

TOWARDS UNIFORM AND TRANSPARENT ACCESS TO THE GRID INFORMATION USING THE PALANTIR*

Ivan Rodero, Francesc Guim and Julita Corbalan

Barcelona Supercomputing Center (BSC)

Technical University of Catalonia(UPC)

Jordi Girona 1-3, 08034 Barcelona, Spain

CoreGRID Institute on Resource Management and Scheduling

ivan.rodero@bsc.es

francesc.guim@bsc.es

julita.corbalan@bsc.es

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 or entities, with different nature, characteristics and so on. These issues make the task of gathering Grid information complex to treat, and it is difficult to provide a general way for accessing to all this information. In this paper 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. Moreover, we present real examples that state how Palantir has been integrated providing the uniform access to systems with heterogeneous information providers.

Keywords:

Grid Computing, Meta-Data Model, Monitoring Systems, Information Systems, Palantir

*This research work is carried out under the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265).

1. Introduction

As other Grid middleware components, monitoring services have to adapt to the characteristics of Grid architecture. They should provide uniform access to information and resources, ways to discover which capabilities it has and how to access it.

In the eNANOS [1] architecture we needed a new component that would allow accessing to all the information related to the entities (a Grid entity is any software or hardware component that is able to provide information) that are involved in our system, i.e: Grid jobs, local processes, resources and so on. This component should integrate and merge information from the bottom part of the Grid, coming from the local components of the centers (such as the eNANOS scheduler) with information coming from the top components of our architecture, in our case from the eNANOS Broker. The nature of the information that it would provide would be much diversified: information from the resource monitoring systems, from the job monitoring systems, performance predictions from predictors etc.

For achieve this goal, our first approach consisted on having a deep study of the available monitoring tools and try to adapt to our requirements the more appropriate one. However, at the best of our knowledge we realized that any of them matched all the requirements that we had. Using our experience in Grid monitoring and information systems obtained in the HPC-Europa project [2], we designed a kind of meta-information system that would implement all the needed functionalities.

The presented system is not intended to substitute any of the other existing monitoring or information tools. It is intended to uniform the access to all the information provided by them. Furthermore, as has been discussed in [2] and [3], most of the HPC centers have already deployed their own site-specific HPC and Grid infrastructure. Therefore, an additional requirement is to keep the autonomy of HPC centers allowing them to use their favorite Information Systems and Information providers.

The goal of this paper is to discuss the need of common and homogenized information protocols, information data models and information access functionalities in the Grid. We propose as a possible solution the Palantir meta-information, and present how the different requirements are provided in its design.

In the first part of the paper we present the motivations that let us to develop the Palantir system. In the second part we present the data model that has been designed for accessing to all the entities information, the architectural components that integrate the system, and finally a description of the main functionalities that will be provided to the Palantir users.

2. What do we need?

Usually, information systems (IS) do not use to provide all the information that administrator or users want. For example in [2] we stated that the Grid brokers are continuously providing new types of information, and the users and Grid components want this to be available as soon as possible. The IS should provide mechanisms for extend their functionalities and the set of information that they provide.

The Grid information providers (brokers, schedulers, monitoring systems etc.) frequently allow accessing to diversified information with different format and semantics. The IS that are providing access to such information should provide simple and generic APIs for query it. This property of generality is especially important due the information may have different semantic and structure, and new entities may appear in the future. Regarding this access there are two factors that must be taken in consideration:

- The methods that the API provides: they should provide mechanisms for discover the available entity types, its characteristics, the information they provide and the real entities that can provide this information; and mechanism for gather this information of each of this entities instantiations.
- The data model used for conveys the required information. The model should not be linked to a specific kind of entities/resources (such as physical resources like hosts or network, or software resources like applications or jobs).

Related to this issue of how to access the information, there is an important question that the IS should manage itself: where the information is stored?. In the current Grid architectures and systems the user has to know exactly where the information can be retrieved. For instance, if the user “fguim” wants to know the state for the job “grid123@pcmas”, he has to know that this information has to be retrieved queering the broker eNANOS that is running on the host “pcmas”. This problem may seem quite obvious to solve in some small architectures, however it can become a challenge for users in bigger systems.

Nowadays how to access the information is not only the unique key for the Grid consumers. Controlling to which information the users are accessing is also a mandatory goal when installing the IS to the real enterprises and Grid systems. Security is crucial issue when accessing to the resources information. They should take into account aspects as accounting, user privileges, communication security etc.

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 performance. Furthermore,

the coming Grids markets, like the models proposed by Dr. Buyya [4], needs this support for become a reality.

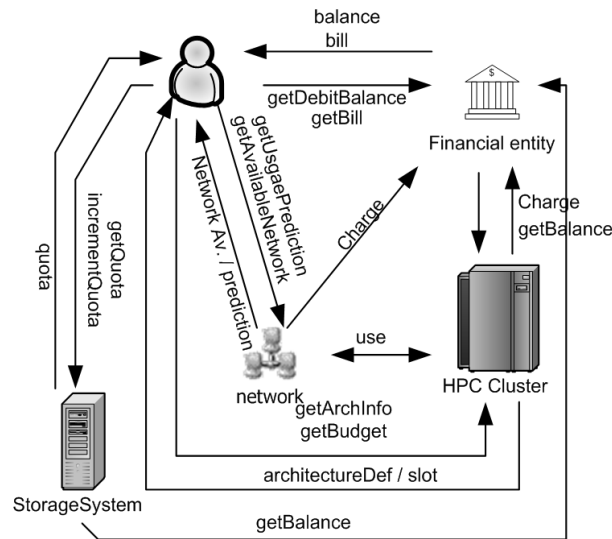


Figure 1. Information workflow in the Grid

The economic Grid is a clear example of an architecture that requires an information system with all the functionalities that we have discussed. Such Grids have to provide access information like bills, resource usage, users, banks accounts etc. It is clear that new entity types may appear and disappear continuously in such environments.

On the other hand each of these entities comes from a very different nature: the bank entities come from the economic namespaces, while the resource usage comes from the computer science domain. In such dynamic and diversified environments the IS must provide a very generic data model. This data model should be able to convey information like the users profile, thread/process/job performance etc.

The coming Grids for the 2010 are supposed to have hundred up to thousands of elements (entities). Each of them may have the role of information provider (IP). At this point, users/applications will not be able to know exactly where the information can be collected. Moreover, they will not know where the information is stored and who is providing it. IS will have to implement mechanism for discover exactly where the information that users are requiring is stored. For example: we can not expect a user will know that the resource consumption for the job “job123@pcmas” has to be queried to the IS “eNANOS@BSC”, but the consumption for the job “job124@kadesh” has to be queried to MDS.

Figure 1 exemplifies a situation where a unified mechanism for accessing the information it is required. There are four different components (a financial entity, a StorageSystem, an HPC-Resource and the user) that also have the role of IP. Each of this components may use different IS for publishing its information. When the components have to interact between them, they have to be aware of the format used for querying them, the format of the replies, its semantics etc. Clearly this situation becomes unsustainable when the number of information providers increase.

3. What can we do?

We carried out a deep study [5] about the more representative information systems available in the research area as a basis of our design: Globus Monitoring and Discovery Service (MDS) [6], GridLab Mercury [7], Network Weather Service (NWS) [8], CrossGrid OMIS Compliant Monitoring service for the Grid [9], and Ganglia [10].

We did not found any of them that exactly match all the requirements that have been discussed before. The main lack was the possibility of extending their functionalities and providing generic and uniform access to all their information.

However, all these IS provide very useful information. As each of these systems is mainly specialized in a specific domain, it is able to provide a high quality data about its namespace (job monitoring, resource monitoring etc.). At this point we the question was: why not unify the access to all these IS?

This question was the base for the Palantir meta-information system designs (see Figure 2). Its main goal is provide a uniform access to the whole Grid information providers.

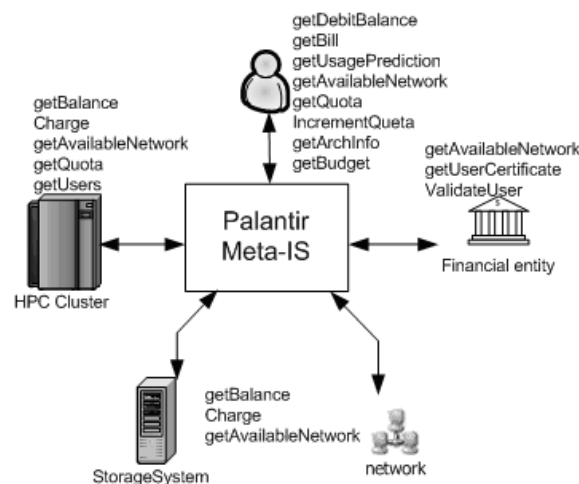


Figure 2. With a unified protocol

4. Palantir in the eNANOS system

Currently, we are mainly focused in providing access to all the components that are involved in the eNANOS architecture plus some other IS. This uniform information access will simplify notoriously the collection of data done by all the elements of our system. Figure 3 shows all the information providers that are being currently integrated as a part of the Palantir installation done in our system. Mainly there are four kinds of information providers:

- Resource monitoring IS. (Ganglia)
- Job Monitoring IS. Including both Grid and local job monitoring information (with information such as job / process / thread performance).
- Performance predictors. IS Including NWS and a set of predictor modules that have been designed by our research group.
- Service state information IS (MDS). They will provide information about the state of the services that used by the other components (applications etc.).
- Progress and performance indicators API. It will provide both absolute and relative information about the progress of the applications through a library and a run-time included inside the eNANOS framework [11].

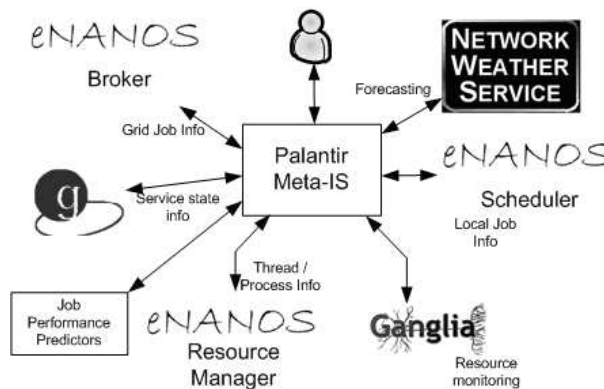


Figure 3. Palantir in the eNANOS architecture

5. The Palantir data model

The abstract Palantir data model is composed for two elements: the Entities and the Entity Metrics. The entities represent the conceptual elements of the systems that can contain information suitable to be requested. They are not

required to be physical resources. For example, hosts, jobs or applications are considered to be entities. Each entity has associated a set of metrics that contain specific information. 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. The attributes for the entity are (see Figure 4):

- Its name. That must be unique in the system (such as Job, predictor etc).
- Its description. That contains a human readable description of the entity.
- Its key. That, using the XML Schema technology, describes how this entity is identified. For example the key for the host entity may be composed by its hostname or by its IP address (or both).
- Its namespaces. It contains a set of URIs (Uniform Resource Identifier) identifying the semantic spaces to which the entity belongs.

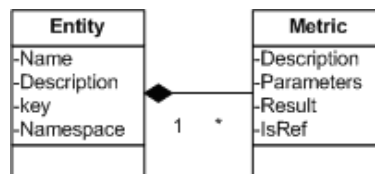


Figure 4. Abstract Data Model

An entity may have associated a set of metrics; each metric also has a set of predefined attributes:

- Its name. That must be unique to the entities to which it is associated.
- A human readable description of the information that it provides.
- Its parameters. Using also the XML Schema technology defines the parameters that can be provided to Palantir when requiring its content to an instantiation of a given entity. For example the entity Predictor requires the job id as a parameter when querying for its metric JobRunTimePrediction.
- The XML Schema that describes the format of information returned when querying its content. In case that the metric contains a reference to another entity instantiation this will indicated by the Boolean attribute IsRef.

The model presented until this point is the abstract model. However in the systems where Palantir is installed this data model is defined using concrete

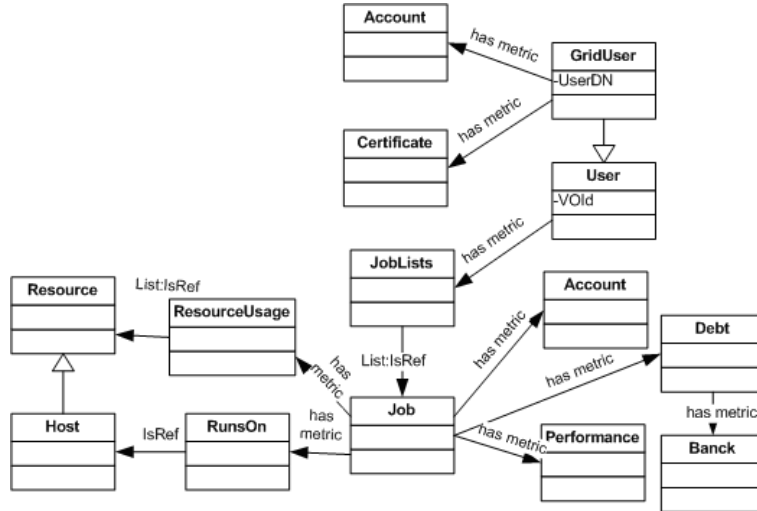


Figure 5. Palantir Data Model example

entities (not the instantiations). For example the Figure 5 presents a possible Palantir model that could be specified in a system where information about users and its jobs is provided.

Different Palantir systems may be installed in different domains. Each of these installations may have a different concrete model. The concrete model is mainly designed taking into account to which information systems the Palantir is providing access. When a new IS is added to an installation its concrete data model may vary. If this new IS provides new information that it is not included in any of the already defined entities the Palantir administrators can choose one of the three following options:

- Creating new entities that will provide information about the new conceptual elements.
- Extending the definition of an existing entity adding new metrics to it. For example, in case that the new IS provides performance information concerning the running jobs, the metric PerformanceIndicator could be added to the entity job.
- Extending the definition of an existing entity creating a sub entity. The proposed model is a UML like model, and entities may inherit the definition of other entities.

However, this analysis will not always be required when a Palantir will be installed in a particular architecture. There will be a set of predefined entities

whom design will be based on a set of IS (Ganglia, NWS etc.) that will available when building a concrete system. Furthermore, future versions may include semi/automatic methods for derive this actions.

6. The Palantir system architecture

In this section we present the overview of the architecture that allows gathering the information for each of the instantiation of the entities available in a concrete installation of the system.

Figure 6 provides a general view of the system architecture. As can be observed there are three top architectural components: at the top there are the Palantir Access Points that allow the uniform access to the users/applications to the system; the Palantir Gateways are the intermediate layers that control the access to the information providers installed in each of the centers joined to the system; and finally, the bottom components of the system are the information modules. They are the responsible of gathering the required information to the different IP/IS.

6.1 The access points

The first layer is integrated by the Palantir Access Points (AP). These components provide a uniform view to the end-user/application of the whole information that is available in the system. They must know which centers are available to provide information.

When the end-user/application carries out a query to the AP, it redirects it to the appropriate Palantir Gateway or Gateways that are able to provide the required information. An important question to address is how the AP knows to which GW have to connect when the end-user/applications requires information about a given entity. The system distinguishes between two kinds of entities: persistent entities and the temporal entities.

The persistent entities are non-volatile entities that will remain available in the system for a long time. Examples of this kind of entities are: host, cluster, performance predictor, storage system etc. For them the database stores among other data in which Gateway/Gateways its information is stored. For example in Figure 6 the Palantir AP will know that the entity "job21@pcmas" can be queried in the Gateway installed in the BSC Center.

The temporal entities are volatile entities or with a limited amount life time, for example: jobs. The AP identifies the GWs where this information can be gathered analyzing the entity key. If the key is in the composed form, a subset of this key must identify a persistent entity that will be used to find out in which GW the query has to be done. However, as not all the temporal entities will be able to satisfy the above property, a temporary entity can be also identified by

a direct links. This directly points to which module/s the information can be retrieved (such: “gw[id=‘X’]/module[id=‘MDS1’]/entity[id=‘job1’]”).

The management of this kind of keys should be transparent to the user. As will be presented in the following section, the system provides a set of discovering methods that allow to the clients to retrieve this kind of keys. In this cases the user does not have be aware of what the key means.

The protocol used from the application to this access point is based on the generic protocol presented in the following section. More important is that client does not have to be aware to which information system the final queries are done, abstracting it to the complexity of the underlying systems.

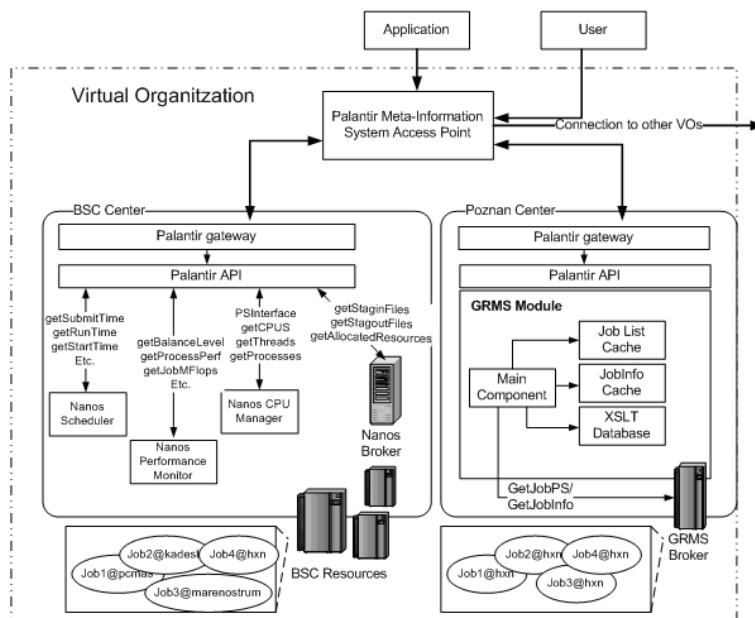


Figure 6. eNANOS and GRMS modules

6.2 The gateways

The main task of the gateway is choosing the appropriate module that will process a given query. It stores a data base with all the entities that are available on underlying modules. When entity information is required it searches in this database where it can be retrieved. The mechanism used for decide to which modules the information is gathered is exactly the same as the once presented for Access Point.

6.3 Information systems modules

Finally, at the bottom part of the architecture, the modules are responsible of carry out the final queries to the information or monitoring systems that they represent.

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.

6.4 An example: integrating two different job monitoring approaches

Figure 6 shows two different modules that have been developed for accessing the job monitoring information of two different systems [12]. The eNANOS module integrates job information coming from different components of our architecture [1]:

- The eNANOS scheduler provides general information about all the jobs that have been submitted on the local hosts.
- The eNANOS Performance Monitor provides information regarding the progress for the different processes that are running in each host: the achieved MFLOPs, the load balance level etc.
- The NANOS CPUManager provides information about how the different jobs/process/thread are behaving on the resources.
- The eNANOS Broker provides information about the jobs that have been submitted to the Grid: stagein files, stageout files, resources etc.

When the eNANOS module receives a query about a certain job metric it gathers the required information using the different APIs that the presented components provide (such as PSInterface of the CPU Manager, or the getBalanceLevel of the Performance monitor).

On the other hand, the same Palantir architecture allows monitoring the jobs that have been submitted to the GRMS broker [13]. The GRMS Palantir Module access to the GRMS monitoring information using the interface GetJobPS that returns a set of XML documents the monitoring information for a job that has been submitted to the broker. Using XSLT the module transforms these documents to the Palantir protocol format.

In this example, the end-user/application is able to accessing to the job monitoring information that comes from two different systems (with different mechanisms, different interfaces and different formats) using the standard format and protocol proposed in Palantir.

7. The Palantir protocol and interfaces

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. The second set of methods allows discovering which type of features (such as entities types, metrics and entities instantiations) and methods are available. And finally, the third type of methods allows gathering the metric values for the entities instantiations.

7.1 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: the StartCommunication opens a secure connection with the system; and the CloseCommunication closes the connection and invalidates the security objects.

```

1 <AvailableEntities xmlns:xs="http://www.w3.org/2001/XMLSchema">
2 <!-- List of resources availables -->
3 <Entity name="host">
4 <Description>This resource allows to carry out queries about the resource
   host</Description>
5 <key>
6 <xs:element name="host">
7 <xs:complexType>
8 <xs:choice>
9 <xs:element name="ip" type="xs:string"/>
10 <xs:element name="hostname" type="xs:string" minOccurs="0"/>
11 </xs:choice>
12 </xs:complexType>
13 </xs:element>
14 </key>
15 </Entity >
16 <Entity name="application">
17 <Description>This resource allows to carry out queires about the software
   resource application</Description>
18 <key>
19 <xs:element name="application">
20 <xs:complexType>
21 <xs:choice>
22 <xs:element name="name" type="xs:string"/>
23 <xs:element name="version" type="xs:string" minOccurs="0"/>
24 </xs:choice>
25 </xs:complexType>
26 </xs:element>
27 </key>
28 </Entity >
29 </AvailableEntities>

```

Source 1: GetResourceInstantiation

7.2 Discovering the available features

The meta-information system, as it is providing a wide range of information, and it is 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.

- **GetEntitiesTypes:** Returns the list of the types of entities available on the system. An example of a returned XML is shown in the Source 1. User can filter the information to be retrieved, for example only entities types that concerns computational resources and forecasting entities.
- **GetEntityInstantiation:** Returns the list of instances of a provided entity type. For instance, user may know the list of entities of type "host". As the GetEntitiesTypes method, the Palantir client can specify some parameters for filter the instantiations to be retrieved.
- **GetMetricInfo:** Returns information about a particular metric of a particular entity or resource. For instance, calling getMetricInfo("application", "prediction_job_memory_usage") we could obtain the XML document presented in Source 2.

```

1 <Metric>
2   <Name>prediction_job_memory_usage</Name>
3   <Description>Returns a prediction of the memory that a given application will use
4     if executed.
5   </Description>
6   <Parameters>
7     <xs:element name="Parameters">
8       <xs:complexType>
9         <xs:sequence>
10          <xs:element name="AppName"/>
11          <xs:element name="Host"/>
12          <xs:element name="User"/>
13          <!-- ETCETERA -->
14        </xs:sequence>
15      </xs:complexType>
16    </xs:element>
17  </Parameters>
18  <!-- ETCETERA -->
19 </Filters>
20 <Notifications>
21   <Periodic available="no"/>
22   <Punctual available="yes"/>
23 </Notifications>
24 </Metric>

```

Source 2: Sample of output for the GetMetricInfo

7.3 Getting the entity information

The client can retrieve the metrics values of a given entity instantiation using the GetMetricValue functionality. The input for this method is basically a set of entity instantiations identifiers plus all the metrics that the client wants to know about each of them. Each metric can be parameterized. For instance the metric JobList for the entity Scheduler can be parameterized as shown in Source 3.

```

1 <Parameters>
2   <JobsMatching >
3     <FilterByDate>
4       <BetweenDates>
5         <StartDate>1136208170544</StartDate>
6         <EndDate>1136380970544</EndDate>
7       </BetweenDates>
8     </FilterByDate>
9     <FilterByState>
10      <State>FAILED</State>
11      <State>SUSPENDED</State>
12    </FilterByState>
13    <SubmissionDate>
14      <BetweenDates>
15        <StartDate>1138886570544</StartDate>
16        <EndDate>1136380970544</EndDate>
17      </BetweenDates>
18    </SubmissionDate>
19  </JobsMatching>
20 </Parameters>

```

Source 3: Parameters for the metric JobLists

8. Conclusions

In this paper we have discussed the need of a new Grid component that has to provide a uniform access to the whole Grid information. The current Grid architectures are composed by different types of information systems that provide: different access methodology, different format information and semantics. We can not expect users neither applications to know exactly how each of this IP has to be queried. This situation result in unsustainable and non maintainable Grid architectures, where all the information consumers are highly dependant to the changes that the different IP may have. Furthermore, adding new information systems can result in important source redefinitions in the already deployed consumers or, in the worst, cases in a redesign of their internals. This problem will become dramatic if the number of Grid IP and consumer components increases as it is expected.

We also have presented the problematic providing several examples where this component would help to simplify the overall Grid infrastructure and the relations among the different information producers/consumers, and it would make the system more extensible.

As a solution proposal, 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. A real use cases of how this system has been integrated successfully in some architectures have been described.

References

- [1] I. Rodero, F. Guim, J. Corbalan, J. Labarta, "eNANOS: Coordinated Scheduling in Grid Environments", *Parallel Computing: Current & Future Issues of High-End Computing*, G.R. Joubert et al. (Eds.), *Parallel Computing (ParCo)*, pp. 81-88, Malaga, Spain, 13-16 September, 2005.
- [2] F. Guim, I. Rodero, J. Corbalan, J. Labarta, A. Oleksiak, J. Nabrzyski, "Uniform job monitoring using the hpc-europa single point of access", *International Workshop on Grid Testbeds*, in conjunction with CCGrid2006, Singapore, 16-19 May, 2006.
- [3] A. Oleksiak, A. Tullo, P. Graham, T. Kuczynski, J. Nabrzyski, D. Szejnfeld, T. Sloan, "HPC-Europa: Towards Uniform Access to European HPC Infrastructures", 6th *IEEE/ACM International Workshop on Grid Computing Grid2005*, Seattle, USA, 2005.
- [4] R. Buyya, "Economic-based Distributed Resource Management and Scheduling for Grid Computing", Ph.D. Thesis, Monash University, Melbourne, Australia, April 12, 2002.
- [5] O. Levillain, F. Guim, I. Rodero, J. Corbalan, J. Labarta, "Comparison of several grid monitoring tools", *Technical Report UPC-DAC-RR-CAP-2006-22*, Computer Architecture Department, Technical University of Catalonia (UPC), Barcelona, Spain, 2006.
- [6] K. Czajkowski, S. Fitzgerald, I. Foster, C. Kesselman, "Grid Information Services for Distributed Resource Sharing", *Symp. On High Performance Distributed Computing*, 2001.
- [7] Z. Balaton, G. Gombas, "Resource and Job Monitoring in the Grid", *Euro-Par 2003*.
- [8] R. Wolski, N. T. Spring, J. Hayes, "The network weather service: a distributed resource performance forecasting service for metacomputing", *Journal of Future Generation Computer Systems*, 1999.
- [9] B. Balis, M. Bubak, W. Funika, R. Wismuller, M. Radecki, T. Szepieniec, T. Arodz, M. Kurdziel, "Performance Evaluation and Monitoring of Interactive Grid Applications", *LNCS 3241*, November 2004.
- [10] M. L. Massie, B. N. Chun, D. E. Culler, "The ganglia distributed monitoring system: design, implementation, and experience", *Parallel Computing (ParCo)*, 2003.
- [11] I. Rodero, F. Guim, J. Corbalan, J. Labarta, "Design and Implementation of a General-Purpose API of Progress and Performance Indicators", *Parallel Computing (ParCo) 2007*.
- [12] F. Guim, I. Rodero, J. Corbalan, J. Labarta, A. Oleksiak, K. Kurowski, J. Nabrzyski, "Integrating the Palantir Grid Meta-Information System with GRMS", S. Gorlatch et al. (Eds.): *Integrated Research in Grid Computing, CoreGRID Integration Workshop*, pp. 49-60, Krakow, Poland, 19-20 October, 2006.
- [13] K. Kurowski, J. Nabrzyski, J. Oleksiak, "Programming Grid Applications with Gridge", *Computational Methods for Science and Technology - OWN 2006*.