Computing and Informatics, Vol. 21, 2002, 333–345

THE DESKTOP GRID ENVIRONMENT ENABLER

Giovanni Aloisio, Massimo Cafaro, Daniele Lezzi

High Performance Computing Center
University of Lecce, Italy
e-mail: {`giovanni.aloisio, massimo.cafaro, daniele.lezzi`}`@unile.it`

Revised manuscript received 8 November 2002

Abstract. This paper describes our ***De***sktop ***Gr***id ***E***nvironment ***E***nabler (DEGREE), a set of Web Services that provides advanced capabilities for grid computing. DEGREE services are based both on the Globus Toolkit and the Grid Resource Broker, a grid portal developed at the University of Lecce. Trusted users can develop innovative, grid-aware applications that seamlessly access computational resources and services exploiting our Web Services independently of platform and programming language.

Keywords: Grid Computing, Web Services, Open Grid Services Architecture, Globus Toolkit

1 INTRODUCTION

The grid computing paradigm [1] emerged recently as a new field distinguished from traditional distributed computing because of its focus on large-scale resource sharing and innovative high-performance applications. The grid infrastructure ties together a number of Virtual Organizations (VO) [2], that reflect dynamic collections of individuals, institutions and computational resources. Flexible, secure and coordinated resource sharing between VOs requires solving a number of challenging issues like authentication/authorization, resource discovery/management, access to remote data etc.

The Globus Toolkit [3] addresses the need for core grid services, and has gained wide acceptance woldwide, so that it is deployed at multiple organizations/institutions as the middleware of choice for grid computing and many scientific endevours rely on it. Leveraging the Globus Toolkit we built the Grid Resource Broker

(GRB) [4, 5], a grid portal providing advanced Globus services that trusted users can access using a web browser.

In this paper we describe how GRB functionalities will be made available to our users first in the context of the Web Services framework, and subsequently in the context of the recently proposed Open Grid Services Architecture (OGSA) [6, 7]. The paper is organized as follows. Section 2 describes the Web Services framework [8] on which OGSA is based, and Section 3 presents OGSA. A description of the software infrastructure developed for our DEGREE Services is given is Section 4. Finally, Section 5 briefly describes the DEGREE services we are going to provide. We conclude the paper in Section 6.

2 WEB SERVICES

The Internet and the Web allow people to publish documents easily; such documents can be accessed regardless of geographic location and require just a browser to be visualized. Recently, the number of available services on the Web has grown considerably, and the users can access commercial, public and e-government services.

Currently, the focus on the usage of such services has been shifted from people to software applications. The business world is increasingly demanding automation of procedures, flexibility and agility to deploy and provide on the web solutions to customers, partners, suppliers and distributors. Often, the driving force is the time-to-market pressure that need to be faced. Moreover, integration of services is required both for B2B (Business to Business) and EAI (Enterprise Application Integration).

Switching from people to software applications in the Web Services framework is made possible by the convergence of two key technologies: the Web, with its well known and accepted standard protocols for universal communication, and Service-Oriented computing where data and business logic is exposed through a programmable interface (e.g. CORBA, RMI, RPC). Thus, Web Services are accessed through the HTTP/HTTPS protocols and utilize XML (Extensible Markup Language) for data exchange. This in turn implies that Web Services are independent of platform, programming language, tool and network infrastructure.

Cross-platform integration becomes easy, because applications are converted to loosely coupled services, accessible using both synchronous and asynchronous communication. These services can be assembled and composed in such a way to foster the reuse of existing back-end infrastructures. The Web Services architecture, as shown in Figure 1, is based on the following core technologies:

- XML
- SOAP (Simple Object Access Protocol)
- WSDL (Web Services Description Language)
- UDDI (Universal Description, Discovery and Integration)

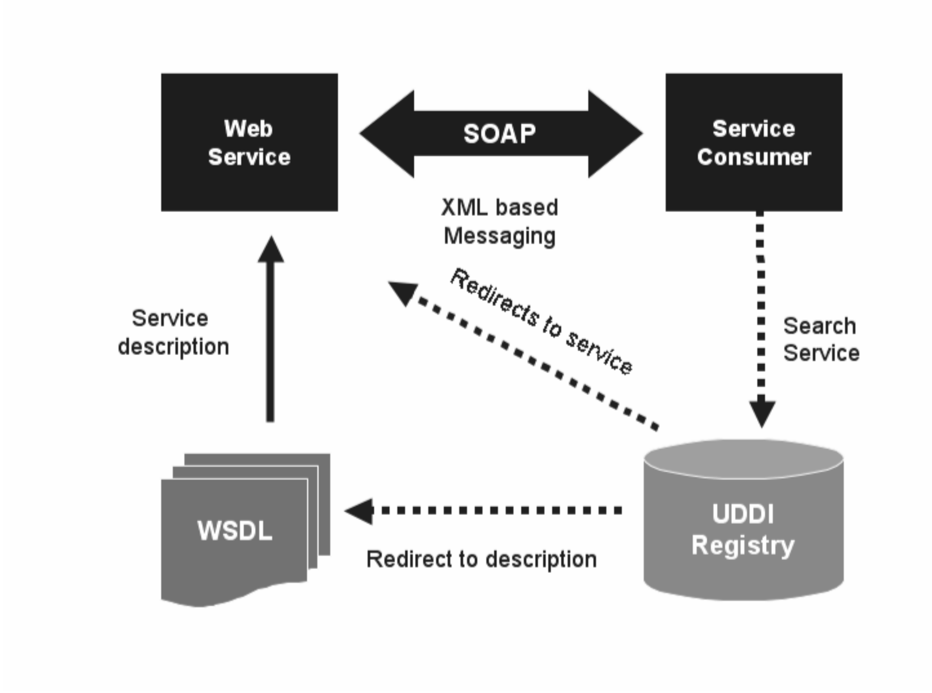

Fig. 1. Web Services Architecture

A Web Service itself that can be accessed by a *service consumer* can be though as a *service provider*; it is worth noting here that the same service can be simultaneously both a provider and a consumer, if it needs to access another service to serve its request. The interaction among the Web Services components exploit the UDDI *registry* as follows.

The service provider defines a reference to its Web Service using WSDL. The WSDL document is published into the UDDI registry, so that a service consumer can search the registry and retrieve the WSDL reference to the Web Service. The service consumer, using the information stored inside the WSDL document, contacts the Web Service and issues a service request.

The XML messaging between service consumer and provider exploits the SOAP protocol that defines two different kind of interactions: RPC and Document-Oriented. The former works exactly as the standard CORBA and EJB middleware: the client requests a service and synchronously waits for service completion. The latter asynchronously starts a service and returns immediately without waiting for service completion.

The typical scenario depicted here is based on *document publishing* (WSDL reference), *searching* for a service and *binding* to the service provider. Publishing, searching and binding in the Web Services Framework harness a set of standard, layered protocols as shown in Figure 2. The first three levels of the stack going bottom-up provide the basis to implement a web service, i.e., the network, an XML based protocol for message exchange and a description of the service. The next levels in the hierarchy provide mechanisms for document publication, searching for a service and for handling a flow of interactions among web services.

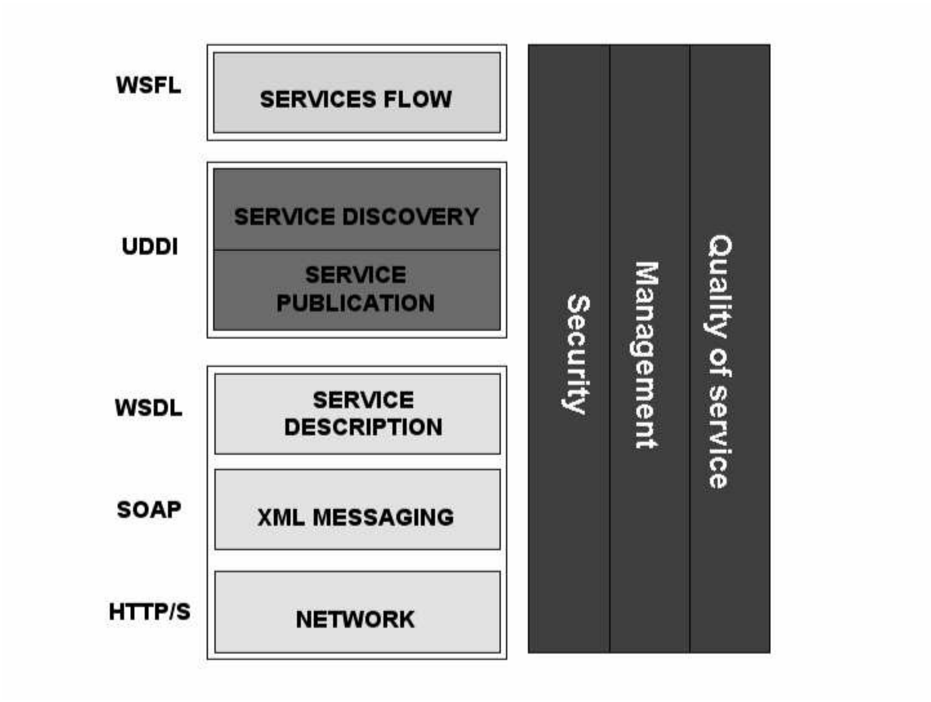

Fig. 2. Web Services Framework layers

The network layer provides core protocols that can be used to access a service: HTTP, HTTPS and others, including FTP and IIOP. The XML messaging layer is based on the *de facto* standard SOAP. Promoted jointly by IBM and Microsoft, SOAP defines messages as composed of three parts: *envelope*, *header* and *body*. The envelope defines message contents, how to process the message and optionals encoding rules needed to serialize the contents. The message header can include information about the kind of interaction requested (RPC or Document-Oriented), and the body contains the actual data or the document to be sent and optional rules about the data format.

The SOAP protocol is used as follows:

1. The service consumer assembles a SOAP message that invokes the service provider and passes it to the underlying SOAP infrastructure which, using the network infrastructure, actually sends the message;
2. The network delivers the message to the service provider; the message is converted automatically from XML to a valid implementation that invokes the service using the encoding rules specified in the message itself;
3. The service provider processes the incoming request and creates a SOAP reply message containing the results; as before, the message is handed to the SOAP infrastructure that using the network sends the message back to the service consumer;
4. Finally, the service consumer receives the SOAP reply message, which is automatically converted from XML to a valid implementation, so that results can be immediately used by the application.

SOAP provides automatic marshalling/unmarshalling of the arguments, like RPC. The interaction described here is the classic *request/response*; however, other mechanisms are available, including *publish/subscribe*, *push* and *one-way messaging.* The service description layer provides the WSDL language used to describe how to invoke a service; the WSDL document actually decouples the service consumer from the provider, being the only shared knowledge needed to serve a request.

WSDL is an XML grammar that describes how to contact a Web Service; it defines the messages to/from the Web Service, logical collection of messages known as *port type*, and how a given port type is bound to a particular wire protocol. The port type represents the actual interface for accessing the web service: it defines a set of permissible operations, related parameters and the input/output messages. The *binding* maps a port type to a specific protocol using a specific set of encoding rules. Finally a service is a set of *ports* that implement port types.

The service publication and discovery layers allow service providers to register a reference and service consumer to search for such a reference. We will see in Section 3 that UDDI in OGSA is replaced by WSIL (Web Services Inspection Language). In the Web Services framework, the UDDI registry gets populated with service descriptions (called *businessServices*), grouped by business categories. The business societies (called *businessEntities*) sign up as providers of the services registered during the population phase and the registry assigns an identifier for each service and society. The societies can now query the registry to retrieve the information as needed. The next section introduces OGSA and discusses how it differs from the Web Services framework.

3 THE OPEN GRID SERVICES ARCHITECTURE

In the previous vision of the grid the attention was focused on the *protocols* needed to provide interoperability among VO components. OGSA now shifts the attention to services as follows: the grid becomes an extensible set of *Grid Services* that may be aggregated to provide new capabilities. Grid services, as envisioned, retain several features of the Web Services framework; for instance it is highly desirable to retain service description and discovery, and binding of service descriptions to wire protocols.

OGSA thus leverages WSDL and SOAP, but gives its own definition of a Grid service, which is "a (potentially transient) stateful service instance supporting reliable and secure invocation (when required), lifetime management, notification, policy management, credential management, and virtualization" [7]. The definition clearly states the need for *transient* services in the grid environment besides *persistent* services as provided by the Web Services framework; this entails the need for interfaces able to manage service lifetime, policies and credentials, and to provide support for notification. Virtualization of resources is a natural consequence of the adoption of Service-Oriented computing: computational resources, storage, networks, applications, archives etc. are all presented as Grid Services.

OGSA ties together Web Services and the Globus Toolkit; in particular the following Globus components are part of OGSA:

- GRAM (Globus Resource Allocation and Management);
- MDS (Metadata Directory Service);
- GSI (Globus Security Infrastructure).

We have seen that in the Web Services framework UDDI is the standard adopted for service discovery and publication; in OGSA these functionalities are provided by WSIL. Service consumers locate service descriptions published by service providers through WSIL: service descriptions are usually URLs pointing to WSDL documents, however a service description can also point to an entry stored inside an UDDI registry. Links to other WSIL documents are common; sometimes we will find links pointing to an UDDI entry. A service provider using WSIL simply creates a WSIL document and makes it accessible over the network. OGSA specifies a number of interfaces, but here we recall the following:

- factory;
- mapper;
- registry.

The factory interface allows for creation of new grid service instances, in particular reliable creation with once and only once semantic. A *Grid Service Handle* (GSH) is returned: this is a globally unique identifier (a URL) based on the name of a home mapper service. The mapper interface is in charge of converting the GSH to a *Grid Service Reference* (GSR), i.e., the WSDL document actually needed to invoke the service. The registry interface returns a WSIL document containing the GSHs of a set of Grid services.

Other interfaces provide support for lifetime management, discovery, notification and authorization. It is worth noting here that the discovery interface differs from the registry interface as follows: it provides support for querying a Grid Service instance to retrieve specific service information.

4 DEGREE INFRASTRUCTURE

In this section we briefly review the software components that belong to our DEGREE architecture. As shown in Figure 3, we have a layered architecture. The highest level represents a user's application. This software must be written using the WSDL references to our services; these references can be found either by searching our UDDI registry or using WSIL, as specified in OGSA.

In order to simplify SOAP based software development, we exploit the gSOAP toolkit [9], a flexible set of compiler tools that provide C/C++ language binding for the development of SOAP Web Services and clients. The gSOAP toolkit was chosen because it is freely available, and in our opinion, well suited for the conversion of legacy application using SOAP because its main feature is a transparent SOAP API. As a matter of fact, gSOAP hides irrelevant SOAP-specific details from the user through the use of compiler technology. The gSOAP stub and skeleton compiler can be used to automatically map native and user-defined C and C++ data types to semantically equivalent SOAP data types and vice-versa. SOAP interoperability is thus achieved without explicit knowledge of SOAP details.

We have collaborated with Robert Van Engelen, which is the principal investigator and developer of the gSOAP project, to add HTTPS and cookie support to gSOAP; moreover we have also implemented Globus GSI support (available as a gSOAP plugin [15]). This is a first step toward an OGSA compliant gSOAP toolkit, since OGSA security is based on GSI. Currently, the OGSA specification is not yet finalized and it is undergoing a revision process in the context of the Global Grid Forum [10]. As a result, we have decided to rewrite GRB as a set of Web Services for the time being, and we plan to convert GRB services in Grid Services as envisioned by OGSA when the specification will be finalized.

The next layer in the architecture is our set of DEGREE services. These are written using two libraries we developed for GRB [11, 12], to provide high level Globus services, and the gSOAP toolkit. Security is addressed using HTTPS and cookies to authenticate trusted users; the GSI enabled plugin for gSOAP will be used to provide authentication using X.509v3 certificates and delegation of credentials. The next layer in our architecture includes Globus and third-party services (e.g. a MyProxy [13] server for user's authentication/authorization that will not be needed once GSI support will be added) exploited as capability providers. Finally, the last layer includes the computational resources available to a user on the grid.

5 DEGREE SERVICES

Currently, all of the services are implemented as stand-alone, multi-threaded servers with support for SSL. DEGREE provides the following services:

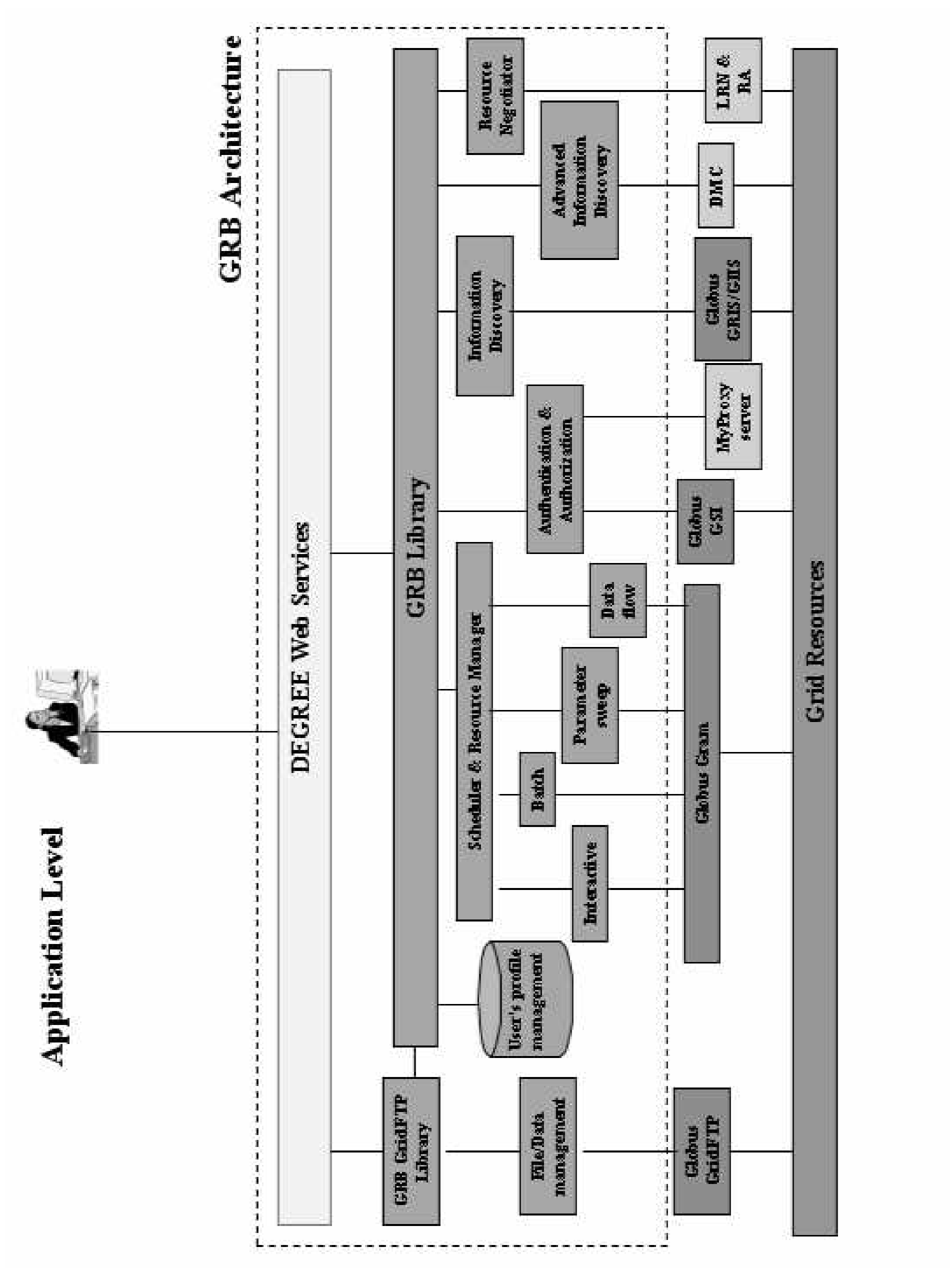

Fig. 3. GRB Architecture

- Authentication/Authorization,
- User's profile management,
- Job submission,
- Job monitoring,
- High performance file transfer,
- Access to Globus information services.

Authentication and authorization is based on X509v3 certificates and the MyProxy package. Before using DEGREE services, users are required to create restricted credentials and store them as a proxy on our MyProxy server. The DEGREE Authentication/Authorization service then retrieves the proxy using the user's supplied pass phrase and grants or denies the access to other services checking the distinguished name of the proxy issuer.

The users pass phrase for the credentials stored inside the MyProxy server is sent from the client application using HTTPS to prevent network sniffing, and cookies are used to establish and maintain session information. This avoids the need to authenticate a user each time she accesses a service. The users proxy is then used for single sign-on to the computational grid. The GRB cookie stores the following information [14]:

- Users login name;
- Timestamp;
- Expiration date;
- Message Authentication Code (MAC) for the previous plaintext data.

We encrypt the cookie using SSL for additional security; moreover we utilize ephemeral cookies, i.e., our cookie is not stored on the users file system (persistent cookies) but in the client application memory. When the user exits the application, the ephemeral cookie automatically disappears. This prevents the possibility that an error in the way the client application or the user handles the cookie file makes it accessible over the Internet.

DEGREE services related to the user's profile management allow the user to modify her user's profile to insert, edit or delete computational resources that the user can access on the grid. Globus version 2.0 must be installed on each machine that the user wants to use. For each machine the user also enters the path to her shell and the node/hour cost. The former is used to take advantage of the user's environment on the remote machine, and the latter is used by GRB to schedule the user's submitted jobs.

Job submission services provide the user with the capability of submitting:

- Interactive jobs;
- Batch jobs;
- Parameter sweep jobs

- Data Flow jobs.

Interactive jobs run for a small amount of time and return their output to the user upon completion. This feature allows to execute simple commands, e.g. to list the files in the user's home directory, to remove a file, etc. The client needs to provide as input the machine where the job will be executed, the executable pathname, command-line arguments (if any), the machine and the pathname that refers to the user's input for the job. Only the first two parameters are required to execute the job, the others are optional. The executable pathname needs to be an absolute one only when the executable must be transferred from a machine that differs from the one chosen for execution. If the executable is available on the same machine where it will be runned, then GRB takes automatically advantage of the information stored in the user's profile to access the user's environment on the remote machine, so that if the executable can be located using the PATH variable, the user can simply refer to it by its name. If the executable pathname starts with a tilde, GRB tries to locate the executable starting from the user's home directory. If the executable and/or the input (can be a single file or a directory) must be transferred, GRB starts automatically a third-party file transfer, before running the job.

Batch jobs are submitted for execution, and GRB returns a job identifier that can be used later to infer the job status, exploiting the GRB job monitoring service. In addition to the parameters needed to execute an interactive job, the user can also specify a computational resource and a pathname that refers to the job output. GRB automatically stages the executable and/or the input, submits the job for execution and returns the job identifier, and finally stages the output upon job completion.

Parameter sweep jobs are also commonly referred to as High Throughput jobs. Here, the user needs to submit the same executable on different machines, each time running with a different input. This is required for parameter studies. The parameters needed for a parameter sweep job include the ones for a batch job submission, the number of machines to be used for execution and their hostnames, and the number of input files. GRB submits the job, staging executable, input and output if needed, and returns the job identifier.

Data Flow jobs are described by a Directed Acyclic Graph (DAG). The vertices represent the jobs to be executed (each one can be though of as a batch job), and the edges model precedence constraints. GRB schedules all of the jobs for execution computing a topological sort of the input graph.

The high performance file transfer service provides the user with the capability of starting a parallel third-party file transfer. The service can transfer single files or directories and requires as input only the source and destination machines and the pathnames where to find/store the files. Since the transfer time can be high, the service returns to the user a file transfer job identifier. This can be used to verify later the file transfer status.

The job monitoring service allows the user to check the status of a batch, parameter sweep or data flow job. The input for the job tracking service is a job identifier, and the output includes all of the information related to the job submitted. Currently the service returns date and time of job submission, the executable pathname, command-line arguments, information about executable, input and output staging, the Globus job identifier (that differs from GRB job identifier) and the job status (running, pending, suspended, failed or done). This service can also be used to check the status of a scheduled high performance file transfer.

Finally, DEGREE also allows to query a Globus GRIS server to get the feature of a specific machine, or a Globus GIIS to find a set of machines that match user's criteria. Information services are a key component in the highly dynamic grid environment, for advanced information discovery and monitoring. The GRIS service needs as input parameters only a machine hostname, while the GIIS service also needs a set of criteria (CPU speed, model, amount of memory, etc.). Queries to the Globus information services exploit the LDAP protocol and, if needed, SASL for security reasons.

6 CONCLUSIONS

In this paper we have described DEGREE, a set of Web Services for advanced grid computing, based on the Grid Resource Broker. DEGREE services allow developers of grid-aware applications to take advantage of the underlying Globus grid infrastructure transparently. No explicit knowledge of the Globus Toolkit is required, services can be composed as needed and provide a friendly interface to high-level Globus services. We plan to extend DEGREE capabilities to support advanced scheduling algorithms. Moreover, we have started work on the gSOAP toolkit to make it OGSA compliant, by adding Globus GSI support. This is a first step toward our final goal, the evolution of our Grid Resource Broker as a set of Grid Services.

REFERENCES

[1] Foster, I.—Kesselman, C.: The Grid. Blueprint for a new computing infrastructure. Morgan Kaufmann, San Francisco, 1999.

[2] Foster, I.—Kesselman, C.—Tuecke, S.: The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International Journal Supercomputer Applications, Vol. 15, 2001, No. 3, pp. 200–222.

[3] Foster, I.—Kesselman, C.: Globus: A Metacomputing Infrastructure Toolkit. Intl J. Supercomputer Applications, Vol. 11, 1997, No. 2, pp. 115–128.

[4] Aloisio G.—Cafaro, M.—Blasi, E.—Epicoco, I.: The Grid Resource Broker, a Ubiquitous Grid Computing Framework. To appear in Journal of Scientific Programming.

[5] Aloisio G.—Cafaro, M.: Web-Based Access to the Grid Using the Grid Resource Broker Portal. To appear in Concurrency, Practice and Experience.

[6] Foster, I.—Kesselman, C.—Nick, J.-Tuecke, S.: Grid Services for Distributed System Integration. Computer, Vol. 35, 2002, No. 6, pp. 37–46.

[7] Foster, I.—Kesselman, C.—Nick, J.—Tuecke, S.: The Physiology of the Grid: An Open Grid Services Architecture for Distributed System Integration. Technical Report for the Globus project. `http://www.globus.org/research/papers/ogsa.pdf`.

[8] Kreger, H.: Web Services Conceptual Architecture WSCA 1.0. IBM, 2001.

[9] van Engelen, R. A.—Gallivan, K. A.: The gSOAP Toolkit for Web Services and Peer-To-Peer Computing Networks. Proceedings of IEEE CCGrid Conference, Berlin, pp. 128–135, May 2002.

[10] Global Grid Forum: `http://www.gridforum.org`.

[11] Aloisio G.—Cafaro, M.—Epicoco, I.: Early Experiences with the GridFTP Protocol Using the GRB-GSIFTP Library. To appear in Future Generation Computing Systems, special issue on Grid Computing, Elsevier.

[12] Aloisio G.—Cafaro, M.—Blasi, E.—Depaolis, L.—Epicoco, I.: The GRB Library: Grid Programming with Globus in C. Proceedings of HPCN Europe 2001, Amsterdam, Netherlands, Lecture Notes in Computer Science, Springer-Verlag, 2110, pp. 133–140.

[13] Novotny, J.—Tuecke, S.—Welch, V.: An Online Credential Repository for the Grid: MyProxy. Proceedings of the Tenth International Symposium on High Performance Distributed Computing (HPDC-10), 2001, IEEE Press.

[14] Fu, K.—Sit, E.—Smith, K.—Feamster, N.: Dos and Don'ts of Client Authentication on the Web. Proceedings of the 10th USENIX Security Symposium, Washington, D. C., 2001.

[15] Cafaro, M.—Lezzi, D.—van Engelen, R. A.: The GSI plugin for gSOAP. `http://sara.unile.it/~cafaro/reg.html`.

Giovanni Aloisio is Full Professor in Computer Engineering at the Engineering Faculty of the University of Lecce and Director of the High Performance Computing Center (HPCC) of the Department of Innovation Engineering & ISUFI/University of Lecce. His research covers High Performance, Distributed and Grid Computing. He was the co-founder of the European Grid Forum (Egrid) now merged into the Global Grid Forum (GGF). He is a member of IEEE Computer Society and holds a permanent visitor position at CACR Caltech. Contact him at `giovanni.aloisio@unile.it`.

Massimo Cafaro is an Assistant Professor at the Department of Innovation Engineering of the University of Lecce and a member of the ISUFI/High Performance Computing Center (ISUFI/HPCC). His research covers High Performance, Distributed and Grid Computing. He is also interested in computer security. Since the beginning of 1998 he is involved in grid projects. He is currently working in the NPACI metasystem trust area to build grid-enabled applications and the Grid Resource Broker, a web portal to computational grids. He received a degree in computer science from the University of Salerno and a Ph.D. in Computer Science from the University of Bari. He is a member of IEEE and of IEEE Computer Society, and holds a visitor position at CACR Caltech. Contact him at `massimo.cafaro@unile.it`.

Daniele Lezzi is currently a member of the ISUFI/High Performance Computing Center (ISUFI/HPCC). He received a degree in computer engineering from the University of Lecce, Italy. His research interests include high performance computing, distributed and grid computing. He's developing a Web Services based grid portal called GRB exploiting the Globus Toolkit package. He's involved in the European Gridlab project, a Grid Application Toolkit and Testbed. Contact him at `daniele.lezzi@unile.it`.