# Organization and Accessibility of Network Monitoring Data in a Grid

Sergio Andreozzi*, Augusto Ciuffoletti**, Antonia Ghiselli*, and Cristina Vistoli*
*CNAF-INFN Bologna
** Dipartimento di Informatica, Università degli Studi di Pisa
E-mail: augusto@di.unipi.it

## Abstract

*The availability of the outcome of network monitoring activities, while valuable for the operation of Grid applications, poses serious scalability problems: in principle, in a Grid composed of n resources, we need to keep record of $n^2$ end-to-end paths.*

*We introduce a scalable approach to network monitoring, that consists in partitioning the Grid into* Domains*, limiting monitoring activity to the measurement of* Domain-*to*-Domain *connectivity. Partitions must be consistent with network performance, since we expect that an observed network performance between* Domains *is representative of the performance between the Grid Services included into such* Domains*. We argue that partition design is a critical step: a consequence of an inconsistent partitioning is the production of invalid characteristics.*

*The paper discusses such approach, also exploring its limits. We describe a fully functional prototype which is currently under test in the frame of the DATATAG project.*

**Keywords***: grid monitoring architecture, grid information service, network monitoring, network service.*

## 1 Introduction

Grid-aware computations need an accurate and up-to-date view of the resources available in the Grid. One special case of resource is connectivity: a host that wants to optimize the processing of certain data replicated onto several databases needs to know which database is reachable with better performance. In order to inform applications about connectivity resources, we need to measure connectivity characteristics and make them available to applications. Packet loss rate, or roundtrip time can be considered as a valid representative of connectivity characteristics.

A scalability problem arises, since the characteristics are often end-to-end: the database containing characteristics for a grid of *n Edge Services* — we use this term to address generic storage or computing resources — should contain in principle $n^2$ distinct entries, one for each pair of

*Edge Service*. Considering that an entry may contain historical records — not only present state snapshots — one understands the size of the database is an issue.

Also additional traffic — and host workload — generated by network monitoring limits the scalability of an end-to-end approach, since each series of observations comes from a monitoring tool that runs on an *Edge Service*. When the number of concurrent monitoring sessions grows with the square of the number of hosts, they may consume a significant share of network resources, even when monitoring tools have a light computational impact (e.g, PingER [**?**]).

One solution consists in characterizing an end-to-end path by combining characteristics of the links that compose the path, thus limiting the number of entries in our database: this approach does not fit our purpose, since it is not trivial — and is a source of inaccuracy — to combine several link characteristics, possibly measured using different tools, to obtain the corresponding characteristic of a path. In addition, the availability of routing information cannot be given for granted. Instead, we want primarily direct measurements, without accessing a link-level description of system topology: the spirit of this option is empirical, mainly to improve independence from technology, and secondarily to simplify deployment.

Another way to improve scalability is partitioning a Grid into subsystems. Partitioning complex systems is a recurring idea in network architectures (one for all, Internet routing): in our context, this approach is aimed at network monitoring overhead reduction, which is obtained by assuming that certain observations are representative of connectivity between subsystems, not merely between path endpoints. The amount of network monitoring data, as well as the computational load due to the monitoring activity, are thus reduced.

Network monitoring schemes that use a *site* concept for a similar purpose have been proposed: such concept is biased by organizational aspects, and by DNS naming. We argue that, in order to produce significant results, a partitioning aimed at network monitoring optimization must reflect the connectivity of the Grid infrastructure: organizational aspects or DNS naming often reflect inappropriate aggregations.

More precisely, in order to be useful for network monitoring optimization, a partitioning of a Grid into *Network Monitoring Domains* (or *Domains* for short) must satisfy the following requirement:

**Requirement 1** *For each pair of* Domains $(D_1, D_2)$*, connectivity within the boundaries of* $D_1$ *and* $D_2$ *is negligible compared with connectivity between the boundaries of* $D_1$ *and* $D_2$*.*

In essence, partitioning should be such that packets are usually lost in the path between *Domains*, where they also spend much of their transmission delay, and communication bandwidth is significantly higher within *Domains*, where there is a limited probability of bottlenecks.

In theory, the partitioning of a weighted graph — where nodes $S$ are the Grid resources and links $L \subseteq S \times S$ are network paths with an associated connectivity $C : L \to \mathcal{R}$ — into *Domains* that satisfy the above statement is an easy task: given a threshold connectivity $C_t$, we delete from the graph all links $l$ such that $C(l) \leq C_t$. The residual connected components correspond to *Domains*. However, the application of such simple rule may be problematic in practice. One lesson learnt from the history of routing protocols is that the choice of the weights associated to the links — the *connectivity* — is a matter of compromise between contradictory requirements. In addition *Domains* may cross administrative boundaries: this fact introduces organizational issues concerning the control of a *Domain*.

The network monitoring scheme we introduce in this paper is based on the option of partitioning the Grid into *Domains*, and the design of an appropriate partitioning is considered as a prerequisite. We understand that the implementation of such prerequisite involves a number of important details, which cannot be evaluated without an experimental testbed. The main objective of our work is to provide an insight and a testbed to evaluate the applicability of Grid partitioning to improve the scalability of a *Grid Monitoring Architecture*.

In section 2 we detail and justify our approach, introducing some terminology and a conceptual framework. The proposed architecture is compared with that introduced by GGF GMA [**?**] and with NWS [10]. The architecture explicitly deals with some relevant issues:

- monitoring activity management, which consists in configuring and coordinating network monitoring activity;
- management of differentiated services.

Section **??** and **??** introduce the data structures that describes the network monitoring architecture. The effects of a failure of requirement 1 are also discussed.

Section 3 outlines the architecture of a prototype, which is currently matter of experiments in the frame of the DATATAG European Project [**?**].

## 2 Architecture of a Grid Information Service

We want to design a modular Grid Information System architecture, starting from GGF GMA architecture [**?**], giving a fine grain description of the content and of the organization of network monitoring data, explicitly oriented to the *Domain* partitioning. A modular approach allows us to identify and design those components that, when integrated in a pre-existing GIS, enable the management and the publication of connectivity characteristics. Successful experiments in this sense have been carried out with Globus MDS [**?**] (see section 3), and EDG R-GMA [6].

We identify the following components of a network monitoring GIS (see figure **??**):

- the *topology database*, that stores and makes available to users the description of the Grid partitioning (e.g. which *Domain* contains a certain computing service);
- the *monitoring database*, that describes the planned monitoring activity in the Grid;
- the *production engine*, that stores and makes available the characteristics of Grid services and fabric (e.g., packet loss rate between computing and storage services);
- the *producers*, that query the *monitoring database* and the *topology database* to configure their network monitoring activity. Observations are published through the *production engine*;
- the *consumers*, that find the *Domain* they belong to, as well as those of other services of interest, querying the *topology database*. Observations are retrieved using the *production engine*.

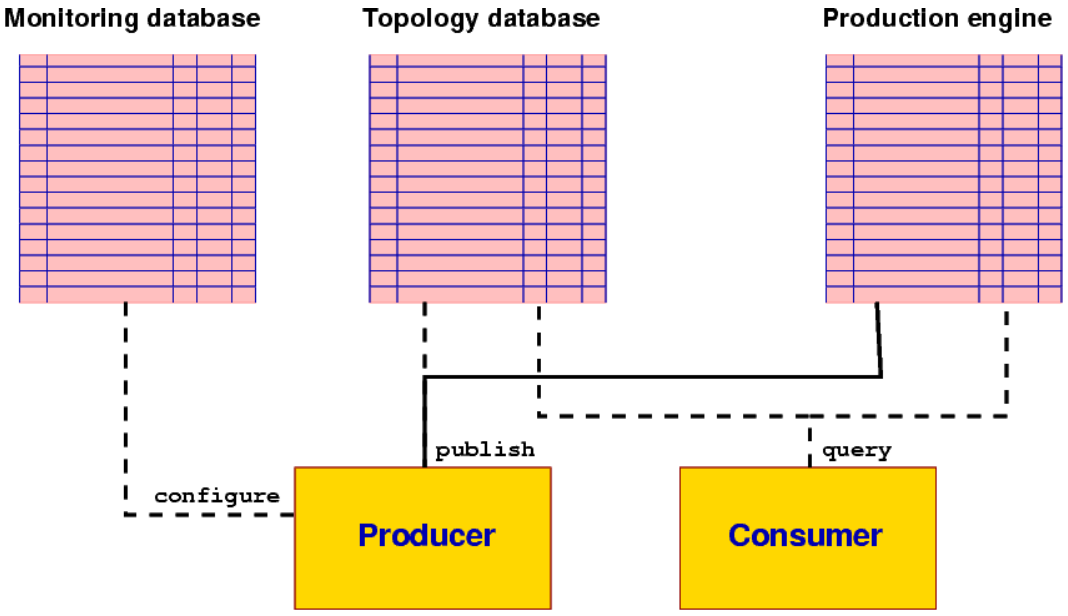


**Figure 1. Modular architecture of a GIS: dashed lines represent read-only access**

We support the option of using three distinct *databases* by outlining how different is the use *Producers* and *Consumers* make of their content:

- the *monitoring database* is accessed infrequently by *producers*, that download the description of their monitoring tasks. This may happen once during a monitoring session, or periodically. This kind of access is read only, and should be restricted on a *per producer* basis;
- the *topology database* is accessed rather frequently, since each *consumer* query may require access to this database, to determine where services are located in the Grid. Instead, *producers* can easily cache relevant information and seldom access the database. These accesses are read only, but access cannot be easily restricted, since any *consumer* can in principle access any part of the database;
- the *production engine* is the heart of the system: it must store and publish thousands of records for each session, index the available observations to speed up *consumer* queries, while periodically accepting new observations from *producers*. Accesses are for both read and write operations: read access should be extended to all potential consumers, while write access should be restricted on a *per producer* basis.

To show the validity of the above modularization, we check it against the internal structure of the Network Weather Service [10], one of the more complete *Grid Information Systems*. *Producers* correspond to **sensor** hosts, each characterized by certain monitoring skills, that include network monitoring. The *monitoring database* consists of **nwsControl** objects, that define NWS **cliques** of sensors that perform mutual monitoring. The *production engine* consists of NWS **memories** that store data, and NWS **forecasters** that process this data to produce answers that match *consumer*'s needs, that are extrapolated from measurement **series** stored by **memories**. The

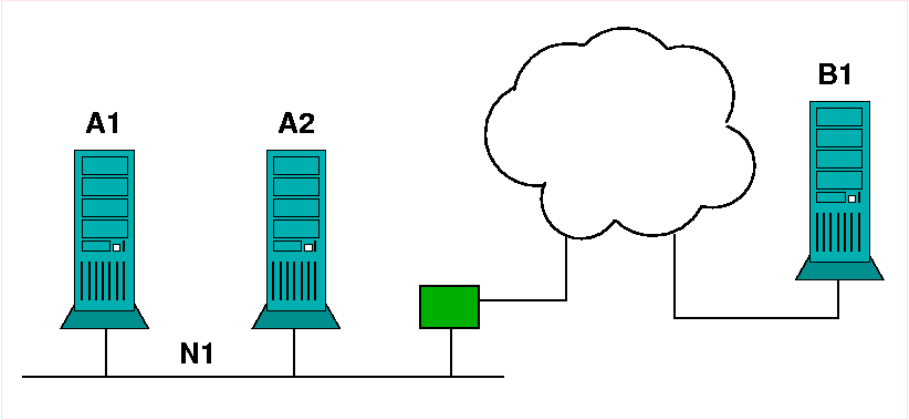


**Figure 2. A use case for domain partitioning**

system lacks a real *topology database*, which is in part implemented by **cliques**, in part relies on the mapping from **sensors** to IP addresses — from which we can infer that two sensors are in the same DNS domain, whatever this may mean. All functionalities and data are tightly packaged in a monolithic product, that can be controlled using simple Unix commands.

Other *Grid Information Systems* privilege the *production engine* so that one is tempted to identify a GIS with its *production engine*: MDS for Globus MDS, R-GMA for EDG R-GMA. While we understand that the success of a GIS mostly depends on the performance of its *production engine*, we argue that this is just a component of a more complex design.

Compared with other designs of *Grid Monitoring Architectures*, the one introduced in this paper gives a clear view of its modular structure: this is the first step towards an improved inter-operability among different *Grid Information Systems*, as discussed in section 3.

The next sections consider the content of the three *databases*, with an emphasis on the *topology database*.

## 3 Structure of the topology database

To understand the role of the *topology database*, we consider the following use case (see figure **??**). Let hosts $A1$ and $A2$ be hosted by the same over dimensioned LAN $N1$, and host $B1$ be reachable by $N1$ through a complex route, that may contain bottlenecks. Assume that $A1$ and $B1$ periodically monitor the route between $N1$ and $B1$ — as if they were members of the same NWS clique. Despite the fact that the connectivity between $A2$ and $B1$ can be easily inferred from that between $A1$ and $B1$, none of the existing GIS — NWS for one — is able to take into account this fact. The hint that may come from their IP address, or from DNS naming, is definitely deceptive.

Our approach uses some knowledge about Grid structure: we understand that *Edge Services* are inter-connected by a fabric of heterogeneous links, whose connectivity varies in a wide range. The connectivity of individual routes is mainly affected by those links that exhibit a poor performance, while other links have a marginal influence. The concept of *Domain* is introduced to exploit this aspect of a Grid fabric: given an end-to-end path between two *Edge Services* included in distinct *Domains*, an optimal measurement strategy focuses on the inter-*Domain* fabric, and disregards the intra-*Domain* fabric. So we define a *Network Service* between each pair of *Domains* that can

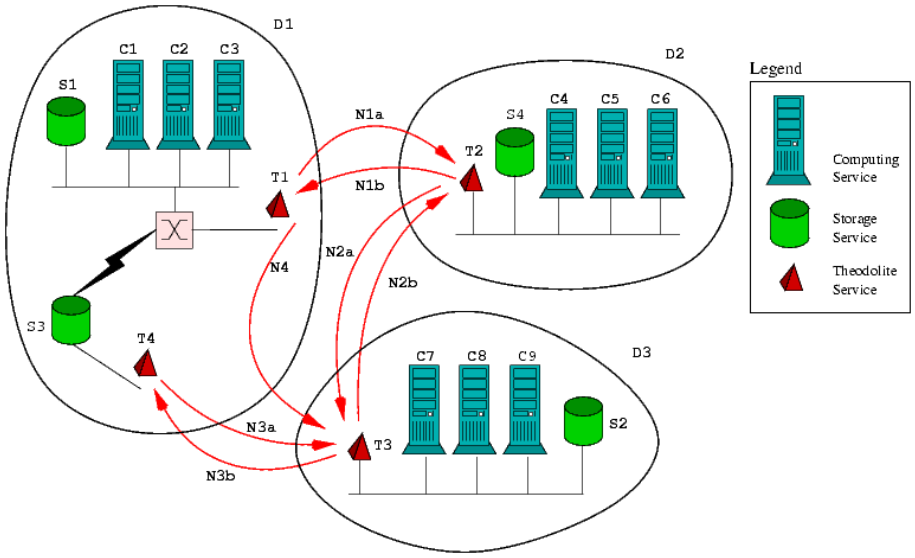


**Figure 3. A partitioning of a grid into domains**

communicate, and disregard routing.

The dimension of the network monitoring problem is still a square, since — at least in principle — all pairs of *Domains* should be monitored, but it is now based on the number of *Domains*, which should be significantly smaller than the number of *Services*.

In order to monitor *Network Services*, network monitoring tools should run in appropriate sites, from where they can observe the behavior of the fabric between the *Domain* boundaries, with minimal interference on/from traffic and workload within the *Domain*. Network monitoring tools offer to Grid applications a peculiar service, that we call *Theodolite Service*.

For instance, (see figure 1) a few sites might decide to offer a number of *Edge Services* (computing services $C1 - 3$ and storage services $S1$ and $S3$), and provide appropriate, dedicated and over-dimensioned high performance connectivity between the hosts where *Edge Services* are located, so that these services make a *Domain*, $D1$. The connectivity with other *Domains* (represented by arrows) might be based on an infrastructure that is considered appropriate, but is neither dedicated nor over-dimensioned (e.g., leased lines): these are the *Network Service* of the Grid. *Theodolite Services* $T1$ and $T4$ might be located on the gateway itself, or on hosts with a direct and fast connection with the gateway.

Since the location of a *Theodolite Service* can be dependent on the monitored *Network Service*, a *Domain* may contain several *Theodolite Services*, each specialized in the monitoring of an outgoing branch. This specialization can be also appropriate to avoid the interference between internal traffic and traffic generated by monitoring tools.

### Communication classes and multi-homed hosts

Grid connectivity may be supported by several communication classes [**?**]: for instance one that is specialized in handling real-time activities, and another that is specialized in processing reliable transactions [2]. Since the performance figures for each class can be very different, even for a single level-2 link, routes that offer multiple classes of service are represented as distinct *Network*

*Services*. It is a consequence that *Edge Services* connected by such *Network Services* may be distinct, and associated to distinct *Domains*.

For instance, in figure 1 both the *Network Services* $N3a$ and $N4$ connect domain $D1$ to $D3$: $N3a$ might exhibit characteristics appropriate for interactive applications, while $N4$ might be more appropriate for playback applications.

A more complex case is the following: assume a *Storage Service* offered by a site has access to a network managed by another site, that offers both a high quality connectivity (for instance IP-Premium [4]) and an ordinary Best Effort service. Under some circumstances, it may be appropriate that the *Storage Service* is split into two. This case is exemplified in figure 1 by the *Storage Services* $S3$ and $S4$, which may correspond to the same storage, yet reachable with different connectivity: the host supporting $S3/S4$ might share the same administration with $C4-6$, but the high quality connectivity with $C1-3$ might justify the inclusion of $S3$ in $D1$.

Our *topology database* should be able to represent such hosts, that we call *multi-homed*, since their case resemble the familiar one of a host that can be reached through distinct level-2 interfaces. Note that nodes that host *multi-homed* services do not route packets between interfaces: otherwise, there may be no justification for a split service.

### 3.1 Events that invalidate *Domain* partitioning

In order for *Domain* partitions to be consistent with requirement 1 the performance of the fabric inside *Domains* should be better than the performance offered by the inter-*Domain* fabric. Although we argue that this is a quite natural state for an healthy Grid, nonetheless we must take into account that this requirement can be violated due to anticipated or unanticipated events. Some of the events that should be taken into account are: a) a traffic burst degrades the performance of an internal link; b) an hardware failure or routing change alters the characteristics of an internal link; c) an upgrade of an inter-domain link enables some components of the inter-*Domains* fabric to outperform some *Domain* fabric.

We divide these events into two classes: *internal events* (case a) and b) above), that are due to the degradation of the fabric supporting connectivity inside a *Domain* and *external events* (case c) above), that are due to the upgrade of the intra-*Domain* fabric. The two classes have distinct effects, and distinct strategies should be used to cope with them.

To support such strategies the intra-*Domain* fabric should be continuously monitored in order to assess its performance. The results of such activity should not be published for Grid applications use, but used as an internal benchmark, and compared with the corresponding characteristics of the inter-*Domain* fabric.

This activity should be carried out by the hosts that support *Edge Services*, and should monitor links that are used as part of the intra-*Domain* fabric.

#### Discovering and Coping with Internal Events

A monitoring tool of a *Theodolite Service* measures the performance of a *Network Service*, which is a path composed of external and internal links between two *Domains*. An *internal event* occurs when the internal links — not of those that are indirectly monitored by the *Theodolite Service* — are a bottleneck for the Grid fabric.

In that case certain *Edge Services* inside the involved *Domains* may appear nearer than they are: in fact, the presence of the internal bottleneck is not revealed by the observations published by the *Theodolite Service*. As a consequence, applications that use this information to select resources may exhibit performances significantly worse than expected.

*Detection* is obtained comparing internal and external observations, and also from those applications that find that a *Edge Service* does not exhibit the expected performance: they should inform the *Theodolite Service* which is responsible of the *Network Service* about the possibility of an internal bottleneck.

One way to *recover* from an *internal event* is to indicate as unreliable the observations for those *Edge Services* that exhibit a degraded performance because of the internal bottleneck. This operation is carried out by the *Theodolite Service*, that indicates the anomaly to the Grid Information System. A simple and effective action might be to indicate unreachable such services.

An alternative consists in inducing a kind of back-pressure: the degraded performance of an internal link is reflected by the *Theodolite Service* of the degraded *Domain* by lowering the published performance of the *Network Service* edged to this *Domain*. This option might need some sort of cooperation between the *Theodolite Services* at the edges of the same *Network Service*.

#### Discovering and Coping with External Events

An external event occurs when the fabric between two *Domains* is upgraded, and outperforms the internal fabric of the connected *Domains*. The *Theodolite Service* detects the improvement, and publishes according to the new link performance, but does not take into account that the internal fabric of the domain is not adequate to support the new service level.

Much like in the previous case, *Edge Services* in the *Domain* may appear nearer than they are: several internal bottlenecks may appear at the same time. In summary the situation is comparable, of even worse, than that explained in case of internal events.

*Detection* is similar to the case of internal events: either a *Theodolite Service* detects the inconsistency, or a Grid application signals a failed expectation.

The *recovery* obtained by tagging the *Edge Services* as unreachable seems inapplicable in this case, since this might produce a sensible degradation of the Grid operation as a consequence of an improvement of the fabric. Also the indication of the observation as unreliable might induce a more subtle form of degradation.

Since external events are, with infrequent exceptions, somewhat planned, the best way to cope with them seems to be to avoid their occurrence: this can be obtained by upgrading the internal fabric before the external one, by trading a lower share of a common resource when it is upgraded, or by reorganizing the *Domain* partitioning around an improved link.

### 3.2 A UML model of the *topology database*

We want to give a formal assessment of the content of the *topology database* described in the previous section. To this purpose, we use a UML class diagram.

A UML class diagram is a graph, whose nodes are *classes* that are conceptually equivalent to those introduced by Object Oriented programming: objects of a given class are characterized by values assigned to *attributes*, and are manipulated using *methods*. Relationships between objects are represented as *associations* between the corresponding classes. Apart from generic ones, there

are two kinds of peculiar associations: those that *aggregate* many objects of one class to make an object of another class, and those that bind a *subclass* to its parent class, from which the subclass inherits attributes and methods.

The UML class diagram of figure 2 describes the *topology database*, that is used by Grid applications and by monitoring tools in order to discover the location of other services, and to access the Grid Information Service.

In the next sections we first describe its *classes*, and next introduce *associations*.

### Classes

We define classes for each of the *Services* introduced in the description of the Grid layout. They are organized in a hierarchy that highlights common features and relevant differences:

***Service*:** it is a superclass that represents any service a Grid offers to Grid applications.

***Edge Service*:** it is a superclass that represents a service that does not consist of connectivity, but is reached through connectivity. Since this kind of resource is usually addressed by a level-2 address, we indicate this address as an *attribute* of this class.

***Storage Service* and *Computing Service*:** they are subclasses of *Edge Service*. Their *attributes* include a reference to the *production engine* records that contain the relevant characteristics of the provisioned service — like amount of available storage and access time for *Storage Services*, or processor share availability for *Computing Services*.

***Network Service*:** represents the fabric between two *Domains*. Its *attributes* include a *class*, corresponding to the offered service class, and a statement of expected connectivity. Other attributes should be used to characterize the *Network Service*: they are discussed in section **??**.

***Theodolite Service*:** a *Theodolite Service* monitors a number of *Network Services*: its *attributes* include the *class* of *Network Service* it monitors. The service offered by a *Theodolite Service* is not apparent, since its role is the observation of network characteristics. One option to make it visible to Grid applications is to allow the *on demand* observation of a metric: an application might be authorized to ask the *Theodolite Service* to refresh a given characteristic, by executing the appropriate network monitoring tool.

We introduce two further classes that represent aggregations of services:

***Domain*:** represents the partitions that compose the GRID. Its attributes include the *class*, corresponding to the *connectivity class* offered by its fabric.

***Multihome*:** represents an aggregation of *Edge Services* that share the same hardware support, but are accessible through distinct interfaces.

The *ConnectivityClass* and *IPaddress* classes represent respectively different kinds of connectivity offered (like BestEffort or Premium IP), and Internet addresses.

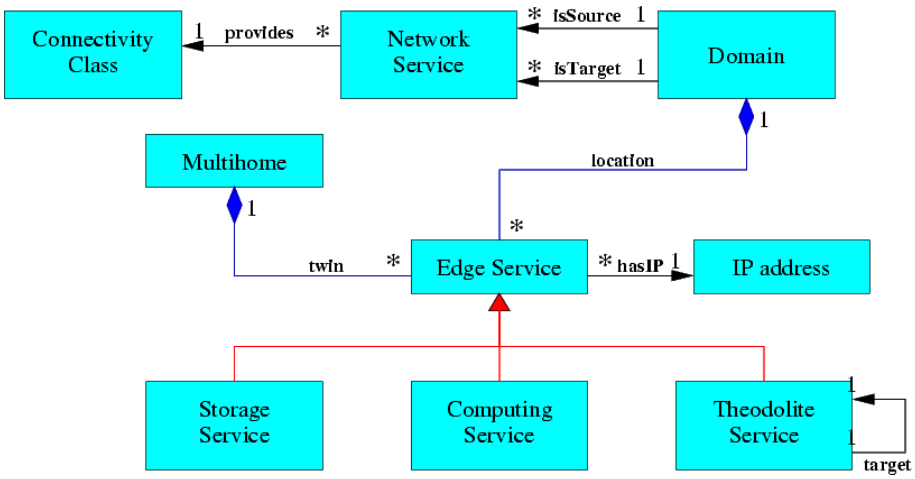


**Figure 4. The UML diagram of the topology database**

### Associations

Inheritance associations are used to classify *Services* into *Edge Services* and *Network Services*, and *Edge Services* into *Computing Services*, *Storage Services*, and *Theodolite Services*.

A key role is played by associations that *aggregate Edge Services*:

**location:** all *Edge Services* are included in a *Domain*: this relationship is represented by the *location* association, that aggregates several *Edge Services* into a *Domain*. Each service can be located in several *Domains* characterized by distinct classes of service.

**twin:** *Edge Services* can also be aggregated according to their hardware support: this is justified when they are accessible through different network interfaces, so they can be differently aggregated to *Domains*. The *twin* association binds a *Service* to at most one *Multihome* that represents its hardware support.

Generic associations are used to represent relevant relationships between *Services*:

**source and destination:** a *Network Service* represents the fabric between two *Domains*: the *source* and *destination* associations reflect this. A *Domain* can be associated as *source* or *destination* of many *Network Services*; instead, each *Network Service* has unique *source* and *destination Domains*;

**target:** the monitoring activity of a *Theodolite Service* is performed in cooperation with another *Theodolite Service*. This fact is represented by the *monitors* association, that binds two *Theodolite Service*: one which runs the test, and another that responds.

## 4 Structure of the Monitoring and Production Databases

The UML class diagram in figure 2 outlines the content of the *monitoring database*, which is rooted on the *Theodolite Service* node of the UML class diagram of the *topology database*.

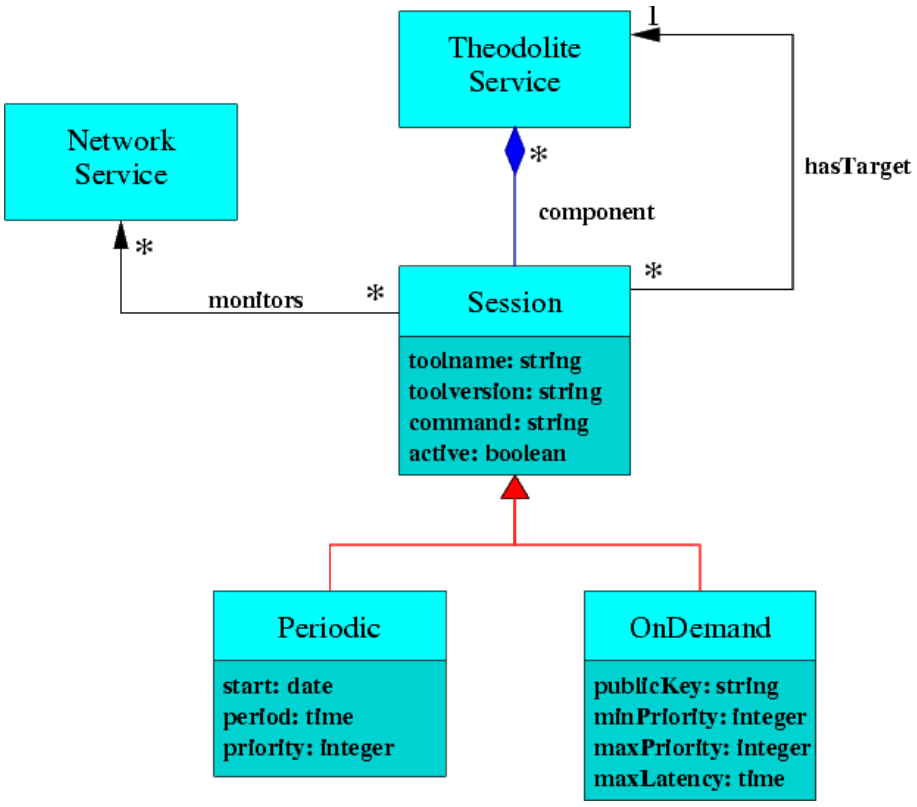


**Figure 5. The UML diagram of the monitoring database**

Each *Theodolite Service* is decomposed into several monitoring *Sessions* subclassed into *Periodic* and *OnDemand*. The attributes of *Periodic* and *On demand* sessions allow the control of the monitoring activity: tool specific configurations are an attribute of a *Session*.

The *production engine* contains the descriptions of *Network Services*, *Storage Services* and *Computing Services*. Their representation as UML diagrams is part of an international standardization effort, in the frame of the Grid Global Forum (see the work of the Grid Schema Working Group [**?**]): for instance, an association might exist between *Storage Service* and a *partition* class, that describes the characteristics of a disk partition made available by the *Storage Service*.

We represent only the *entry point* of such complex data structure, whose design is not related with the concepts discussed in this paper.

## 5 Architecture of a prototype implementation

A prototype implementation has been implemented that is based on the above concepts. We paid special attention to modularity, to allow partial redesign or replacement of parts that become obsolete, or appear as unfit.

In particular, we opted not to implement the *production engine*, but reuse one of the existing ones. We successfully implemented separate adaptors for MDS, the LDAP-based GIS of Globus, and R-GMA, an SQL-based GIS proposed in the frame of the European DATAGRID project. Figure 3 shows the prototype architecture with the MDS adapter. The interface to the *monitoring*

and *topology* databases has been also cleared from SQL-specific details: a sketch of its content is in section **??**.

According with such modular approach, the architecture of the *network monitoring host* is divided into a GlueDomains part, and an MDS specific part: in figure 3 they are separated by a dotted line, the GlueDomains part being on the left side.

The *GlueDomains* side is composed of a hierarchy of processes:

**GlueDomains** is a daemon process that controls the whole monitoring activity of the host. It spawns the processes that implement the *theodolite services*. The description of the *theodolite services* is obtained querying the *monitoring database* hosted by the *GlueDomains* server, each time a theodolite service is spawned. The query returns the list of all theodolite services that are associated with any of the IP addresses of the host.

**Theodolite** is a process that implements a *theodolite service*. It spawns — and re-spawns when needed — all monitoring sessions associated with a *theodolite service*. The description of all *sessions* associated with the *theodolite service* is retrieved from the *monitoring database*. The identifier of the monitored *Network Service* is retrieved from the *Topology Database*, given the identifier of the *theodolite* and of the *target* associated with the *session*. The *theodolite* may interact with the GIS adapter to initialize the publication of the observations for those *Network Services*.

**Session** is a process that implements a monitoring session. All parameters that configure a specific session are passed from the theodolite process, so that the session should not need to access the *monitoring* or *topology* databases. The session interacts with the GIS adapter to record the observations in the *production engine*.

We distinguish two kinds of *controls structures* for a session:

- a *periodic* control, like the usual pinger;
- an *on demand* control, which receives a trigger from another process: for instance, an `iperf` server process, which bounces packets from a client. Such kind of control may also help the GRID-wide scheduling of monitoring activities, which is useful for expensive tests, like those used for bandwidth measurements.

The right part of the *network monitoring host* in figure 3 is MDS-specific. The design of the R-GMA specific part is simpler, and is omitted.

A flow of LDIF entries is generated by the functions provided by the MDS adaptor. Such flow is re-ordered and buffered by a *MDSproxy* process, which runs as a daemon. Periodically, the buffer is flushed by the GRIS host using the `GDIP` command, called through `ssh`.

To gain a further insight in the prototype architecture, we describe the interface offered by the GIS adapter to implement *theodolite* and *session* processes. Such interface abstracts from the characteristics of the specific GIS (MDS or R-GMA in our case), and provides the designer with a GIS-independent view of the publication process. Such description complements the *producer-oriented* view given in figure 3 with the tools offered to the *consumers* of network observations.

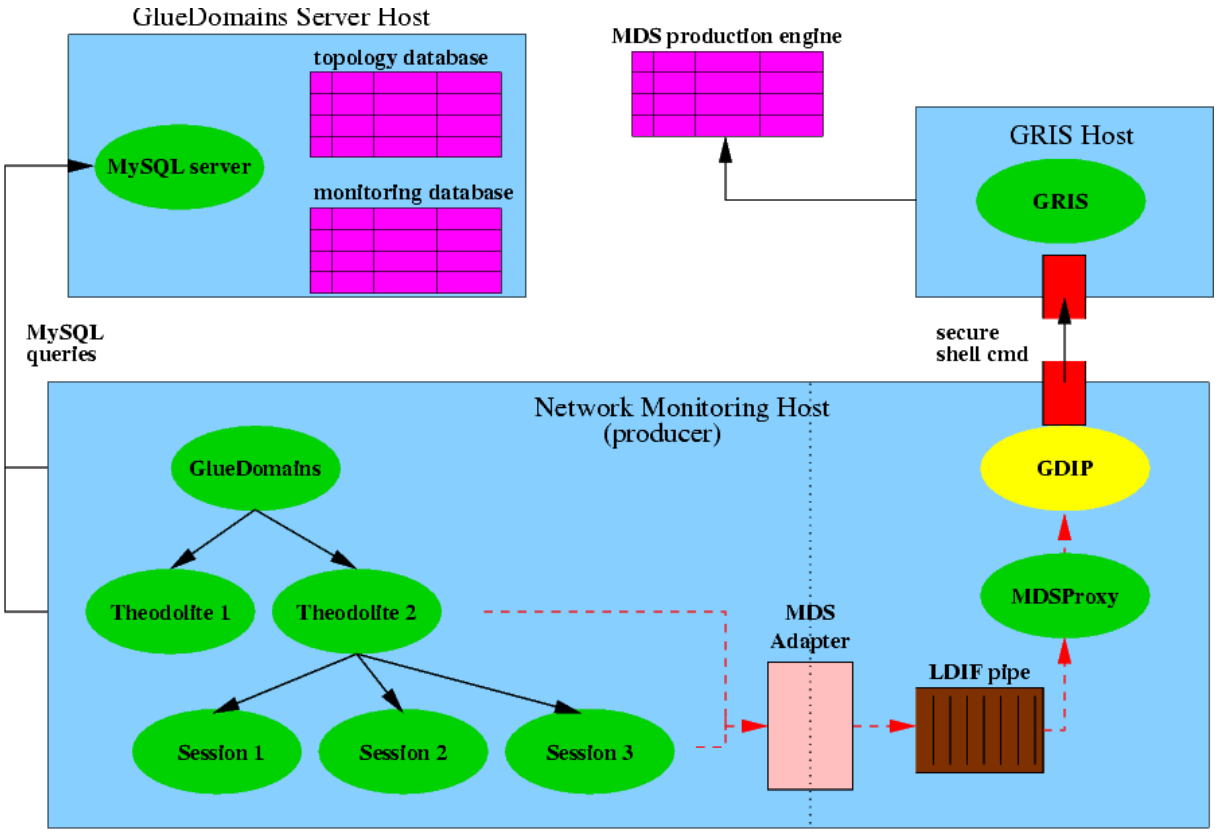


**Figure 6. Prototype architecture (with MDS adapter)**

### 5.1 An API to query the *topology database*

Although very flexible, the *topology database* structure described in section **??** is designed having in mind a model of how applications and tools use the content in its tables. To make explicit such model, we describe the API functions that are offered to access the *topology* database.

As outlined in section 2, users of a Grid Information System can be classified into *consumers* and *producers*:

- a *consumer* wants to read *Observation* records to evaluate the quality of the fabric between two *Edge Services*;
- a *producer* wants to store *Observation* records to make them available to applications.

The two roles are sharply distinguished: *consumers* perform read operations, while *producers* write new *Observations*. This consideration simplifies the structure of the basic API functions, and update operations are not considered. We outline the interface offered to such these two categories of users.

#### The API for *consumer* applications

A typical *consumer* is a Grid application that wants to characterize the connectivity between the two *Edge Services*. We assume that it knows the identity of the *Edge Services*: this information may be either pre-defined, or obtained querying the *production engine*.

In order to enable the application to query the *production engine* for the *Observations* of interest, the API should return the *Network Services* — possibly more than one — between the *Domains* the *Edge Services* belong to. More precisely, we distinguish three, non mutually exclusive, cases:

- the *Edge Services* belong to the same *Domain*;
- the *Edge Services* are multi-homed, and have an interface in a common *Domain*;
- the *Edge Services* belong to distinct *Domains*.

Three distinct functions cover these cases, and we briefly describe their implementation: they have in common the call parameters, that consist of the source and target `edgeService`s.

The function `getDomainList` returns a list of `networkDomain`s that contain both *Edge Services*: this result is obtained by selecting `edgeService`s with requested `edgeServiceId`s and identical `domainId`, and returning a list of objects corresponding to these `domainId`s.

The function `getMultihomeList` returns a list of 3-ples: the first two items are alternate `edgeService`s that are multi-homed with the source and the target, respectively, and the third item is the `networkService` among the two. The result is obtained by selecting first the `edgeService`s multi-homed with the source and the target, and next the `networkService`s between these *Edge Services*.

The function `getNetworkServiceList` returns a list of the `NetworkService`s between the source and target service. This result is obtained by selecting from the `networkService` table those that connect two *Domains* that contain respectively source and target service. The list of `NetworkService`s is then used to query the observation database.

It is appropriate that the results of these queries are cached, and periodically refreshed, to avoid unnecessary computational and communication load. The performance can be further optimized, in case the database is replicated for read operations, since all of the above queries can be directed to a read-only replica.

### API for *producer* applications

A typical *producer* is a monitoring tool embedded in a *Session* that wants to store in the *production engine* a record containing an observation, obtained measuring some network characteristic between the local host and a target *Theodolite Service*.

This operation needs to know the `NetworkService`, that is used to index the *Observation* record in the *Observations* database. It returns a list of 2-ples each composed of

- another `theodoliteService`, that is one of the targets of the requesting `theodoliteService`, which is passed as a parameter;
- the `networkService` between the two.

As in the case of the *consumer* application, the *producer* application should cache the result of this query, and use it for successive write operations. In our prototype, this query is performed by the *theodolite* process, and its result is passed to spawned *sessions*.

## 6 Conclusions

Partitioning a Grid into *Domains* is a way to limit network monitoring overhead: this indirectly improves Grid scalability, and usability of network monitoring output. However, a precondition for the application of such option is that intra-*Domains* fabric is characterized by a higher connectivity, with respect to inter *Domain* fabric: this is the basic requirement for *Domains* partitioning.

We explore this approach, and evaluate its applicability: a prototype implementation is at the core of a *proof of concept*, in the frame of the DATATAG project [**?**]: the code and installation instructions for the *GlueDomains* prototype are at [1].

## References


[1] Homepage the of GlueDomains project. http://www.di.unipi.it/∼augusto/gluedomains.

[2] A. Alessandrini, H. Blom, F. Bonnassieux, T. Ferrari, R. Hughes-Jones, M. Goutelle, R. Harakaly, Y.T. Li, M. Maimour, C.P. Pham, P. Primet, and S. Ravot. Network services: requirements, deployement and use in testbeds. Technical Report D7.3, DataGRID, 2002.

[3] Sergio Andreozzi, Natascia De Bortoli, Sergio Fantinel, Antonia Ghiselli, Gennaro Tortone, and Vistoli Cristina. Gridice: a monitoring service for the grid. In *Third Cracow Grid Workshop*, Cracow, Poland, October 2003.

[4] Mauro Campanella. Implementation architecture specification for the Premium IP service. Technical Report Deliverable 9.1, DANTE - GN1(GEANT), 2000. http://www.dante.net/geant/public-deliverables/GEA-01-032av2.pdf.

[5] A. Cooke, A. Gray, L. Ma, W. Nutt, J. Magowan, P. Taylor, R. Byrom, L. Field, S. Hicks, and J. et Al. Leak. R-GMA: An information integration system for grid monitoring. In *Proceedings of the Eleventh International Conference on Cooperative Information Systems*, 2003.

[6] Andrew Cooke, Werner Nutt, Ari Datta, Roney Cordenonsi, Rob Byrom, Laurence Field, Steve Hicks, Manish Soni, Antony Wilson, Xiaomei Zhu, Linda Cornwall, Abdeslem Djaoui, Steve Fisher, Norbert Podhorszki, Brian Coghlan, Stuart Kenny, David O'Callaghan, and John Ryan. R-gma: First results after deployment. In *CHEP03 - Computing in High Energy and Nuclear Physics*, La Jolla - USA, february 2003.

[7] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid Information Services for distributed resource sharing. In IEEE Press, editor, *Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10)*, Aug 2001.

[8] R. Harakaly, P. Primet, F. Bonnassieux, and B. Gaidioz. Probes coordination protocol for network performance measurement in grid computing environment. *to appear, Journal of Parallel and Distributed Computing Practices, Special Issue on Internet-based Computing*, 2002.

[9] The Globus Team. Globus Toolkit 2.2 MDS Technology Brief, Jan 2003. Draft.

[10] Rich Wolski. Dinamically forecasting network performance using the network weather service. Technical Report TR-CS96-494, University of California at San Diego, January 1998.