Distributed General Logging Architecture for Grid Environments

Carlos de Alfonso, Miguel Caballer, José V. Carrión and Vicente Hernández

Departamento de Sistemas Informáticos y Computación,
Universidad Politécnica de Valencia,
46022 Valencia, Spain
{calfonso,micafer,jocarrion,vhernand}@dsic.upv.es

Abstract. The registry of information about the activity of applications is an important issue in Grid environments. There are different projects which have developed tools for supporting the track of the resources. Nevertheless, most of them are mainly focused in measuring CPU usage, memory, disk, etc. because they are oriented to the classical use of the Grid to share computational power and storage capacity. This paper proposes a distributed architecture which provides logging facilities in service oriented Grid environments (DiLoS). This architecture is structured so that can fit to hierarchical, flat, etc. grid deployments. The information generated during the activity in the services are scattered among the different levels of a Grid deployment, providing features such as backup of the information, delegation of the storage, etc. In order to create the hierarchy of log services, the architecture is based on discovery facilities that can be implemented in different ways, which the user may configure according to the specific deployment. A case of use is described, where the DiLoS architecture has been applied to the gCitizen project.

1 Introduction

Almost any operation which occurrs in a shared environment such as the Grid is susceptible to be registered for its late analysis. Some examples of events which are likely to be registered are the access to the services (who and when acceded), the kind of resources which are mainly used by someone who is authorised to work in the Grid, how many time has been used a resource, the changes in the state of a service, etc.

The registered information can be used later for extracting statistics about the usage of the resources in the Grid with upgrading purposes, obtaining information about the proper (or not) usage of the resources when a problem arises, debugging a distributed application, or obtaining how many resources have been used in a project for accounting purposes, among others.

In a computing environment, the most common technique for registering the activity of an application (service, daemon, etc.) in a traditional system is to create a *log file* which would contain text lines describing every action which has been performed by this application.

This model translated to a Grid environment would mean to have a lot of files pertaining to any of the resources which have been deployed. Nevertheless it would need a mechanism for mergeing these files, and thus obtaining the *log* information about the whole Grid system.

One of the main objectives of Grid technology is to provide with a great number of resources which would be frequently used by a huge number of people. In such case, the activity in the Grid, considered in a medium-long period of time, would generate a lot of registries which would be hard to store, manage or backup. Also, these data would be hard to query for obtaining useful information when analyzing the activity of the system. So, in a Grid environment, it is needed a more appropiate method for the registration of this activity.

This paper proposes a Distributed Log System (DiLoS) which defines an architecture for the registration and maintenance of the information about the activity in the system. The DiLoS system scatters the *log* files through a Virtual Organization, for obtaining features such as backup, redundancy, ease of access to logs, or decentralization, among others.

The paper is organized as follows: Section 2 discuss about the systems which are already used in Grid environment for accounting, auditory, debugging or activity registration purposes. Next, the section 3 describes the DiLoS systems, its components and the protocols used. Also it is described how the grid components would be integrated in the system. Later, the section 4 exposes a case of study about the usage of the DiLoS system in a Grid deployment oriented to eGovernment. Finally, section 5 summarizes and outlines the work which will follow to the proposal of the described architecture.

2 State of art and motivation

The registry of information is a feature which has been traditionally assumed in Grid environments. This characteristic would support features such as accounting or auditing. Nevertheless, up to now, most of the developments which address these issues only implement partial solutions for specific subjects of each project about Grid computing.

Most of these deployments are oriented to obtain accounting information about specific resource usage by the users (memory, disk, load, executed jobs, etc). The data is mainly used for being published for the rest of members of the Virtual Oganization with the aim of monitoring the state of the system, and scheduling tasks according that resource usage.

The next summary exposes a simple classification about the tools and architectures which are currently being deployed [1].

– **NASA-IPG [2]:** The system follows a common monitoring architecture for Grid environments, sensors to measure some characteristics of the resources, actuators to perform some process control actions and event services that provides a mechanism for forwarding sensor-collected information to other processes that are interested in that information. The event services represent the monitored data using eXtensible Markup Language (XML).

- **Heartbeart Monitor [3]:** The Globus Heatbeat Monitor (HBM) was designed to provide a mechanism for monitoring the state of processes and notifying the failure of them. It allows simultaneous monitoring both Globus system processes and application processes.
- **Netlogger [4]:** NetLogger proposes a distributed system which contains a central daemon called *netlogd* which receives information about the usage of specific resources (network, memory, cpu, etc.), from the applications in the system. On the other side, the applications are instrumented, using the NetLogger API, to generate log lines in the Universal format for Logger Messages (ULM), which contain the values about the monitorization of the usage of the resources during the execution of a set of commands. This project is mainly oriented to the analysis of high performance applications.
- **GMS [5]:** GMS supports resource monitoring, visualizing, and accounting. GMS was developed on top of existing Globus CORE. The system has been successfully deployed across the Nanyang Campus Grid. The system is able to capture Grid jobs information, organize and store the information into a relational database, and support analyzing and visualization of the data through the Web.
- **DGAS [6]:** It was originally developed within the EDG project and is now being maintained and re-engineered within the EGEE project. The Purpose of DGAS is to implement Resource Usage Metering, Accounting and Account Balancing (through resource pricing) in a fully distributed Grid environment. It is conceived to be distributed, secure and extensible.
- **APEL [7]:** APEL (Accounting Processor for Event Logs) parses batch, system and gatekeeper logs generated by a site and builds accounting records, which provide a summary of the resources consumed based on attributes such as CPU time, Wall Clock Time, Memory and grid user DN. The collection of accounting usage records is done through R-GMA. Each site publishes its own accounting data using an R-GMA primary producer using its locally assigned R-GMA server. To collect the data from all participating sites, data is streamed to a centralised database via a secondary producer. It is developed in LCG Project and used in the EGEE project.

Most of the projects commented above are mainly oriented to traditional computational Grids in which CPU usage, memory, disk capacity, load of the system or jobs in execution are the parameters that worth measuring. Nevertheless in a Grid environment also happens other events that should be tracked, such as *who accedes a services*, *when does it*, etc. The tools which are currently being deployed are not useful for these purpose, as they are oriented to monitor the system and not to track the services.

Moreover, current Grid trends are oriented to architectures based on the provision of general services. This means that shared resources are more heterogeneous, as they are others than CPU cycles or storage capacity. The current Grid environments are built by deploying general services, which need general *logging* facilities for enabling the track of the whole system.

Currently, there is a lack of support for these general services. The traditional systems uses *log* files to register any information which is generated by the running applications or the operations in a service from the Operative System. In this sense, Syslog [8] is the most commonly used system logging utility for UNIX systems. The applications register significant information in *log* files, which are classified by the configuration of the *log daemon*. The saved data is useful for analyzing the state of system, extracting statistics of usage or guess who has misused a resource, for instance.

3 DiLoS Architecture

The DiLoS architecture is composed by two kinds of elements. On one hand, the services which provide the log data and need to be integrated into the DiLoS architecture, and on the other hand a specific service called “Log” which will coordinate the distributed information.

The services which provide the *logs* are likely to be organized according to any distribution (such as hierarchical, cyclical, plane, etc.). Regarding the specific organization, the DiLoS Log Services (DLS) are distributed through system, so that they are properly acceded by the other services. Figure 1 outlines the architecture proposed by DiLoS, and some of the functional use cases which may happen in the system.

Each DLS is in charge of gathering the information about a set of services, which are under its scope. The scope of each DLS is defined according to the specific deployment. So, DLS do not need to be installed on each node, as each of them may gather the log information from many services. The services which provide the logs have to be configured in such way that they are able to accede the DLS that is in charge of it.

Each service saves log data in a local repository using the LOG operation. Also the DLS may have their local log repository (a DLS behaves as any other services in the system in which it is deployed).

Periodically, the services send their local registries to the DLS which is in charge of gathering its information (using the PUSH operation). Then, the DLS stores the registries of the service into a General Log repository, in order to integrate them into the Distributed Log of the Grid system.

Notwithstanding the services are the only responsible to send the information to the DLS, the data integration operation may also be initiated from the upper level, in order to *flush* the local Log repositories. The DLS would call its services for their log registries (PULL operation). This operation is part of the protocol which suggest the services to start their PUSH operation. Nevertheless PULL and PUSH operations are asynchronous. Moreover, it is not compulsory for the services to PUSH data to a DLS as a response of a PULL operation (i.e. it might not have new log entries).

On the other side, in order to provide a redundance of the information, the General Repositories managed by the DLS are also sent to other DLS which are into another scope.

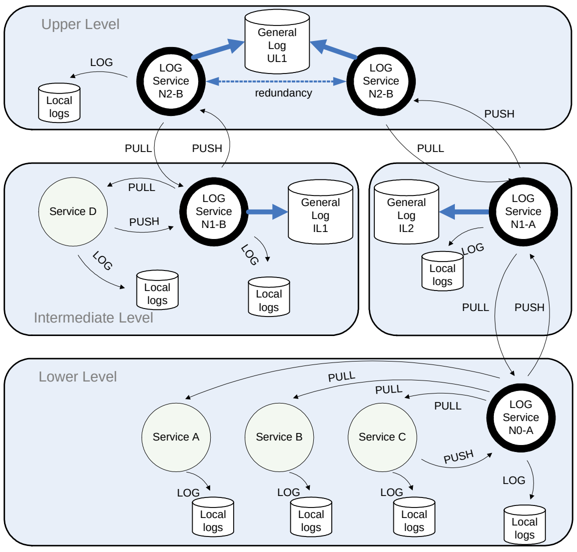

Fig. 1. DiLoS functional architecture.

The architecture considers the possibility of having more than one instance of DLS for each scope, in order to enhance performance, backup capacity, load balancing and redundancy. When a service is ready to send their log registries to the DLS, the service itself is the only responsible of deciding the instance to which is going to PUSH them.

In order to integrate the services into the DiLoS architecture, the interfaces would implement the next operations, which are also provided by the API:

- PULL: It is a public method included in general services, but also in DLSs. It is exclusively invoked by a DiLoS Log service, in order to suggest the service which implements it to send the log registries to the corresponding DLS service, deployed in DiLoS architecture.
- PUSH: It is a public method implemented only by DLSs. It is invoked by any service when they need to transmit its registries to the log service. The registries PUSHed are not changed by the DLS, as the main purpose is to store them into the general repository. The connection between different DLSs is also carried out using this operation.
- LOG: It is a private method which is implemented by general services. It is called by the service itself to locally save a log. Later it would be transmitted to DiLoS log services using PUSH operations.
- QUERY: This public method returns a set of logs which accomplish with a pattern (date, user, name service).

Although the protocol for relaying the logs to a DLS is created by the pair of calls PULL/PUSH, it is important to remark that these operations are asynchronous: the DiLoS log services would require the services for log registries by using the PULL operation, but these services are the effective responsible of deciding whether to send or not the data, and when to perform the operation.

Furthermore, it is not compulsory that the DLS to which the services PUSH the logs would be the same to which calls for the PULL operation. The services are the responsible of deciding the DLS to which send their registries, by using a specific discovery system for the infrastructure in which is deployed. The discovery of the DLS services is modelled by an operation which the user may provide for its particular organization. Some examples are (1) a static file which contains the URIs of the DLS, (2) a function which searches for services in a Globus Index Service or (3) a method which links with a specific discovery architecture.

The DiLoS architecture provides a simple implementation for DLS discovery, which is based on static files for any service, which contain the list of possible DLSs to which the service can PUSH its log registries.

In order to clarify the functionality of this operation the next fragment exposes the pseudo code of the function, which would send the log entries, using the discovery system.

```
Procedure Relay (log_block)
       LLS = Discover_log_services
       If ( is empty (LLS) ) Then
              Abort_operation
       Else
              LLS(i).PUSH(log_block)
              Save_reference_log_service(LLS(i))
       End if
End Procedure
```

The usage of the specific discovery system is the key for connecting the DiLoS architecture to any particular Grid infrastructure. When the services are deployed, they have to be configured so that they create the proper organization which may be reflected in the system. As an example, a local department may use a DLS while another department should use its specific DLS; so, the services under each scope should be configured in such way that they discover its corresponding DLS when they try to PUSH their block of log registries.

3.1 Logging Policy

According to the structure proposed by DiLoS architecture, it may seem that it tries to centralise the information into a General Log repository. Nevertheless, it is only an extreme which would be possible, by applying the facilities provided by the architecture.

The effective owners of the log registries are the services themselves, as they are in a Grid environment. Nevertheless, the usage of DLS provides mechanisms for these services to delegate the storage of such information to the DiLoS Log Services, which would be part of the Grid infrastructure.

Furthermore, each service is the effective responsible of sending the information to the DLSs, but also deciding which kind of registries are going to be sent to these Log Services. In fact, the DLS are introduced for modelling situations such as backup of information or providing more storage capacity.

In some cases, the laws also enforce the electronic transactions to be stored by some entities (such as the LSSI in Spain [9]). The DLS would also be useful for implementing such policies, as the Log services may be associated to the authorities which may store the information.

Nevertheless, it is possible to isolate a set of data (preventing its relay to other scopes) from other levels, where it may be useless.

3.2 Data saved in a log

Each service deployed in a Grid environment is the responsible of deciding which kind of information may be registered, and thus saved into the *log file*. In this sense, in order to provide support for general services, the DiLoS architecture does not force to store a strict kind of information. This decission is supported by the fact of the deployers of the services are also the responsible of providing applications to interpret the information that is registered.

So, the DiLoS architecture allows registering any type of log information. Nevertheless, the DiLoS architecture defines three basic fields which are needed to be stored for the integration and querying for data, but also an extended field in which the services can save their data in their specific format

Every registry stored by the DiLoS log system is composed by the next fields:

1. User identity: it is the identity of the effective user which has called the service, and thus is the responsible of the operation.
2. Time stamp: it defines the time and date when the service calls the LOG function.
3. Service identification: it can be a particular identification for the service which saves the log line. It is important that this field is interpreted according to the specific system deployment. An example of deployment would use WS-naming [10] as a method for services identification.
4. LOG part: it is the extended field which encapsulates the data to be register by the service.

3.3 Use Cases

The DiLoS architecture defines the interface, the architecture and the protocol which would be used for implementing a Distributed Logging System. Nevertheless, there may be a large variety of particular deployments of the DiLoS architecture, which would implement the particularities of each system. The figure 2 explains a general example of use case.

1. At first, a general service performs an operation, and it needs to register this occurrence. So, the service uses LOG interface to save a log line in its local repository (a log file).

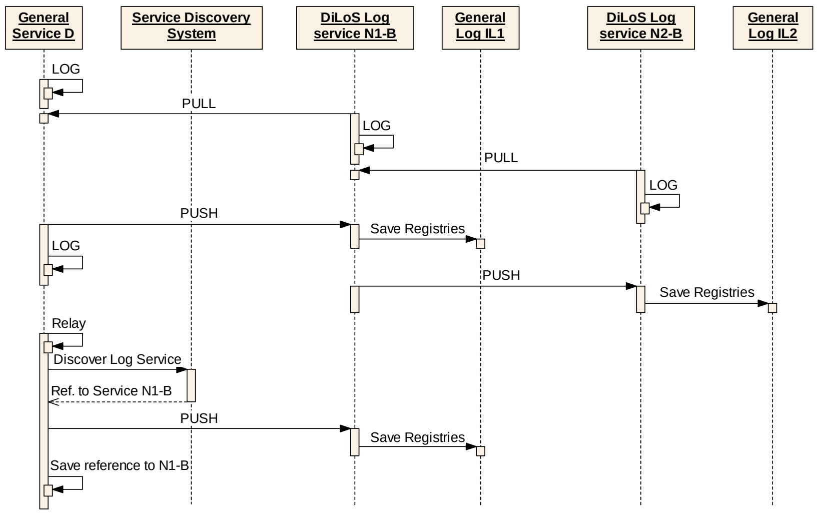

Fig. 2. DiLoS sequence diagram. Interaction among services in DiLoS Architecture.

2. A log service called N1-B, which is placed in a upper level, requires gathering the log registries of service D. So, the log service N1-B invokes the PULL operation over the service D. As any other general service it can call its internal LOG function to store some information about its operations performed. After some time, the service D calls the PUSH interface of log service N1-B. In this way, a set of registries are sent and stored in a general log repository by the log service N1-B.
3. Moreover, the service N2-B located at level 2 needs collect data from another log services placed at lower levels. The service N2-B can search for a DLS through the discovery system. The procedure continues, as the second point calling PULL operation over the service N1-B.
4. The local repository of the general service D is full, and it is needed to perform a backup of the log entries. The service D initializes the RELAY procedure, which will discover the DLS to send the block of registries, and will perform the PUSH operation.

These would be the description of the steps which would be carried on in a general use case. It is possible to summarize the use cases in the next categories:

1. Only store and not relaying (single mode): the service store log data in a local repository (LOG), but it does not use the DLS to scatter its registries. The service works as an independent entity.
2. Relay but not storing (diskless mode): the services relays every log line generated (PUSH). The information is delegated to one or several log services.
3. Store and relay all the data (backup mode): in this mode, the DLS services work as backup systems. As in the first case all the log data is stored locally,

but the information is also sent to a DLS (LOG and PUSH) in order to have a backup copy for its later recovery.

4. Store a set and relay the rest (selected mode): when the services decide that a particular information do not has to be sent to any other entity, it is stored locally this data. In other case, the information is PUSHed into another DLS (LOG and PUSH selected logs).

3.4 Application models

The main purpose of the DiLoS architecture is to provide a General Log System for general Grid services. Any application (service, daemon, activity system, etc.) generates a set of useful information which has to be available for several reasons. DiLoS cover the requirements of most of the log models in current Grid environments, performing the specific configurations.

Some of the purposes for which this architecture would be applicable are:

– Audit procedures: It is important to collect information about a period of time with the aim of identifying the global state of the system. The DiLoS architecture provides mechanisms for storing the identity of the users which use the services, and thus identifying what happened at each moment.
– Accounting: in a Grid environment can exist specific services which provide resources to the user. These resources may be meassured, and annotated who has used them for later stablishing a price of usage and thus creating a model of exploitation. The services can use the DiLoS log services to registry log lines including information about the executed jobs, time of cpu spent for each one, disk quota, permitted or deny access, etc. and any log line would be associated to the identity of the user who has called the service.
– Debugging: An application can employ DiLoS to analyze logs, and deciding whether the behaviour of the application is what was expected or not, and thus correcting it.

4 A particular implementation: the gCitizen project

As commented in previous sections, the DiLoS system can be adapted to most of the service oriented Grid deployments. The key issue is to provide the specific implementation of the functions which decide where common services have to send their log entries. As an example it is described the DiLoS customization for the middleware developed in the gCitizen project [11].

The aim of the gCitizen project is the creation of a Grid middleware for the transparent management of the information about citizens in the public administration, and the administrative procedures in which the citizens are involved, independently of the point of entry to the Administration (understood as a global entity). The vision of the project is outlined in the Figure 3.

The gCitizen project uses the Globus Toolkit 4 implementation of the WSRF standard, adding some new components which complete the architecture in order to use it in an eGovernment framework.

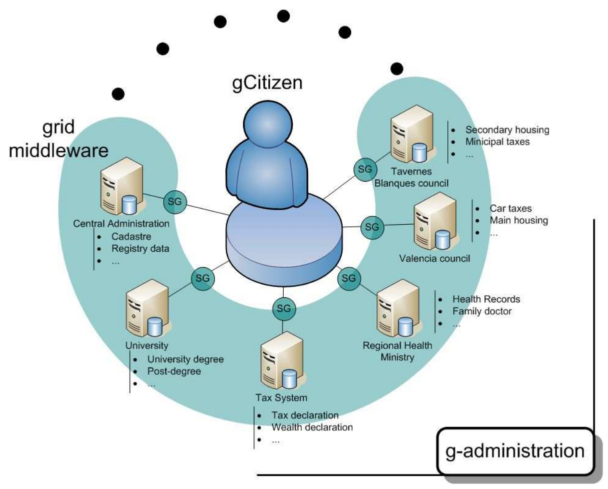

Fig. 3. gCitizen Project

One of the main components of gCitizen is a General Addressing Convention (GAC), which is used to identify the services in the system. GAC follows a similar approach to the LDAP Distinguished Names [12, 13], using the standard DN fields as C, ST, O, OU, etc. and some other additional fields added to complete the hierarchy in the public administration. It uses the DN to identify the services, but also provides some semantic information about its functionality and the hierarchical location in the gCitizen system. An example of the DNs used in the gCitizen project is:

```
/C=ES/A=Valencia/ST=Valencia/O=Diputacion de Valencia/
                OU=Gestion/CN=Padron
```

In order to find the services using this addressing convention, the gCitizen project provides the Distributed Discovery Architecture (DiDA), which enables to obtain the physical location of a service, using its DN.

The DiLoS architecture has been used in the gCitizen project, applying the DiDA discovery system and the GAC convention in order to guess the proper DLS to which each service may send its log entries.

The most suitable distribution of DiLoS services in gCitizen is to set up one DLS at each administrative level, and another one for each department. Nevertheless, some of these sections would not be able to deploy the corresponding DLS, and thus would delegate the responsibility to the DLS at upper levels.

The DiLoS architecture makes easy the customization, by detaching the discovery of the corresponding DLS as a function which may be provided by the user. In this sense, a function has been specifically created for gCitizen, making use of the GAC and the DiDA architecture.

According to the distribution of the DLS, the name of the one to which a service must send its logs is obtained by changing the CN from its DN, and using the "CN=Log" instead. For the service shown previously the name of the DiLoS service of the same level it would be:

```
/C=ES/A=Valencia/ST=Valencia/O=Diputacion de Valencia/
                    OU=Gestion/CN=Log
```

If this service does not exist or is unavailable, the service must go up in the levels of the hierarchy provided by its DN, searching a DLS at an upper organizational level. This scheme must be followed until it finds an available DLS or it reaches the top of the hierarchy:

```
/C=ES/A=Valencia/ST=Valencia/CN=Log
```

When a DLS needs to use the PULL operation, it must remove the CN from its DN, and contact every service which matches the remaining DN. As an example, the previous DLS would try to use the next pattern:

```
/C=ES/A=Valencia/ST=Valencia/*
```

The services in gCitizen log the external calls, the user who has performed the operation, a timestamp which indicates when it was carried out, and some specific information for each operation.

5 Conclusions and further work

Most of the current Grid monitoring developments are oriented to the registration of the information regarding to computing services among the Grid. Nevertheless, there is a lack of support for registering the activity in Grid deployments composed by general services.

In this sense, the DiLoS architecture has been developed. It provides a protocol and an architecture for scattering the logging information among distinct scopes in the Grid. In this sense, the DiLoS system provides elements for backing up the information, delegating its storage, etc. for the services which are deployed in the Grid.

Notwithstanding these facilities, the services are the effective owners of the information, and thus they are the responsible of deciding whether to send or not to the DiLoS Log services which are part of the logging infrastructure.

This is a complete architecture with a well defined protocol and responsibilities. It also provides an implementation which would cover common infrastructures. In order to ease the process, it is based on a discovery mechanism, which is the key to addapt DiLoS to almost any Grid organization.

Nevertheless, there are some issues which should be enhanced in order to complete the system. As an example, the recovery of the information needs a revision, in order to establish a protocol for gathering all the information which is scattered in the system. Currently, the services need to recover it from upper services, querying them about the registries generated by itself.

Another issue which would be interesting to be studied is the usage of standard formats for creating the log entries. As an example, it would be interesting to use XML further than the current *plain text* format.

6 Acknowledgments

The authors wish to thank the financial support received from the "Ministerio de Industria, Turismo y Comercio" to develop the project with reference FIT-350101-2004-54.

References

1. Zoltán Balaton, Peter Kacsuk, Norbert Podhorszki, and Ferenc Vajda. Comparison of representative grid monitoring tools. http://www.lpds.sztaki.hu/publications/reports/lpds-2-2000.pdf, 2000.
2. A. Waheed, W. Smith, J. George, and J. Yan. An infrastructure for monitoring and management in computational grids. In S. Dwarkadas, editor, *Proceedings of the 5th International Workshop on Languages, Compilers, and Run-Time Systems for Scalable Computers (LCR 2000)*, page 235, Rochester, NY, USA, May 2000.
3. The Globus Alliance. The globus heartbeat monitor specification. http://www-fp.globus.org/hbm/heartbeat_spec.html.
4. Brian Tierney, William E. Johnston, Brian Crowley, Gary Hoo, Chris Brooks, and Dan Gunter. The netlogger methodology for high performance distributed systems performance analysis. In *HPDC*, pages 260–267, 1998.
5. Hee-Khiang Ng, Quoc-Thuan Ho, Bu-Sung Lee, Dudy Lim, Yew-Soon Ong, and Wentong Cai. Nanyang campus inter-organization grid monitoring system. http://ntu-cg.ntu.edu.sg/pub/GMS.pdf, 2005.
6. Cosimo Anglano, Stefano Barale, Luciano Gaido, Andrea Guarise, Giuseppe Patania, Rosario M. Piro, and Albert Werbrouck. The distributed grid accounting system (dgas). http://www.to.infn.it/grid/accounting/main.html, 2004.
7. Rob Byrom, Roney Cordenonsib, Linda Cornwall, Martin Craig, Abdeslem Djaoui, Alastair Duncan, Steve Fisher, John Gordon, Steve Hicks, Dave Kant, Jason Leakec, Robin Middleton, Matt Thorpe, and Antony Wilson. John Walk. Apel: An implementation of grid accounting using r-gma. http://www.gridpp.ac.uk/abstracts/allhands2005/ahm05_rgma.pdf, 2005.
8. C. Lonvick. The BSD Syslog protocol. RFC 3164, Internet Engineering Task Force (IETF), 2001.
9. Ministerio de Industria Turismo y Comercio. Ley de Servicios de la Sociedad de la Información y de comercio electrónico. http://www.lssi.es, 2002.
10. Andrew Grimshaw and Manuel Pereira. OGSA naming working group. https://forge.gridforum.org/projects/ogsa-naming-wg, 2005.
11. Carlos de Alfonso, Miguel Caballer, and Vicente Hernández. gCitizen, Grid Technology for eGovernment Systems Integration. In *Proceedings of IADIS International Conference e-Commerce 2005*, pages 321–324, 2005.
12. W. Yeong, T. Howes, and S. Kille. Lightweight directory access protocol. RFC 1777, Internet Engineering Task Force (IETF), 1995.
13. S. Kille. A string representation of distinguished names. RFC 1779, Internet Engineering Task Force (IETF), 1995.