Grid Architectures and the European DataGrid

Andrea Domenici

Abstract— **The term "Grid" is used in reference with wide area (possibly planetary) computer networks that enable very large communities of users to access massive computational facilities, ideally with the same transparency and ease that users of electrical devices, whether industrial machines or home appliances, expect from the electrical power grid. This paper introduces the basic concepts of Grid architectures, drawing upon the experience of the European DataGrid project, one of the most important enterprises in this field, aimed at providing support to scientific communities involved in High Energy Physics, Earth Observation, and Biology.**

Keywords— **Computational Grids, European DataGrid, distributed systems, Internet.**

I. Introduction

COMPUTING POWER should be like electrical power: cheap, available, reliable. Recalling the development of electrical power, Ian Foster and Carl Kesselman, the authors that made the term *computational grid* popular, observe [1] that

"the truly revolutionary development was not, in fact, electricity, but the electric power grid".

The above sentence may be surprising at first, but it is quite obvious if we think how difficult it would be to use electric power without an infrastructure that delivers power from various sources (as diverse as hydroelectric, fossil-fuel, nuclear, and wind-farm plants) to a huge number of devices (as diverse as home appliances and industrial machines). This infrastructure is made of such hardware components as transformers and power lines, but it is a set of well defined standards that turn those components into a proper infrastructure. Such standards concern, for example, line voltages and frequencies, or the design of plugs and outlets.

By analogy to the power grid, a computational grid should deliver computational power to user applications. The "computational power plants" are the computers connected to the Internet, machines of different size, architecture and capabilities, ranging from single desktops to supercomputers. The Internet itself is intended to become the foundation of the computational grid, providing the basic interconnection services. Thus,

"a Computational Grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities" [1].

The power grid analogy helps us realize that this computational infrastructure will be defined by a set of standards and services: we call such a set of standards and services a *Grid architecture*. The analogy, of course, should not be pushed too far, since computing power is obviously very different from electrical power. Computing power is the power to store, transmit, and transform *information*. While electricity can only be produced and consumed, information is meant to be communicated and *shared*. A computational grid can then give us something that the power grid cannot: the ability to *collaborate* by sharing information.

Therefore, Grids must support large scale collaboration among many individuals working for different organizations towards common goals. This implies that Grid services are not limited to giving access to computers, but they must also provide means to support collaborative work.

We expect that computational grids will enhance resource utilization and allow sharing of computational results to a scale and with a transparency that far exceed those afforded by existing infrastructures such as the Web. Just as *the Internet* arose by integrating many smaller-scale internets, it is likely that a global Grid (*the Grid*) will emerge from many computational grids. The Grid, however, will not replace the Internet but rather integrate it within itself. It will also co-exist and interoperate with the Web. We may observe that the Web and the Grid, presently at least, have different implicit aims: while the Web is focused on data access and end-user interaction, the Grid is focused both on data and computation access, and on collaborative computation. However, emerging Web technologies such as Web Services converge towards the Grid approach, so that Grid and Web technologies are likely to support each other.

Several Grid projects are active now, including, but not limited to, the European DataGrid (Sec. III) and LHC Computing Grid (`cern.ch/lcg`) in Europe, GriPhyN (`www.griphyn.org`) and PPDG (`www.ppdg.net`) in the USA, ApGRID (`www.apgrid.org`) and TWGRID (`www.twgrid.org`) in Asia. Some projects, such as DataTAG (`www.datatag.org`) have the harmonization and interoperation of existing Grids as their main objective.

In the following sections we will introduce some general ideas about Grid architectures (Sec. II), then we will present one of the main Grid projects currently being developed (Sec. III) and describe some aspects of its architecture (Sec. IV).

II. Architecture of a Grid

The high-level requirements for the Grid infrastructure obviously include those that have driven the development of the Internet: *interoperability* across different hardware and software architectures, *transparency* with respect to location, satisfactory *performance* and *reliability*. More specific to the Grid is the goal of enabling very large dynamic collaborations across different administrative structures. These structures are organizations that own resources and share them with other organizations they collaborate with; the collaborations are dynamic since the quantity and qual-

The author is with the Dept. of Information Engineering (DIIEIT) at the U. of Pisa, and the National Institute of Nuclear Physics (INFN), Pisa. E-mail: `Andrea.Domenici@iet.unipi.it`.

ity of resources, of users, and of activities to carry out can vary rapidly over time.

The goal of enabling such collaborations involves support for diverse and sophisticated authentication and authorization mechanisms and policies. The foreseen size and complexity of the collaborations pose more stringent and exacting requirements on the "classic" features of interoperability, performance, and reliability. Finally, *scalability* is an essential property of any solution for problems in the domain of Grid architectures.

A. Virtual organizations

The concept of *virtual organization* (VO) is used to refer to the collaborations described above. Grids are assumed to be deployed in situations where different institutions (e.g., universities) provide resources (e.g., computers and instruments) that are used by people from the same or other institutions, who take part in different projects (e.g., scientific experiments); sharing of resources is controlled by well defined rules. A virtual organization is then a set of individuals or institutions defined by such sharing rules [2].

Let us consider a hypothetical example: the universities of Durango and Pisa link their computing centers into a Grid (DuPiGrid); some researchers from Durango, Pisa, and Belgrade collaborate in a Radio Frequency Astronomy experiment (RFA); some researchers from Helsinki and Mumbai need resources for an X-Ray Astronomy experiment (XRA); finally, a Swiss-based private company needs resources for Imaging Applications (IA).

Let us now assume that researchers from the RFA experiment have costless unlimited access to DuPiGrid, and that their data may be private or public. Researchers from the XRA experiment have costless access to DuPiGrid resources left available by RFA, and also their data may be private or public. People from the IA activity pay for DuPiGrid resources left available, and their data are private; their fees are shared between Durango and Pisa according to effective resource usage.

The above collaborations and sharing rules define three VO's (RFA, XRA, and IA).

Many variations on the example above can be envisioned: the number of possible ways to structure VO's is in practice unlimited. No matter how VO's are structured, they have some common characteristics that Grid architectures must accommodate:

- the numbers both of individual end users and of resources may be very large;
- membership to VO's is dynamic, as users may join and leave any time; resource availability is also dynamic;
- sharing rules may change in time;
- the institutions providing resources must retain full autonomous control on them.

Grid architectures support VO by providing Grid-wide mechanisms for authentication, authorization, and accounting. These mechanisms cannot replace the existing mechanisms used in the various Grid sites, since this would violate the requirement tha each site has full autonomous control on its resources. Instead, the Grid mechanisms must present a uniform and location-transparent view of the resource access policies to the users, while mapping this view to several different local mechanisms.

B. Main design criteria

The general approach to the design of Grid architectures is referred to as "The Hourglass Model": an hourglass has a wide top, a wide base, and a narrow neck inbetween. The top represents the large set of Grid applications, the base stands for the large set of architectures and technologies of the underlying resources, and the neck represents a relatively small number of services and protocols that enable applications to access the resources. Quoting from [2], the hourglass neck is

"a narrow set of core abstractions and protocols ... onto which many different high-level behaviors can be mapped ... and which themselves can be mapped onto many different underlying technologies".

The design of Grid architectures emphasizes reuse of existing technologies and the definition of protocols.

C. The layered Grid architecture

A Grid architecture [2] is best described as a layered structure whose strata are as follows (Fig. 1):

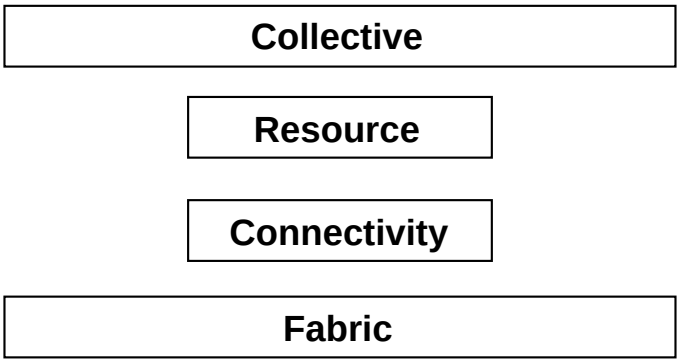


Fig. 1. The Grid layers.

Fabric : the set of interfaces to local resources. A resource may be a single computer or a cluster that behaves logically as a single resource, a disk server, a network link or interface, and the like. The software that controls the resources directly constitutes the Fabric layer.
Connectivity : protocols for secure and efficient communication, such as IP, TCP, UDP, TLS/SSL, and more.
Resource : protocols enabling sharing of individual resources. The Resource protocols enable remote access to single resources, provided by such services as information on resource state, negotiation, control, and monitoring.
Collective : global protocols to coordinate multiple resources. The Collective protocols define services that provide applications with a uniform and unified view of the Grid: applications interact with the Collective layer, which in turn coordinates the available resources to satisfy the needs of the applications. Such services include job scheduling, workload management, data replication, authentication and authorization, and monitoring.

The two middle layers, Connectivity and Resource, correspond to the neck of the hourglass.

D. The Globus (TM) Toolkit

The Globus (TM) Toolkit [3], maintained by the Globus Project (`www.globus.org`) can be considered as a reference implementation for Grid architectures. The toolkit consists in a core set of protocols and services with associated Application Programming Interfaces (API), Software Development Kits (SDK), i.e., libraries, and user interfaces. In the following, we will mention only some of the toolkit components, at the three higher levels.

At the Connectivity level, the Globus (TM) Toolkit employs the Internet protocol suite (IP, TCP...), and introduces the *Grid Security Infrastructure* (GSI) [4], [5]. The GSI extends the Transport Layer Security (TLS) protocols [6] to provide *single sign on* and *delegation*. This means that a Grid user logs on to the Grid just once and afterwards can access any Grid resource without having to be authenticated again. A program started by the user can spawn other processes, possibly remotely, delegating them the user's authorization, so that they can access resources on the user's behalf.

At the Resource level, we mention the *Grid Resource Allocation Manager* (GRAM) and *GridFTP*. The former is responsible for allocating computing resources and for controlling and monitoring the processes running on them [7]. The latter is an enhancement of FTP, providing such features as third-party transfers, parallel transfers, and integration with GSI authentication [8].

At the Collective level, we consider the *Meta Directory Service* (MDS), that gathers information from resources and makes them available to applications and to other Grid services [9].

III. The European DataGrid

The European Union has funded the three-year *European DataGrid* (EDG) project (`http://eu-datagrid.web.cern.ch`), aimed at setting up a computational and data-intensive grid of resources dedicated to the analysis of data coming from scientific exploration. This Grid is characterized as "data-intensive" since the scientific activities that will use it are expected to produce and process huge quantities of data, in the order of several Petabytes (10^{15} bytes). A large share of this data will come from High Energy Physics experiments, but also Earth Observation and Biomedical activities are involved.

The project started in January 2001 and is approaching its completion. Its main partners are:

- CERN, European Organization for Nuclear Research (`www.cern.ch`);
- ESA/ESRIN, European Space Agency (`www.esa.int`);
- CNRS, Centre National de la Recherche Scientifique (France) (`www.cnrs.fr`);
- INFN, Istituto Nazionale di Fisica Nucleare (Italy) (`www.infn.it`);
- NIKHEF, Nationaal Instituut voor Kern en Hoge Energie Fysica (Netherlands) (`www.nikhef.nl`);
- PPARC, Particle Physics and Astronomy Research Council (UK) (`www.pparc.ac.uk`).

The EDG project collaborates with national European projects, such as INFNGRID (`www.infn.it/grid`) in Italy, or UK-GridPP (`www.gridpp.ac.uk`) in the UK.

The project has achieved the goal of deploying a *testbed*, i.e., an experimental Grid composed of 12 computing centres in five European countries, with a computing power of 1075 CPU's and a storage capacity of 15 Terabytes.

IV. EDG Architecture

The EDG middleware is very complex and continually evolving, so that even a high-level description of its main components would go beyond the limits of this paper. In the following some of the basic concepts underlying this software will be introduced, without attempting to give complete and accurate descriptions of its components. Please be aware that the names used may differ from those found in the most recent documentation.

A. The EDG Fabric

The Fabric [2], [10], i.e., the physical infrastructure of the EDG testbed is a set of *farms*, clusters of ethernet-linked hosts. The the hosts are usually PC's that run Linux RedHat 6.2, currently being upgraded to RH 7.3. The hosts in a farm have different functions: some provide computing power, while other provide storage capacity. The former are called *Worker Nodes* (WN); a cluster of WN's is seen by the Grid as a single resource, called a *Computing Element* (CE). The hosts that provide storage usually control RAID pools or mass storage systems (tape libraries). Such hosts, or their clusters, are resources called *Storage Elements* (SE). A *User Interface* (UI) is a host that users connect to for accessing the Grid. The user commands that submit jobs to the Grid, monitor their execution, and retrieve computation results, run on UI's. Other machines may be dedicated to special functions, such as hosting various Grid services.

A farm does not necessarily have all possible types of hosts, and some hosts, like the UI's, may not necessarily belong to a farm. A Grid site might provide only computing power (CE's) or storage capacity (SE's). However, CE's and SE's are often linked together through a fast connection.

Each resource is controlled by a *Local Resource Management System* (LRMS), that performs such tasks as accepting jobs and dispatching them to worker nodes. Some of the LRMS's that can be used are PBS [11] and Condor [12]. The LRMS is interfaced to the Grid by a *Resource Management System* (RMS).

B. The EDG layers and services

The EDG has a layered architecture similar to the one described in Sec. II-C.

In Fig. 2 the rounded boxes represent some of the main services within the layers. Each of those services depend on services at lower levels, and possibly also on services at the same level. For example, the Job Management may be used

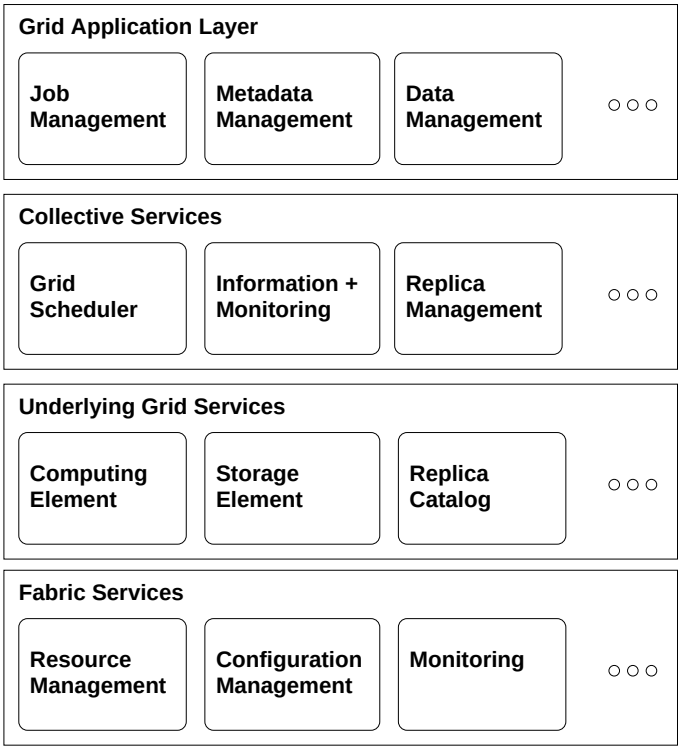


Fig. 2. The EDG layered architecture.

to sequence the submission of a group of jobs composing a single application, such as a series of computations in the analysis of a set of experimental data, each computation being executed by a different program. A description of each job, written in the *Job Description Language* (JDL) [13], is given to the Grid Scheduler, that must (i) find a CE that is available and satisfies the job's requirements (such as processor speed, memory, software environment...), and (ii) locate the data needed by the job. In order to find a convenient CE (the *brokering*, or *matchmaking* functionality), the Grid Scheduler components rely on the Information and Monitoring service, that maintains status and configuration information about the Grid's resources, such as CPU speed, available memory, number of queued jobs, and so on. In order to locate the data, the Grid Scheduler relies on the Replica Management [14] service. In the Grid, a file can be *replicated*, i.e., reproduced in several copies to optimize access time and minimize contention on the SE's. The Replica Management service keeps track of the replicas and is able to determine which replica is most convenient for a job running on a given CE. New replicas can be created, if needed. To perform this task, the Replica Management service uses the Replica Catalog (RC) service, which is similar to a distributed database holding the information needed to locate the replicas of each file.

Fig. 3 shows the dependencies implied by the interactions described above. Services are represented as packages, and arrows represent usage dependencies.

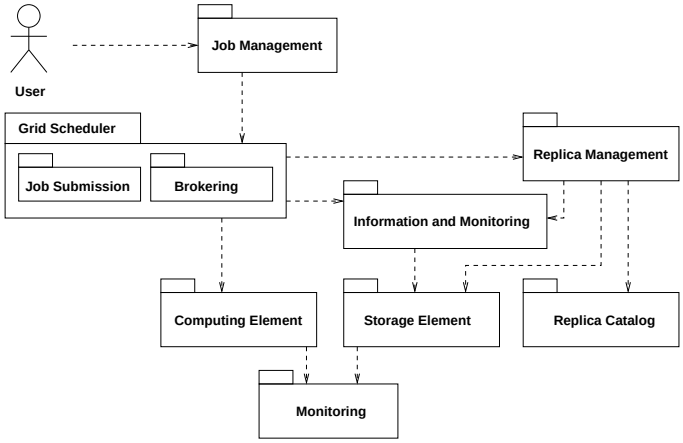


Fig. 3. Some dependencies in the EDG architecture.

C. Services and protocols

The EDG architecture brings together and integrates many pre-existing "non-Grid" components, along with third-party Grid software, taken especially from the Globus (TM) Toolkit, upon which the original EDG middleware is built. A good share of software development in EDG consists in interfacing all these components. Most components are distributed client/server systems, so each of them is defined by a *protocol* that specifies how the client and server subsystems interact, and by a *client API* that specifies how the clients invoke the service. The two subsystems are then implemented by a server process and a client library.

If we take the Replica Management service as an example, we may observe how components are integrated. One of its low-level components is GridFTP, used to transfer files between sites. GridFTP is defined by the GridFTP protocol [8] and the GridFTP client API, expressed as a set of function prototypes in C. Another component is the Replica Catalog [15]. In the first releases of the EDG software, the RC was implemented as an LDAP directory. An LDAP directory [16], [17] is a hierarchical database that is accessed remotely with the LDAP protocol. Client API's and libraries are available in various languages, and the C API and library have been adopted by EDG (and by the Globus (TM) Toolkit). The LDAP server, in turn, stores the directory in a relational database backend. The Replica Management itself does not currently have a server, hence it does not have a protocol, since most of its work is done by its subsystems, so the top-level component of this service is just a library that implements the Replica Management API. In the earlier releases, this API was in C++. The API includes operations that access the RC to manage information about replicas and use GridFTP to create or delete replicas.

Given the large number of components in the EDG software, each with its protocols and API's, it is clear that much effort has been devoted to making them fit together. In the later releases of the software, the architecture is moving towards a new model, called the *Open Grid Services Architecture* (OGSA) [18] where most services will adhere

to a uniform interfacing approach, based on *Web Services*. A Web service is a distributed application whose protocol is defined in terms of SOAP [19], [20] messages and is described by a WSDL [21] specification. This specification is software-readable and published on the Web, so that client applications can easily find the service and interact with it.

V. Conclusions

Computational grids are taking shape and already providing services to large and sophisticated communities. In particular, the European DataGrid project is entering its final stage and in the next months it is due to deliver a large Grid testbed hosting many software packages that together form a fully operational Grid architecture, optimized for handling great amounts of data from scientific experiments that involve thousands of simultaneous users from multiple institutions all over the world. A successor project, the LHC Computing Grid, is already working to fine-tune this architecture to prepare the larger infrastructure that will be used by the Large Hadron Collider (`lhc-new-homepage.web.cern.ch`) experiments scheduled to start at CERN in 2006.

This paper has introduced some of the basic concepts underlying these developments and sketched a very small sample of the architectural issues met in an important Grid project, in the hope of motivating more researchers and professionals to further the study of this exciting new field of computer technology.

Acknowledgements

This paper reports on results produced by several collective efforts whose participants number in the thousands!

I will limit myself to expressing my gratitude to three people who introduced me to the world of Grid computing: Flavia Donno (INFN), Ben Segal (CERN), and Heinz Stockinger (CERN). I also thank all my friends of the EDG Work Package 2 (Data Management), inflexibly managed by Peter Kunszt (CERN).

The INFN Grid project and the Department of Information Engineering at the University of Pisa supported this work.

References

[1] I. Foster and C. Kesselman, Eds., *The Grid – Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 1999.

[2] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the Grid: Enabling scalable virtual organizations," *International J. Supercomputer Applications*, vol. 15, no. 3, 2001.

[3] I. Foster and C. Kesselman, "The Globus project: A status report," in *Proc. IPPS/SPDP '98 Heterogeneous Computing Workshop*, 1998, pp. 4–18.

[4] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke, "A security architecture for computational grids," in *ACM Conference on Computers and Security*, 1998, pp. 83–91.

[5] R. Butler, D. Engert, I. Foster, C. Kesselman, S. Tuecke, J. Volmer, and V. Welch, "Design and deployment of a national-scale authentication infrastructure," *IEEE Computer*, vol. 33, no. 12, pp. 60–66, 2000.

[6] T. Dierks and C. Allen, "The TLS Protocol Version 1.0 (RFC2246)," `http://www.ietf.org/rfc/rfc2246.txt`, 1999.

[7] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke, "A resource management architecture for metacomputing systems," in *Proc. IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing*, 1998, pp. 62–82.

[8] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, L. Liming, S. Meder, and S. Tuecke, "GridFTP protocol specification," document, GGF GridFTP Working Group, September 2002.

[9] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, and S. Tuecke, "A directory service for configuring high-performance distributed computations," in *Proceedings of the 6th IEEE Symposium on High-Performance Distributed Computing*, 1997, pp. 365–375.

[10] "Architectural design and evaluation criteria – WP4 - fabric management," Tech. Rep. DataGrid-04-D4.2-0119-2_1, DataGrid WP4, 2001, Draft.

[11] M. Papakhian, "Comparing job-management systems: The user's perspective," *IEEE Computational Science & Engineering*, (April–June) 1998.

[12] M. Litzkow, M. Livny, and M. Mutka, "Condor – a hunter of idle workstations," in *Proc. 8th Intl Conf. on Distributed Computing Systems*, 1988.

[13] "Definition of architecture, technical plan and evaluation criteria for scheduling, resource management, security and job description," Tech. Rep. DataGrid-01-D1.2-0112-0-3, DataGrid WP1, 2001, Draft.

[14] "Data Management (WP2) Architecture Report – Design, requirements and evaluation criteria," Tech. Rep. DataGrid-02-D2.2-0103-1_2, DataGrid WP2, 2001, Draft.

[15] A. Domenici, "Notes on the usage of an experimental Replica Catalog for the CERN DataGrid Testbed," Tech. Rep., DataGrid WP2, 2001.

[16] M. Wahl, T. Howes, and S. Kille, "Lightweight Directory Access Protocol (v3) (RFC2251)," `ftp://ftp.isi.edu/in-notes/`, 1997.

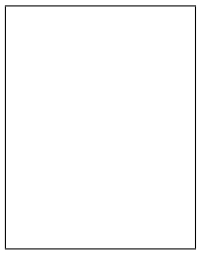
[17] T. A. Howes, M. C. Smith, and G. S. Good, *Understanding and Deploying LDAP Directory Services*, New Riders, 1999.

[18] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, "The physiology of the Grid: An open grid services architecture for distributed systems integration," Tech. Rep., Open Grid Service Infrastructure WG, Global Grid Forum, June 2002.

[19] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. Frystyk Nielsen, S. Thatte, and Dave Winer, "Simple Object Access Protocol (SOAP) 1.1," `http://www.w3.org/TR`, May 2003, W3C Note.

[20] H. Frystyk Nielsen, M. Gudgin, M. Hadley, and J.-J. Moreau N. Mendelsohn, "Soap version 1.2 part 1: Messaging framework," `http://www.w3.org/TR`, June 2003, W3C Recommendation.

[21] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, "Web Services Description Language (WSDL) 1.1," `http://www.w3.org/TR`, March 2001, W3C Note.



Andrea Domenici graduated in Electronics Engineering at the University of Pisa in 1986, where he later obtained his Ph. D. (*Dottorato di ricerca*) while doing research on logic programming languages and contributing to the development of a Gödel interpreter at the University of Bristol. He has been a researcher at the Scuola Superiore di Studi Universitari e di Perfezionamento S. Anna, Pisa, and at the University of Pisa, where he has been teaching Software Engineering since 1995. His main professional interests are currently object-oriented design, distributed systems, and Grid architectures. Activity in the latter field is carried out in collaboration with INFN, the Italian National Institute of Nuclear Physics.