

# The Palantir Grid Meta-Information System

F. Guim<sup>1</sup>, I. Rodero<sup>1</sup>, M. Tomas<sup>2</sup>, J. Corbalan<sup>1</sup>, J. Labarta<sup>1</sup>  
 Barcelona Supercomputing Center<sup>1</sup>,  
 Universitat Politècnica de Catalunya, Computer Architecture Department<sup>2</sup>  
 {francesc.guim, irodero, julita.corbalan, jesus.labarta} @ bsc.es<sup>1</sup>  
 miquelt@ac.upc.edu<sup>2</sup>

**Abstract**—Grids allow large scale resource-sharing across different administrative domains. Those diverse resources are likely to join or quit the Grid at any moment or possibly to break down. Grid monitoring tools have to adapt supporting access information to these heterogeneous and not reliable environments. There is a wide range of types of resources to be monitored, with different nature, characteristics and so on. For instance Grid users want to monitor from which is the state of the jobs that submit till how many processors has a given resources of a given centre. These issues make the task of monitoring complex to treat, and it is difficult to provide a general ways for accessing to all this information.

In this paper we discuss the main characteristics of a subset of the monitoring tools currently available in terms of their extensibility, interfaces and security facilities they provide. We propose a set of functionalities that a Grid Information System should provide. We describe the Palantir meta-information system that has been designed for uniform the access to different monitoring and information systems and that implements all the discussed functionalities. This meta-information system tries to abstract the resource information that underlies on the local systems, and homogenize the access to all this.

We present the architecture of Palantir and its main characteristics: on the first part of it description it is presented the API that Palantir provides to the end users, including the semantic for each of the methods and some examples of usage; and the second part provides a general architectural description of its components.

**Index Terms**—Grid, Job monitoring, resource monitoring, information systems

## I. INTRODUCTION

As other Grid middleware applications, monitoring services has to adapt to the characteristics of Grid architecture. It should provide uniform access to information and resources, ways to discover which capabilities it has and how to access it. However it is not easy, because Grid environments are volatiles, heterogeneous and not reliable. We define a resource as any Grid entity, software or hardware that is able to provide information, for instance a resource can be from a host till a user job.

This work has been supported by the HPC-Europa European project under contract 506079 and by the Spanish Ministry of Science and Education under contract TIN2004-07739-C02-01.

In the eNANOS[1] architecture we needed a new component that allows accessing to all the information related to the entities that are involved in our system, i.e: Grid jobs, local processes, resources and so on. This component had to integrate and merge information from the bottom part of the Grid, coming from the local components of the centres, such as the NANOS scheduler, with information coming from the top components of our architecture, in our case from the eNANOS Broker. However, the nature of the information that it will provide is much diversified, not only it will provide monitoring information from the monitoring system, or job monitoring system, it had to provide information from other nature, for instance performance predictions. On the other hand, this new component had to be extensible, in sense that if new information has to be provided it has to be easy to extend its features. For achieve this goal, our first approach consisted on having a deeper study of the available monitoring tools that have been deployed till the moment and try to adapt the more appropriate one. However, we realized that any of them matched all the requirements that we had. Using our recent experience in grid monitoring and information systems obtained in [2][3], we designed a kind of meta-information system that should implement all the needed functionalities. Something important to remark is that the presented system is not intended to substitute of other existing monitoring or information tools. Furthermore, it is intended to uniform the access to the information provided by them providing easy and powerful mechanisms for gathering data from different sources, and providing mechanism for implement new information modules in case that it is needed.

The goal of this paper is to provide a description and characteristics that our meta-information system has in terms of functionalities and architecture.

In the first part of the paper we present the analysis that we have carried out of a set of monitoring tools and works that we consider more relevant. Mainly we discuss the following aspects: architecture of the system; the extensibility and interfaces provided; and finally, security issues. Many studies about the state of the art of Grid Tools already exist in the literature. However our goal is to present a more pragmatic point of view about the existing tools, including the possible extensions, metrics availability and so on. The second part contains the description of the Palantir system: we present its architecture and the application programming interface that is provided to the clients, its semantic and functionalities.

## II. RELATED WORK

In this section we provide a general analysis of a set of monitoring tools that we have considered more representative. As explained before we will discuss three main aspects:

- **Extensibility:** we strongly believe that monitoring systems by default do not use to provide all the information that user or brokers need. So they should provide mechanism for extend them in the case that is required to monitor new resources or data. Not only they should be extensible, they should provide easy mechanisms for implementing such extensions.
- **Interfaces:** the provided interfaces for access to the information it provides. They should provide simple and generic API for query information from other programs or entities of the Grid, such as local schedulers or brokers. The basic mechanism for using interfaces could be used by programs or applications that are running on the same host, such linking programs with a given library, but more complete features are desirable, for instance, in our environments, where applications are distributed among the Grid, provide remote access API is fundamental.
- **Security:** allow security when accessing to the resources information it is a crucial issue. Security not only in terms that the connection to the monitoring systems should be done with secure protocols in the case that the queries are done over the TCP/IP stack, they should take into account aspects as accounting and user privileges. It is pretty common that some resource can only be queried by users or applications that have been granted before, for instance only the user that have submitted a job or an administrator can know its status.

We focused on the following monitoring tools, which are the most commonly used: Globus Monitoring and Discovery Service (MDS)[5][6], GridLab Mercury[7], Network Weather Service (NWS) [8], Crossgrid OMIS Compliant Monitoring service for the Grid (OCM-G) [9][6] and Ganglia [11].

### A. Monitoring and Discovery Service

MDS is the Grid Information Service provided in the Globus Toolkit [12]. It supports monitoring sensors to register to the Information Service.

MDS is composed of two types of components: information providers and data aggregation services. The information providers are present at the host end of the monitoring hierarchy whereas the data aggregation services are the internal nodes.

This architecture provides a hierarchy over different administrative domains to form Virtual Organizations (VO). Data is updated periodically through the hierarchy, but not frequently enough to get fresh monitored data at the user end for several kinds of metrics (eg CPU usage). What is more, only simple shell scripts are available at the host end in MDS; so to get these fine-grain resource-level metrics, we have to add new sensors.

It is indeed possible to add new sensors and new metrics to MDS, either by using command line shell scripts, or by coding C or Java modules.

The interfaces provided for these new sensors are LDAP (Lightweight Directory Access Protocol, [13]) in MDS2, OGSA (Open Grid Services Architecture,[14]) in MDS3 and MDS4. The corresponding data formats are the LDAP Data Interchange Format, LDIF (MDS2) or XML (MDS3).

Authentication with GSI is possible via LDAP or GSI (Grid Security Infrastructure, [15]) certificates. Once authenticated, authorization only works at VO level; fine-grain access restriction has to be enforced outside the Grid, thanks to a web portal for example. Data transmission can be encrypted through Secure Sockets Layer / Transport Layer Security (SSL/TLS) implementation.

### B. Mercury

GridLab [16] is a European research project developing application tools and middleware for Grid environments. They produce a set of application-oriented Grid services and toolkits providing capabilities such as dynamic resource brokering, monitoring, data management, security, information, adaptive services and more. Among them, the Mercury Monitor is designed to satisfy requirements of Grid performance monitoring.

Like MDS, Mercury is a hierarchal monitoring service, organized with Local Monitors at the host end, and Main Monitors. The last one can be used as proxies on cluster's gateways and so allow monitoring across different administrative domains.

To build the hierarchy, Local Monitors should be registered offline by the administrator in configuration files, because Mercury has no discovery method.

Multiple Main Monitors may connect to the same Local Monitors making it easy to implement load-balanced and fault-tolerant hierarchies. What is more, only requested data are exchanged between monitors, so the intrusiveness is reduced.

New modules can be added by programming a Mercury module in C. These modules can be new sensors (in Local Monitors) or new data processors to implement complex metrics (in Main Monitor). The metrics added can have parameters, and return either scalar (integer, string, etc) or complex results (arrays or records).

Mercury allows several authentication methods: by IP, through GSS-API (Generic Security Service, [17]), or by new modules that can be added. Once authenticated, the user has access to the metrics following a resource-level authorization. Encryption layer is supported.

Many metrics are provided, allowing an accurate resource-oriented monitoring. Mercury is the only tool to provide such fine-grain access policies. Finally you can extend the monitoring tool by adding modules, even if the module format is currently not documented.

### C. Network Weather Service

Network Weather Service (NWS) is a monitoring system

designed for short-term performance forecasts. NWS provides a set of system sensors monitoring end-to-end TCP/IP performance, available CPU percentage, and available memory. Based on collected data, NWS dynamically characterizes and forecasts the performance of network and computational resources.

NWS has four types of components: a Directory Service where other components register, Permanent Storage Servers, Resource Monitors and Forecasters.

Even if there is a discovery method, the Directory Service cannot be distributed, and each VO will have its name server. It is also impossible to have a hierarchical system, and so it cannot work through different domains alone. However, it can be integrated in other tools, since it uses LDAP, like MDS.

To register to the name server, NWS uses LDAP. No new metric can be easily added in NWS. This feature is planned to be added. However you can add LDAP services which could register to the Directory Service. The data format used by the Monitors is non standard and data are only accessible through a command line tool.

Concerning security, NWS authentication is made to the name server via LDAP certificates. No authorization process exists inside the VO seen by the name server.

#### D. G-PM/OCM-G

The OMIS (On-line Monitoring Interface Specification, [18]) is an API aiming at defining a standard interface for communication between middleware applications. The OCM-G (OMIS-Compliant Monitoring system for the Grid) is an application monitoring tool, which is part of the Crossgrid project [19]. Crossgrid is also providing G-PM (Grid-oriented Performance Measurement tool) which is a graphical performance analysis tool that allows requesting standard performance metrics as well as user-defined metrics at runtime.

OCM-G has been developed for RedHat and may also run on other Linux 2.4.x distributions. It has a distributed architecture: Application Monitors are linked into every monitored Application Processes; Local Monitors gather information at the host level; a Service Managers (SM) per site collects data and finally a Main Service Manager (MainSM) publishes the data outside the Grid.

G-PM allows a distributed on-line analysis of raw data to provide user-defined metrics at runtime, whereas usual approaches (Pablo, Paraver, etc) only support a centralized and offline analysis.

Internally, OCM-G uses Globus IO communications. To publish its data to the external tool (an analysis or a visualization tool), OMIS interface is used. Application measurements are made by putting trace calls and recompiling.

OCM-G uses GSS-API authentication, OMIS service-level authorization and RSA-based encrypted connections.

#### E. Ganglia

Ganglia is a distributed resource-oriented monitoring system for high-performance computing systems such as clusters and

grids. It relies on a multicast-based listen/announce protocol to monitor state within clusters and uses a hierarchy amongst representative cluster nodes to federate clusters and aggregate their state. Ganglia also comes with a web frontend to visualize the metrics evolution.

Ganglia has a hierarchical architecture made of gmonds and gmetad daemons. Inside clusters, gmond daemons monitor changes, collect local datas, announce relevant changes and listen to the state of other daemons via a multicast channel if possible. Gmetad daemons use a tree of point-to-point connections amongst representative cluster nodes to aggregate the state of the entire Grid. To get information, gmetads poll following a Round-Robin algorithm amongst the available daemons under themselves in the hierarchy. Thus the load is distributed and the system is reliable.

Time and value thresholds can be defined for every kind of resource, to reduce data exchange due to data update.

As Mercury, Ganglia has no registration or discovery protocol: every daemon must know at startup (via a configuration file) which other daemons are possibly under him in the hierarchy.

Ganglia provides a simple way to add metrics: the command line program gmetrics.

Communication between agents uses XDR over UDP inside clusters and XML over TCP between gmetads and their sources.

Only Anonymous IP authentication is available. There is no authorization method, as long as you see the machine you are asking information. So, all the security has to be enforced elsewhere.

In the following sections we present the Palantir system: its data model, the global architecture characterization, and finally the interfaces provided to the final users of the system.

### III. PALANTIR DATA MODEL

Figure 1 shows the conceptual data model used in the Palantir system. The entities represent the conceptual parts of the systems that can contain information suitable to be requested. They are not required to be physical resources. For example, hosts and jobs are considered to be entities. Each entity has associated a set of Metrics. They contain specific information about the entity to who are linked, for instance the metric *elapsed time* is a metric that can be associated to the entity job.

Each entity type has a set of instantiations: for example the entity *host* may the instantiations "*host1.bsc.es*", "*host2.bsc.es*" etc. Final users query metrics of specific entities instantiations.

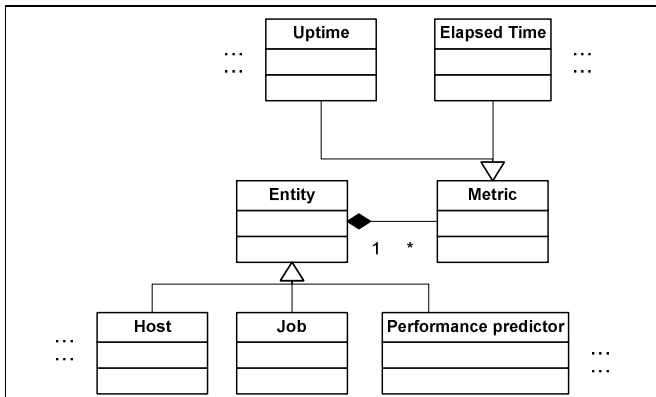


Figure 1: Palantir Data model

#### IV. PALANTIR SYSTEM ARCHITECTURE

This section provides a general description of what is the architecture of the Palantir meta-information system that we designed. In [4], the Global Grid Forum Performance Working Group developed a model for Grid monitoring tools, the Grid Monitoring Architecture (GMA), we have based the design of our system on such proposal. Figure 2 provides a general view of the system architecture. As can be observed there are three top architectural components: Palantir Access Point, the Palantir Gateways and finally the information modules. In the following subsections their main characteristics are presented.

##### A. The Palantir Access Point

The first layer is the Palantir Meta-Information System access point. It manages the queries that are done by the users or applications redirecting them to the appropriate Palantir Gateway. This redirection is based on two different facts:

- If monitoring information is queried, then the center where the queried entity remains. The unique requirement is that the client must provide which information has to be gathered, for example: *uptime* for the host *host1.bsc.es*.
- If discovery information is required, for example, the entities of type *host* available on the whole system. The query may be split to many queries to different gateways.

The protocol used from the application to this access point is based on the generic API presented in the following section. The client has not to be aware if the final queries are done to the MDS or to the NWS system, the API provides a generic authentication and information access methods. More important is that client does not have to be aware to which information system the queries are done, abstracting it to the complexity of the underlying systems.

##### B. The Palantir Gateways

The second layer of components that are located on the local centers, are the Palantir Gateways. They are responsible to carry out the queries to the information systems and monitoring system modules.

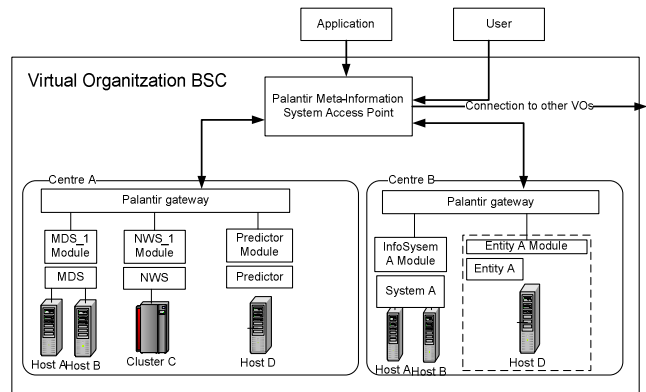


Figure 2: Palantir global architecture

The main task of the gateway is choosing the appropriate module that will process the query. It stores a data base with all the entities that are available on the systems, and when entity information is required it searches in such database where it can be retrieved.

The gateway distinguishes two kinds of entities:

- Persistent entities. Entities that are non-volatile or that in case that there are no problems (network, a host is down etc.) will remain available for a long time. Examples of this kind of entities are: host, network, cluster etc. For these entities the database stores among other data in which information system are located, for example in Figure 2 the Palantir Gateway of the *Center A* will know that the entity "*Host A*" can be queried in the module with id "*MDS1*". Regularly the data base is updated with the creation or destruction of entities in the information systems.
- Non-persistent entities. Entities that are volatile or with a limited amount life time. For example: jobs. These kind of entities are identified with a composed key that contains information about where they can be found. Basically, a subset of its key must identify a persistent entity that will be used to find out in which module the query has to be done. For example entity "*job1@hostD*" has the key *job1+hostD*, then the gateway knows that the persistent subkey is *hostD* and can found in which module the information about the entity can be found. For this kind of entities no information is based on the data base.

The gateways have a cache mechanism to avoid unnecessary communications with the local modules. The TTL of such data can be tuned by the system administrators, big TTL can be useful for entities whose data is not likely to change for a while, for instance static information for the entity *broker*, such as *policy*, *ip*, *name* etc. On the other hand small or null TTL can be applied to very dynamic entities such as the entities *job*.

##### C. Monitoring and Information Systems modules

They are the components present in the bottom layer of the architecture. Modules are responsible of carry out the final

queries to the information or monitoring systems that they represent. The modules receive queries based on the generic format presented in the following section and they take the appropriate actions for answer them.

Obviously, at this level modules must know how the queries have to be done to the systems. For each monitoring or information system that can be queried through the Palantir, a module must be implemented.

It is important to remark that all the modules must provide a set of common functionalities for support the requirements explained in the following section. The queries to the modules are done over TCP/IP using XML documents as containers of the queries, thus allows modules to be implemented without restrictions of programming language, structure, performance constrictions etc. This allows to the modules developers the freedom to implement as preferred.

#### D. Current status

We are currently working on the implementation of the overall architecture. The actual version of the system has support to resource monitoring and job monitoring. We have already implemented access to three different information systems:

- The resource monitoring is provided by two modules that interact with the monitoring systems MDS and ganglia.
- The job monitoring is provided by one module that interacts with the different components that are present in the eNANOS architecture. This module allows gathering job monitoring information coming from two different levels: grid and local level. The grid job information is provided by the eNANOS broker, and the local job monitoring information is provided by the local eNANOS scheduler and the CPUManager.
- The predictor system provides performance predictions for the applications that can be executed in the system.

Figure 3 presents the global view of the current architecture of the eNANOS Architecture, and how the Palantir information fits in it. The eNANOS broker and schedulers are data providers for the “eNANOS Architecture Module”. However, at the same time, they are data consumers, and can gather information from the Palantir system. More detailed information about the eNANOS architecture can be found in [1].

The presented architecture gives access to the different components that can provide information on the Grid in a generic way. An important characteristic is that if a new entity or system that is suitable to give information is added to the Grid, we are able to implement a new module that will allow users or applications accessing to it using the same API used for the rest. On the other hand each time that a new entity is added a new module has to be implemented.

## V. PALANTIR INTERFACES

This section provides the description of the application programming interface that we have designed for accessing the

meta-information system. It has been designed for be generic and allowing carrying out queries of different kind of resources or Grid entities.

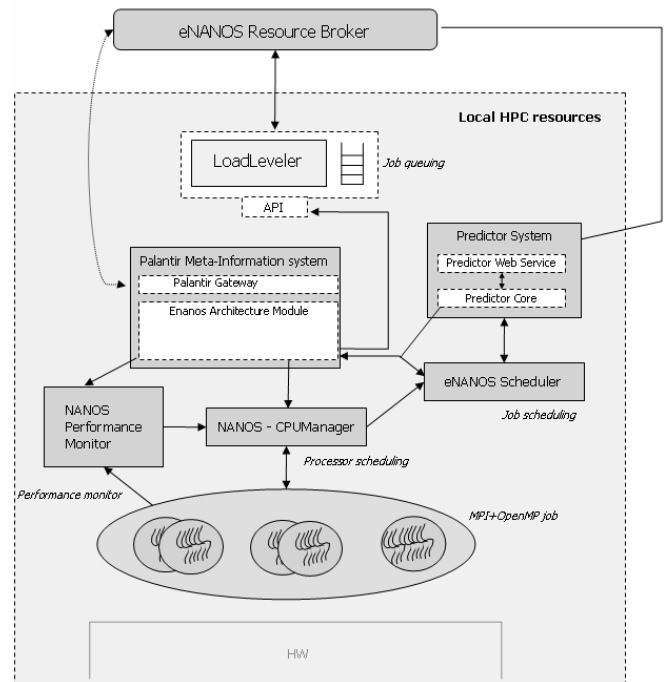


Figure 3: The Palantir in the eNANOS architecture

The API is divided in three main parts. The first one is a set of methods that allows starting and ending communications between authorized components and the meta-information system. As we have explained before, the authentication and accounting are very important challenges that must be taken into account. The second set of methods allows discovering which type of features and methods are available. The system is intended to be extensible, so it is important if new features are added the user-end or the applications that are using it would be able to discover and learn how to use them. And finally, the third set of methods is oriented to retrieve the information from the system.

The data format that is used by all the methods is XML, because it is a well known standard allowing easy filtering through XSL transformations, but more important its extensibility will allow adding other features of functionalities.

#### A. Connecting to the information system

These mechanisms allow the user to be authenticated against the system. The accounting can be used if the underlying systems allow them. However, the security object always will be used for carry out a secure connection. Two main methods will be provided:

**StartCommunication:** opens a secure connection with the monitoring tool.

**Input:** user id and certificate.

**Output:** a temporary security object (user's security credentials); an error code if the authentication failed.

**CloseCommunication:** closes the connection and invalidates the temporary security object.

**Input:** the security object.

**Output:** an error code if the object was invalid.

### B. Discovering the available features

The meta-information system, as it is providing a wide range of information, and it's using several underlying systems, provides ways to discover how to query it and what information can be gathered. Below are presented all the methods that have been defined for provide these functionalities.

**GetResources:** Returns the list of the types of resources available on the system. It is important to clarify that this list will be a set of abstract resources not a specific physical resources, for instance it would return the resource type "host" instead of host "host1.bsc.es".

**Input:** the security object.

**Output:** an XML document containing the list of resources available on the system (for example, Host, Clusters, Jobs). Each resource will have associated a human readable description, and a XML Schema describing how the resource is identified. An example of a returned XML is shown in the Source 1 .

**GetResourceInstantiation:** Returns the list of instances of a provided entity type. For instance, user may know the list of entities of type "host". Each instantiation will be returned following the XML Schema format that describes how to identify such resource.

**Input:** the security object and the type of resource.

**Output:** XML document containing the list of instantiation Grid entities or resources that match the provided type. An example of the returned XML is shown in the Source 2.

**GetMetricInfo:** Returns information about a particular metric of a particular entity or resource.

**Input:** the security object, the type of resources and the metric name.

**Output:** an XML document that will contain a human readable description of the metric, parameters that must be provided when querying it, the description of the available filters that can be used applied to the given metric (filters are explained in the following subsection), and the available notifications. For instance, calling getMetricInfo("application","prediction\_job\_memory\_usage") we could obtain the XML document presented in Source 4.

If the underlying systems allow accounting, the queries would take into account the user privileges, for instance in the case that a user queries for the list of resource instantiation of an resource "hosts" the returning XML would be a list of jobs on which he/she has been granted for retrieve information.

```
<AvailableResources
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  <!-- List of resources availables -->
  <Resource name="host">
    <Description>This resource allows to carry
    out queires about the resource
  host/Description>
    <key>
      <xs:element name="host">
        <xs:complexType>
          <xs:choice>
            <xs:element name="ip" type="xs:string"/>
            <xs:element name="hostname"
              type="xs:string" minOccurs="0"/>
          </xs:choice>
        </xs:complexType>
      </xs:element>
    </key>
  </Resource>
  <Resource name="application">
    <Description>This resource allows to carry out
    queires about the software resource
  application/Description>
    <key>
      <xs:element name="application">
        <xs:complexType>
          <xs:choice>
            <xs:element name="name"
              type="xs:string"/>
            <xs:element name="version"
              type="xs:string" minOccurs="0"/>
          </xs:choice>
        </xs:complexType>
      </xs:element>
    </key>
  </Resource>
  <!-- ETCETERA -->
</AvailableResources>
```

#### Source 1: Sample of output for the method GetResource

```
<ResourceInstantiation name="host">
  <host>
    <hostname>pcmas.ac.upc.edu</hostname>
  </host>
  <host>
    <hostname>kahfre.cepba.upc.edu</hostname>
  </host>
  <!--ETCETERA -->
</ResourceInstantiation>
```

#### Source 2: Sample of output for the method GetResourceInstantiation

##### Document 1:

```
<Metrics Resource="host">
  <Metric>
    <Name>uptime</Name>
    <Description>Returns the uptime of the given
    host.</Description>
  </Metric>
  <!-- ETCETERA -->
</Metrics>
```

##### Document 2:

```
<Metrics Resource="application">
  <Metric>
    <Name>hosts</Name>
    <Description>Returns the list of host that have
    install the application.</Description>
  </Metric>
  <!-- ETCETERA -->
</Metrics>
```

#### Source 3: Sample of output for the method GetMetric

```

<Metric>
  <Name>prediction_job_memory_usage</Name>
  <Description>Returns a prediction of the memory
that a given application will use if
executed.</Description>
  <Parameters>
    <xs:element name="Parameters">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="AppName"/>
          <xs:element name="Host"/>
          <xs:element name="User"/>
          <!-- ETCETERA -->
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </Parameters>
  <Filters>
    <Filter name="state" type='string'>
      <description> Allows to filter jobs by its
state </description>
    </Filter>
    <Filter name="submit_time" type='integer'
units="seconds">
      <description> Allows to filter jobs by its
submission time </description>
    </Filter>
    <!-- ETCETERA-->
  </Filters>
  <Notifications>
    <Periodic available="no"/>
    <Punctual available="yes"/>
  </Notifications>
</Metric>

```

**Source 4: Sample of output for the GetMetricInfo**

### C. Getting the entity information

User/application must be able to filter the information that wants to be retrieved, for instance knowing only the uptime of a host this in a given virtual organization. Thereby, when carrying out a query for a metric of a given entity, user/application should have mechanisms for specify which information has to be gathered and which not. In our system, this can be done using the filters. Before requesting a metric, the filters have to be set up if needed.

Only the metrics' values the matching to the set filters will be retrieved, until the filters will be cleared. Setting the filters can be useful in some cases, for example when user always only wants to gather information about her/his finished jobs. However, in other cases, the filters can only make sense only for one query, for instance asking for those hosts that are alive. In this case, the filters can be provided directly to the GetMetricValue.

**AddMetricFilter:** Modify the current filter to restrict the metrics' values by selecting one of the parameters and the allowed values.

**Input:** the security object, the metric to be filtered and the filters to be applied. Source 5 shows an example of metric filter XML document.

**Output:** Ok if the metric has been set correctly, false otherwise.

**ClearMetricFilter:** Clear all previous filters for a given metric and entity. From the moment that this method is

called, no filters will be applied to the metric, when the user or application calls query it.

**Input:** the security object, the metric to be filtered and the filters to be applied. Source 5 shows an example of metric filter XML document.

**Output:** Ok if the metric has been set correctly.

```

<MetricFilter entity="job">
  <MetricData>
    <FilterByDate>
      <BetweenDates>
        <StartDate>1136208170544</StartDate>
        <EndDate>1136380970544</EndDate>
      </BetweenDates>
    </FilterByDate>
    <FilterByState>
      <State>FAILED</State>
    </FilterByState>
    <SubmissionDate>
      <BetweenDates>
        <StartDate>1138886570544</StartDate>
        <EndDate>1136380970544</EndDate>
      </BetweenDates>
    </SubmissionDate>
  </MetricData>
</MetricFilter>

```

**Source 5: Example of metric syntax for the entity job**

**GetMetricValue** Requests the current value of a metric. If filters are set to the current metric for the user/application they are applied.

**Input:** the security object, the name of the metric and optionally an XML document with the filters.

**Output:** the XML list of values of the metric everywhere the parameters match the current filters.

**Subscribe:** Requests for a periodic update of the metric.

**Input:** the security object, the name of the metric, the update frequency and the callback method or webservice.

**Output:** Ok if success, error code otherwise.

**UnSubscribe:** Cancels a subscription made with the Subscribe command.

**Input:** the security object, the name of the metric.

**Output:** Ok if success, error code otherwise.

**AddNotification:** Requests a notification on a specified event.

**Input:** the security object, a boolean asking whether it should be a singleshot notification, the name of the metric, the time and value thresholds.

**Output:** Ok if success, error code otherwise.

### D. Example of interaction with the Palantir system

Figure 4 and Figure 5 present two sequence diagrams that exemplify a possible interaction between all the components of the system and the final user:

1. Client starts a communication with the system providing its credentials.
2. Once authenticated it ask for the list of available entities in the system.
3. The access point (AP) gathers the list of the available entities to the Palantir Gateway and forward the list to the

client.

4. Let's suppose that it decides to query information about the *predictor* entity. It queries about all the *predictor* entity instantiation. Decides to what entity it wants to ask the information.
5. The AP gather the list of the available entities to the Gateway and forward the list to the client. Depending on internal parameters (such TTLs, or of the non-persistent nature of the entity) the Gateway may decide to ask to the modules that are linked to it about the instantiated entities, or it may decide to use its own data base.
6. Client queries for the metrics list that the *predictor* entity has, it decide which metrics it wants to ask, and it asks how the to query the chosen metrics. The AP gathers this list from the available Gateways and returns the list.
7. When it is chosen the entity instantiation, and the list of metrics that are required, it gathers all the metric data with the *GetMetricData* functionality.
8. The AP asks to the Gateway about the metric data for the given instantiation. The Gateway based on the key that identifies the entity chooses the appropriate module and gathers the required data. The AP returns the provided data by the Gateway to the client. Finally the communication is closed.

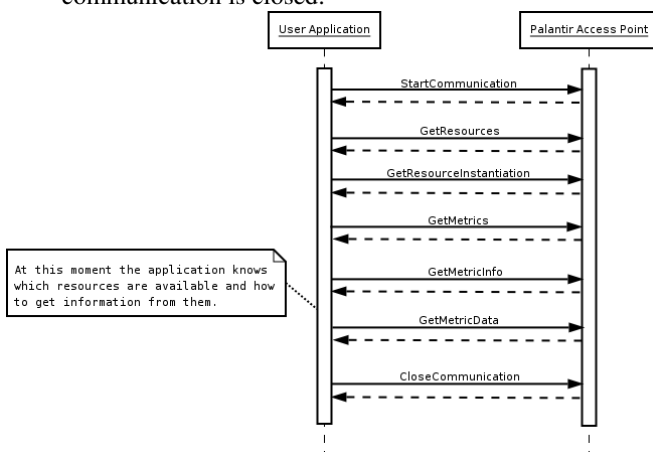


Figure 4: Interaction between client and Palantir

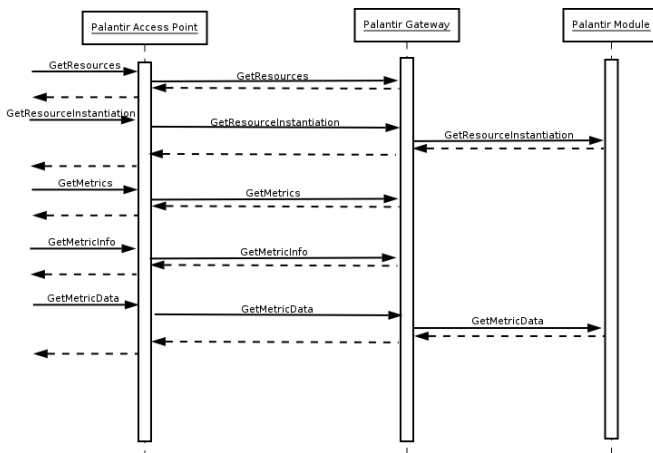


Figure 5: Interaction between the internal Palantir components

## VI. CONCLUSIONS

In this paper we have presented the Grid Palantir meta-information systems. We have described how it unifies the access to different monitoring and information systems and how its functionalities are provided and implemented. It has been shown how it abstracts the access to different data providers, and it has been demonstrated to be useful in situations where a very wide range of information is provided. The current state of the Palantir system has been also presented and how it has been integrated in the eNANOS architecture.

## REFERENCES

- [1] Ivan Rodero, Francesc Guim, Julita Corbalán and Jesus Labarta. eNANOS: Coordinated Scheduling in Grid Environments. Parco - Parallel Computing 2005
- [2] Francesc Guim Bernat, Ivan Rodero, Julita Corbalan, Jesus Labarta, Ariel Oleksak, Jarek Nabrzyski. Uniform Job Monitoring using the HPC-Europa Single Point of Access. International Workshop on Grid Testbeds 2006 - Singapur.
- [3] A. Oleksiak, A. Tulló, P. Graham, T. Kuczynski, J. Nabrzyski, D. Szejnfeld and T. Sloan. HPC-Europa: Towards Uniform Access to European HPC Infrastructures. To appear in Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing.
- [4] B. Tierney, R. Aydt, D. Gunter, W. Smith, V. Taylor, R. Wolski and M. Swany. A Grid monitoring architecture. Global Grid Forum 2002.
- [5] Zoltan Balaton and Fernc Vajda, Grid Information and Monitoring Systems 2004
- [6] K. Czajkowski, S. Fitzgerald, I. Foster, C. Kesselman. Grid Information Services for Distributed Resource Sharing. 10 th IEEE Symp. On High Performance Distributed Computing ,2001
- [7] Zoltan Balaton and Gabor Gombas. Resource and Job Monitoring in the Grid, Euro-Par 2003.
- [8] Rich Wolski, Neil T. Spring and Jim Hayes. The network weather service: a distributed resource performance forecasting service for metacomputing, Journal: Future Generation Computer Systems, Volume: 15, 1999
- [9] Bartosz Balis, Marian Bubak, Wlodzimierz Funika, Roland Wismuller, Marcin Radecki, Tomasz Szepieniec, Tomasz Arodz and Marcin Kurdziel. Performance Evaluation and Monitoring of Interactive Grid Applications. November 2004
- [10] Marian Bubak, Wlodzimierz Funika and Roland Wismüller. The CrossGrid Performance Analysis Tool for Interactive Grid Applications, year 2002
- [11] Matthew L. Massie, Brent N. Chun and David E. Culler. The ganglia distributed monitoring system: design, implementation, and experience, Parallel Computing 2004
- [12] I Foster, C Kesselman. Globus: A metacomputing infrastructure toolkit, J Intl - International Journal of Supercomputer Applications, 1997
- [13] J. Hodges and R. Morgan. RFC 3377: Lightweight Directory Access Protocol (v3): Technical Specification, 2002
- [14] I. Foster, D.Berry, A. Djaoi, A. Grimshaw, B.Horn, H. Kishimoto, F. Maciel, A. Savva, F. Siebenlist, R. Subramaniam, J. Treadwell and J. Von Reich. The open Grid services architecture version 1.0. Tech. rep., Global Grid Forum, 2004
- [15] The Globus Security Team. Globus Toolkit Version 4 Grid Security Infrastructure: A Standard Perspective. Version 2, 2004
- [16] GridLab site can be found at [www.gridlab.org](http://www.gridlab.org)
- [17] John Linn,RFC 1508: Generic security service application program interface, 1993
- [18] T. Ludwig, R. Wismüller, V. Sunderam and A. Bode. On-line Monitoring Interface Specification, Technische Universität München, 1997
- [19] Crossgrid site can be found at <http://www.crossgrid.org>
- [20] J Novotny, M Russell, O Wehrens. GridSphere: An Advanced Portal Framework. Proceedings of the 30th EUROMICRO Conference.