reason is that such internal architectures are application specific and no universal solution can be applied. Global Grid Forum (GGF), the leading organization for Grid standardization, does not currently have any initiative working towards this direction [13]. Since the process of defining the *Open Grid Services Architecture (OGSA)* [9] is progressing, we believe that among the future goals of the standardization process should be the study of such domain-specific internal architectures in Grid SOA-based environments.

In the context of our framework we need a standard way to facilitate the interaction between the Web Service and the simulation model that it executes. We have decided to follow the paradigm of *Common Gateway Interface (CGI)* [3] standard since it provides a solution to a similar problem. CGI defines how external gateway programs can interface with information servers such as HTTP servers. Similarly, we execute standard pre-compiled, binary applications, that interact with their environment through the three standard streams (*stdin*, *stdout* and *stderr*), and also have access to the operating system environment variables. Applications are deployed and executed from a specific directory known to the respective Application WS-Resource, and run as regular OS processes printing their output to the *stdout* stream.

The *application-to-service* interaction is achieved using a predefined set of short XML formatted messages. The service constantly scans the messages printed by the application to the *stdout* and *stderr* stream, looking for encoded XML output. When it finds such output it parses the contents to extract the relevant information sent by the application. Reversely, the *service-to-application* interaction is accomplished through the *stdin* stream of the application. The service instructs the application to terminate, pause or continue execution, using a simple vocabulary.

We are implementing such an internal architecture for MPI Simulation Models using the MPI *Profiling Interface* and *Error Handlers* [14]. More specifically we are developing an MPI library which intercepts significant function calls in order to convey important events to the service (e.g. errors, barriers that denote checkpoints etc).

4.3. Sample Workflow Execution

Below we describe a typical scenario for executing a workflow of two simulation models, following the ADA composition pattern and the central coordinator scheme. According to the example, there are two Application WS-Resources, namely ModA and ModB,

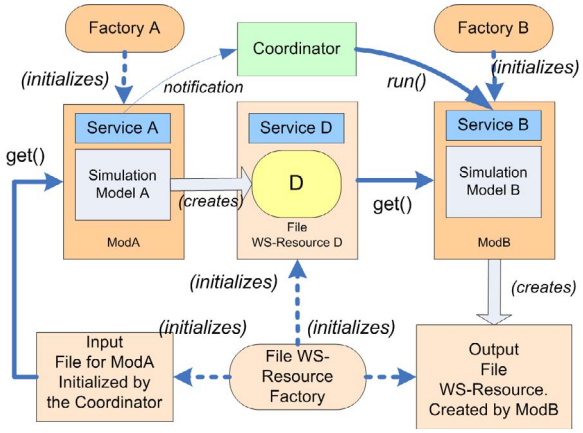

Figure 5. Workflow Execution Example

which expose two respective simulation models. ModB requires as input for execution the results of ModA. A coordinator application is responsible for the execution of the two models and to handle the details of their interaction. We assume that the EPRs for the input File WS-Resources of ModA are prior known to the coordinator and that we apply a *Factory pattern* for the instantiation of all new WS-Resources. The whole process comprises roughly the following steps:

1. The coordinator contacts the Factory services for Models A and B to create an instance of the respective WS-Resources. The coordinator registers its interest to be notified whenever there is a change in the execution state of ModA or ModB.

2. The coordinator instantiates the InputFiles RP of Application A. ModA moves to READY state. ModB remains in INIT state since it has to wait for the results from AppA.

3. ModA is executed (the *run()* operation is invoked by the coordinator) and the value of the RunningState RP is set to RUNNING.

4. ModA completes a step and contacts the File Factory to create the File WS-Resources that correspond to the output files of the simulation. The EPRs of the File WS-Resources are stored in ModA's OutputFile list. Then ModA moves to DONESTEP state which triggers the sending of a notification to the coordinator.

5. The coordinator retrieves the EPR of the output files, by accessing the OutputFiles RP of ModA, and uses them to populate the InputFiles RP of ModB. Then it invokes the run() operation since ModB has now moved into READY state.

6. ModB resolves the WS-Resource listed in the InputFiles RP and calls the *get()* operation of the File WS-Resources. The operation transfers the files in the execution context of ModB.

7. ModB runs and consumes the latest results of ModA. When it finishes, ModB moves to DONESTEP state.

The execution circle can be repeated arbitrarily with the coordinator refreshing the input files of ModA. In every step of the execution a client may invoke the operations of the services, or investigate the ResourceProperties of the resources to acquire insight about the state of the workflow execution or the execution of a specific application.

5. Related Work

The field of Service-Oriented Grids in general, and the exploitation of Web Services in e-Science applications in particular, are attracting enormous interest [10]. GGF has various Working Groups working on the areas of architectures and standards. Among them, work done by the *Basic Execution Services* (OGSA-BES-WG) and *Application Content Services* (ACS-WG) focus on the packaging and execution of arbitrary applications. The Globus Toolkit and the OMII Software distributions [18] provide general purpose, infrastructure web services, which can be used to build SOA Grids.

Numerous R&D projects are building SOA Grid environments and tools. Among the most influential is gLite [11] which is developed in the context of EGEE (Enabling Grids for E-sciencE) and LCG (LHC Computing Grid). gLite is a rich-featured software bundle, that exploits Web Services to provide Grid Services for Job Execution, Data management etc. Notable also is the NSF funded LEAD project [5], which focuses on tools for execution of scientific workflows, the development of scientific portals and web service-based e-science environments in general.

The above efforts are focused on the development of general-purpose Service-Oriented standards, infrastructure and platforms that can handle the execution and monitoring of arbitrary types of applications. Our work builds upon these general-purpose infrastructure tools and proposes a 1-1 mapping of customized WSRF Web Services that explicitly manage specific applications. This way we believe that we can better expose the particular semantics of an application and thus facilitate its exploitation by third users as self-contained Grid resources. Inline with this we advocate the development of relevant domain-specific standards that will

standardize the interfaces and the expected semantics of such Web Services.

6. Conclusions and Future Work

In this paper, we have used the WS-Resource notion of WSRF to outline an e-Science Grid framework, that facilitates the execution and composition of Computational Simulation Models. We identified two basic WS-Resources, namely Applications and Files, which are used respectively to abstract legacy simulation models and flat files with simulation results. Finally, we have demonstrated how WS-Resources can be composed in arbitrary scientific workflows.

We have followed a different approach than that for instance of the Job Submission services of gLite or OMII. In our framework the owner of the application is responsible for the details of the execution environment and the computing resources that will be dedicated to run the application. The service is provided as an access point to the application itself and not to the application execution environment (e.g. Condor, Globus, MPI etc), although we can apply such extensions to the architecture.

Our basic philosophy is that Web Services provide only one fundamental operation, namely run(), which initiates the execution of the model. We also advocate the significance of data, promoting them as first order Services.

We are currently progressing with the implementation of the framework. We have worked on a proof of concept that has helped us in the fine tuning of the theoretical base of the framework. In the imminent future we will continue with the development, focusing on the domain specific Data Transformation Services, the details of security and information services, as well as on issues like WS-Resource persistency and framework administration. One of our first priorities also is to finalize the Internal Web Services Architecture in the domain of MPI Simulation Models, and the formalization of an XML syntax for service-to-application and application-to-service interactions. Finally, since we focus on Simulation Model composition, we are currently extending the Data Services in order to support mesh quantities, mesh data conversions etc, aiming to provide a comprehensive Service-Oriented programming framework for simulation models coupling.

References

[1] M. Atkinson,et al. Web service grids: An evolutionary approach. *Concurrency and Computation: Practice and Experience, Wiley*, Volume 17, Issue 2-4:377 – 389, Feb 2005.

[2] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard. Web Services Architecture. Working draft, W3C, http://www.w3.org/TR/ws-arch, 2004.

[3] The Common Gateway Interface (CGI). [Online]. http://hoohoo.ncsa.uiuc.edu/cgi/.

[4] K. Czajkowski, D. Ferguson, I. Foster, J. Frey, S. Graham, T. Maguire, D. Snelling, and S. Tuecke. From open grid services infrastructure to WSResource framework: Refactoring & evolution. Technical report, Global Grid Forum, February 2004.

[5] K. Droegemeier,et al. Linked environments for atmospheric discovery (LEAD): A cyberinfrastructure for mesoscale meteorology research and education. In *Info. Processing Systems for Meteorology, Oceanography, and Hydrology*. Amer. Meteorol. Soc., 2004.

[6] E. Floros and Y. Cotronis. Towards a grid services based framework for the virtualization, execution and composition of MPI applications. *Parallel Processing Letters, World Scientific*, Vol. 15(1&2), March/June 2005.

[7] I. Foster. Service-oriented science. *Science*, vol 308, May 2005.

[8] I. Foster, C. Kesselman, M. Nick, and S. Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration. Technical report, Open Grid Service Infrastructure WG, Global Grid Forum, June 2002.

[9] I. Foster, et al. The Open Grid Service Architecture v1.0. GGF Document Series, GGF, July 2004. GFD.30.

[10] D. Gannon, et al. Building grid portal applications from a web service component architecture. In *Proc. of the IEEE*, volume Vol. 93, No. 3, March 2005.

[11] gLite: Lightweight Middleware for Grid Computing. http://glite.web.cern.ch/glite/.

[12] Globus Toolkit 4. http://www.globus.org/toolkit.

[13] H. Kishimoto and J. Treadwell. Defining the grid: A roadmap for ogsa standards. GGF Document Series GFD-I.053, GGF, September 2005.

[14] MPI: A Message Passing Interface Standard. Technical report, Message Passing Interface Forum, June 1995.

[15] OASIS. Universal Description Discovery and Integration (UDDI). [Online]. http://www.uddi.org/.

[16] OASIS. Web services resource specification (ws-resource), October 2005.

[17] Open Grid Service Infrastructure Primer. GGF Document Series GFD-I.031, GGF, August 2004.

[18] Open Middleware Infrastructure Institute (OMII). [Online]. http://www.omii.ac.uk.

[19] WS-Addressing. [Online]. http://www-106.ibm.com/developerworks/library/specification/ws-add.

[20] WS-Notification. [Online]. http://www-106.ibm.com/developerworks/library/specification/ws-notification.

[21] Web Services Resource Framework (WSRF). [Online]. http://www.oasis-open.org/committees/tc_home.php?wgabbrev=wsrf.