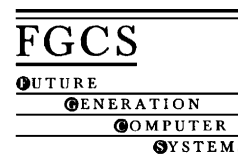




ELSEVIER

Future Generation Computer Systems 18 (2002) 1101–1112



www.elsevier.com/locate/future

# Distributed data mining on the grid

Mario Cannataro<sup>a</sup>, Domenico Talia<sup>a,b,\*</sup>, Paolo Trunfio<sup>a,b</sup>

<sup>a</sup> ICAR-CNR, Via P. Bucci, Cubo 41-C, 87036 Rende (CS), Italy

<sup>b</sup> DEIS, Università della Calabria, Via P. Bucci, Cubo 41-C, 87036 Rende (CS), Italy

## Abstract

In many industrial, scientific and commercial applications, it is often necessary to analyze large data sets, maintained over geographically distributed sites, by using the computational power of distributed and parallel systems. The grid can play a significant role in providing an effective computational support for knowledge discovery applications. We describe a software architecture for geographically distributed high-performance knowledge discovery applications called *KNOWLEDGE GRID*, which is designed on top of computational grid mechanisms, provided by grid environments such as Globus. The *KNOWLEDGE GRID* uses the basic grid services such as communication, authentication, information, and resource management to build more specific parallel and distributed knowledge discovery tools and services. The paper discusses how the *KNOWLEDGE GRID* can be used to implement distributed data mining services.

© 2002 Elsevier Science B.V. All rights reserved.

**Keywords:** Knowledge discovery; Distributed data mining; Grid services

## 1. Introduction

In many scientific and business areas, massive data collections of terabyte and petabyte scale need to be used and analyzed. These huge amounts of data represent a critical resource in several application areas. Moreover, in many cases these data sets must be shared by large communities of users that pool their resources from different sites of a single organization or from a large number of institutions.

Grids are geographically distributed platforms for computation, composed of a set of heterogeneous machines accessible to their users via a single interface. Grid computing has been proposed as an important computational model, distinguished from

conventional distributed computing by its focus on large-scale resource sharing, innovative applications, and, in some cases, high-performance orientation. The main original application area was advanced science and engineering. Recently, grid computing is emerging as an effective paradigm for coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations operating in the industry and business arena [1]. Thus, today grids can be used as effective infrastructures for distributed high-performance computing and data processing.

Together with the grid shift towards industry and business applications, a parallel shift toward the implementation of data grids has been registered. Data grids are designed in order to allow large data sets to be stored in repositories and moved about with the same ease that small files can be moved. They represent an enhancement of computational grids, driven by the need to handle large data sets without constant, repeated authentication, aiming to support the implementation

\* Corresponding author. Present address: DEIS, Università della Calabria, Via P. Bucci, Cubo 41-C, 87036 Rende (CS), Italy.

Tel.: +39-09-8449-4726; fax: +39-09-8483-9054.

E-mail addresses: cannataro@isi.cs.cnr.it (M. Cannataro),

talia@deis.unical.it (D. Talia), trunfio@si.deis.unical.it (P. Trunfio).

of distributed data-intensive applications. Data grids seem to be present largely motivated by the data handling needs of next-generation particle accelerators. Significant examples are the EU data grid [2], the particle physics data grid [3] and the Japanese grid data farm [4] projects. In particular, the Globus data grid project [5] is currently engaged in defining and developing a persistent data grid middleware that offers:

- A high-performance, secure, robust data transfer mechanism.
- A set of tools for creating and manipulating replicas of large data sets.
- A mechanism for maintaining a catalog of data set replicas.

Data grid middleware is crucial for the management of data on grids, however in many scientific and business areas is also vital to have tools and environments that support the process of analysis, inference and discovery over the available data. These environments support scientists and engineers in the implementation and use of grid-based problem solving environments (PSEs) for modeling, simulation and analysis of scientific experiments. The same occurs for executives that ask analysts to mine very large distributed data sets to discover corporate knowledge to be used in decision making.

On the basis of these motivations, the evolution of data grids are represented by knowledge grids that offer high level tools and techniques for the distributed mining and extraction of knowledge from data repositories available on the grid. The development of such an infrastructure is the main goal of our research activities that are focused on the design and implementation of an environment for geographically distributed high-performance knowledge discovery applications called *KNOWLEDGE GRID*. The *KNOWLEDGE GRID* can be used to perform data mining on very large data sets available over grids, to make scientific discoveries, improve industrial processes and organization models, and uncover business valuable information.

The outline of the paper is as follows. [Section 2](#) describes the *KNOWLEDGE GRID* general architecture and the features of the main components. [Section 3](#) discusses a meta-learning application on grids developed on the *KNOWLEDGE GRID*. [Section 4](#) outlines the implementation status of the system and [Section 5](#) concludes the paper.

## 2. The *KNOWLEDGE GRID* architecture

The *KNOWLEDGE GRID* architecture is defined on top of grid toolkits and services, i.e. it uses basic grid services to build specific knowledge extraction services. Following the *Integrated Grid Architecture* approach [6], these services can be developed in different ways using the available grid toolkits and services. The current implementation is based on the Globus toolkit [7]. As in Globus, the *KNOWLEDGE GRID* offers global services based on the cooperation and combination of local services. We designed the *KNOWLEDGE GRID* architecture so that more specialized data mining tools are compatible with lower-level grid mechanisms and also with the *Data Grid* services. This approach benefits from “standard” grid services that are more and more utilized and offers an open parallel and distributed knowledge discovery architecture that can be configured on top of grid middleware in a simple way.

### 2.1. *KNOWLEDGE GRID* services

The *KNOWLEDGE GRID* services are organized in two hierarchic levels:

- the Core K-grid layer;
- the High level K-grid layer.

The former refers to services directly implemented on the top of generic grid services, the latter is used to describe, develop and execute distributed knowledge discovery computations over the *KNOWLEDGE GRID*. The *KNOWLEDGE GRID* layers are depicted in [Fig. 1](#). The figure shows layers as implemented on the top of Globus services; moreover, the *KNOWLEDGE GRID* data and metadata repositories are also shown. In the following the term *K-grid node* will denote a Globus node implementing the *KNOWLEDGE GRID* services.

#### 2.1.1. Core K-grid layer

The Core K-grid layer offers the basic services for the definition, composition and execution of a distributed knowledge discovery computation over the grid. Its main goals are the management of all metadata describing features of data sources, third party data mining tools, data management, and data visualization tools and algorithms. Moreover, this layer coordinates the application execution by attempting to fulfill

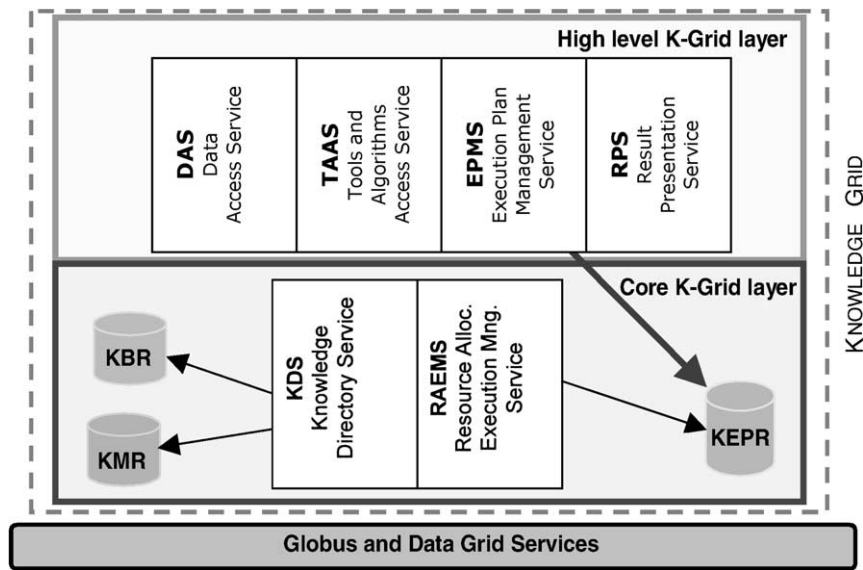


Fig. 1. The KNOWLEDGE GRID architecture.

the application requirements and the available grid resources. This layer comprises two main services.

**2.1.1.1. KNOWLEDGE directory service (KDS).** This service extends the basic Globus MDS service and it is responsible for maintaining a description of all the data and tools used in the KNOWLEDGE GRID. The metadata managed by the KDS regard the following kind of objects:

- Repositories of data to be mined, such as databases, plain files, eXtensible Markup Language (XML) documents and other structured or unstructured data (data sources).
- Tools and algorithms used to extract, filter and manipulate data (data management tools).
- Tools and algorithms used to analyze (mine) data, that is data analysis tools available on the grid.
- Tools and algorithms used to visualize, store and manipulate mining results, that is data visualization tools.
- Distributed knowledge discovery execution plans. An execution plan is an abstract description of a KNOWLEDGE GRID application, that is a graph describing the interaction and data flow among data sources, DM tools, visualization tools, and result storing.

- Knowledge obtained as a result of the mining process, i.e. learned models and discovered patterns.

The metadata information are represented by XML documents and are stored in a *Knowledge Metadata Repository* (KMR). They describe features of different data sources that can be mined, as location, format, availability, available views and level of aggregation of data.

Whereas it could be infeasible to maintain the data to be mined in an ad hoc repository, it could be useful to maintain a repository of the discovered knowledge. These information (see below) is stored in a *Knowledge Base Repository* (KBR), but the metadata describing them are managed by the KDS. The KDS is then used not only to search and access raw data, but also to find previously discovered knowledge that can be used to compare the output of a given mining computation when varying data, or to apply data mining tools in an incremental way.

Data management, analysis and visualization tools are usually pre-existent to the KNOWLEDGE GRID (i.e. they reside over file systems or code libraries). However, to make them available to knowledge discovery computations, relevant metadata have to be stored in the KMR. At the same time, metadata are vital to allow the use of data sources. Another important

repository is the *Knowledge Execution Plan Repository* (KEPR) storing the execution plans of data mining processes.

**2.1.1.2. Resource allocation and execution management service (RAEMS).** These services are used to find the best mapping between an execution plan and available resources, with the goal of satisfying the application requirements (computing power, storage, memory, database, network bandwidth and latency) and grid constraints. The mapping has to be effectively obtained (co-)allocating resources. After the execution plan activation, this layer manages and coordinates the application execution. Other than using the KDS and the Globus MDS services, this layer is directly based on the Globus GRAM services. Resource requests of each single data mining program are expressed using the *Resource Specification Language* (RSL). The analysis and processing of the execution plan will generate global resource requests that in turn are translated into local RSL requests for local GRAMs.

### 2.1.2. High level K-grid layer

The High level K-grid layer includes services used to compose, validate, and execute a parallel and distributed knowledge discovery computation. Moreover, the layer offers services to store and analyze the discovered knowledge. Main services are as follows.

**2.1.2.1. Data access service (DAS).** The DAS is responsible for the search, selection (data search services), extraction, transformation and delivery (data extraction services) of data to be mined. The search and selection services are based on the core KDS service. On the basis of the user requirements and constraints, the DAS automates (or assists the user in) the searching and finding of data sources to be analyzed by the DM tools.

The extraction, transformation and delivery of data to be mined are based on the Globus GASS services and also use the KDS. When useful data are found, data mining tools can require some transformation, whereas the user requirements or security constraints can require some data filtering before extraction. These operations can be usually done after the DM tools selection. The extraction functions can be embedded in the data mining programs or, more use-

fully, can be coded and stored in a utility repository, accessible by the KDS.

**2.1.2.2. Tools and algorithms access service (TAAS).** This service is responsible for the search, selection, downloading of data mining tools and algorithms. As before, the metadata regarding their availability, location, and configuration are stored in the KMR and managed by the KDS, whereas the tools and algorithms are stored in the local storage facility of each K-grid node. A node wishing to “export” data mining tools to other users has to “publish” them using the KDS services, which store the metadata in the local portion of the KMR. Some relevant metadata are parameters, format of input/output data, type of data mining algorithm implemented, resource requirements and constraints, and so on.

**2.1.2.3. Execution plan management service (EPMS).** An execution plan is represented by a graph describing the interaction and data flows between data sources, extraction tools, DM tools, visualization tools, and storing of knowledge results in the KBR. In simplest cases, a user can directly describe the execution plan, using a visual composition tool where the programs are connected to the data sources. However, due to the variety of results produced by the DAS and TAAS, different execution plans can be produced, in terms of data and tools location, strategies to move or stage intermediate results and so on. Thus, the EPMS is a semi-automatic tool that takes data and programs selected by the user, and generates a set of different, possible execution plans that meet user, data and algorithms requirements and constraints.

Execution plans are stored in the KEPR to allow for the implementation of iterative knowledge discovery processes, e.g. periodical analysis of the same data sources that vary during time. More simply, the same execution plan can be used to analyze different sets of data. Furthermore, different execution plans can be used to analyze in parallel the same set of data, and to compare the results using different point of views (e.g., performance, accuracy).

**2.1.2.4. Results presentation service (RPS).** Result visualization is a significant step in the data mining process that can help users in the model interpretation. This service specifies how to generate, present

and visualize the knowledge models extracted (e.g., association rules, clustering models, classifications), and offers the API to store them in different formats in the KBR. The result metadata are stored in the KMR to be managed by the KDS. The KDS is used not only to search and access raw data, but also to find pre-discovered knowledge that can be used to compare the output of a given KNOWLEDGE GRID computation with different data sets, or to apply data mining tools in an incremental way.

### 3. Distributed meta-learning on grids

After the general description of the KNOWLEDGE GRID architecture, here we describe how it can be exploited through a practical example of distributed meta-learning process.

Several classes of knowledge discovery applications over the grid can be taken into account, both in the parallel data mining (PDM) and distributed data mining (DDM) domains. PDM applications could get a great improvement from a computational grid setting, scaling up the speed of the DM process over grid nodes. On the other hand, DDM systems, offering techniques for the analysis of data that are stored on remote sites of a computer network, could mainly benefit of grid environments both for accessing distributed domain-specific sources of data and by distributing the KDD process over the computational grid resources.

#### 3.1. A meta-learning scenario

In the following we discuss a simple example of meta-learning process over the KNOWLEDGE GRID. It is not representative of every possible scenarios, however it can be useful to show how the execution of a significant DDM application can benefit from the KNOWLEDGE GRID services. Meta-learning aims to generate a number of independent classifiers by applying learning programs to a collection of distributed data sets in parallel. The classifiers computed by learning programs are then collected and combined to obtain a global classifier [8].

Fig. 2 shows a distributed meta-learning scenario, in which a global classifier  $GC$  is obtained on  $NodeZ$  starting from the original data set  $DS$  stored on  $NodeA$ . This process can be described as follows:

- Step 1: on  $NodeA$ , training sets  $TR_1, \dots, TR_n$ , testing set  $TS$  and validation set  $VS$  are extracted from  $DS$  by the partitioner  $P$ . Then  $TR_1, \dots, TR_n$ ,  $TS$  and  $VS$  are respectively moved from  $NodeA$  to  $Node_1, \dots, Node_n$ , and to  $NodeZ$ .
- Step 2: on each  $Node_i$  ( $i = 1, \dots, n$ ) the classifier  $C_i$  is trained from  $TR_i$  by the learner  $L_i$ . Then each  $C_i$  is moved from  $Node_i$  to  $NodeZ$ .
- Step 3: on  $NodeZ$ , the  $C_1, \dots, C_n$  classifiers are combined and tested on  $TS$  and validated on  $VS$  by the combiner/tester  $CT$  to produce the global classifier  $GC$ .

If data of interest are already distributed over different nodes, step 1 is omitted and the meta-learning process starts from step 2, in which  $TR_i$  are the portions of data stored on various  $Node_i$ , and  $VS$  and  $TS$  are extracted from  $TS_i$  across distributed nodes.

The meta-learning scenario described above could be applied to the following example. Let us assume that:

- (i) the data set  $DS$  is stored on  $NodeA$  ( $NodeA$  is a K-grid node);
- (ii) the necessary tools for meta-learning process, that is the partitioner  $P$ , the learner  $L$  (we assume  $L_1 = L_2 = \dots = L_n = L$ ) and the combiner/tester  $CT$ , are available as multi-platform executables on a  $NodeS$  ( $NodeS$  is a K-grid node).

We suppose that a grid user (GU) needs to perform the classification of data in  $DS$  by means of meta-learning tools available on  $NodeS$ . This task could be performed over the KNOWLEDGE GRID as described in Fig. 3:

- Step 1: the GU starts the search of computational resources for executing the meta-learning task from his/her K-grid node ( $NodeU$ ). The search, performed by means of the KDS, locates the computational resources needed to execute the learning process, respectively the  $HC_1$  and  $HC_2$  Globus nodes (i.e. homogeneous clusters which provide the  $n$  computing elements on which learning processes will be performed in parallel), and the  $NodeZ$  Globus node, on which the combining process will be executed. The search process compares the meta-information about the  $L$  and  $CT$

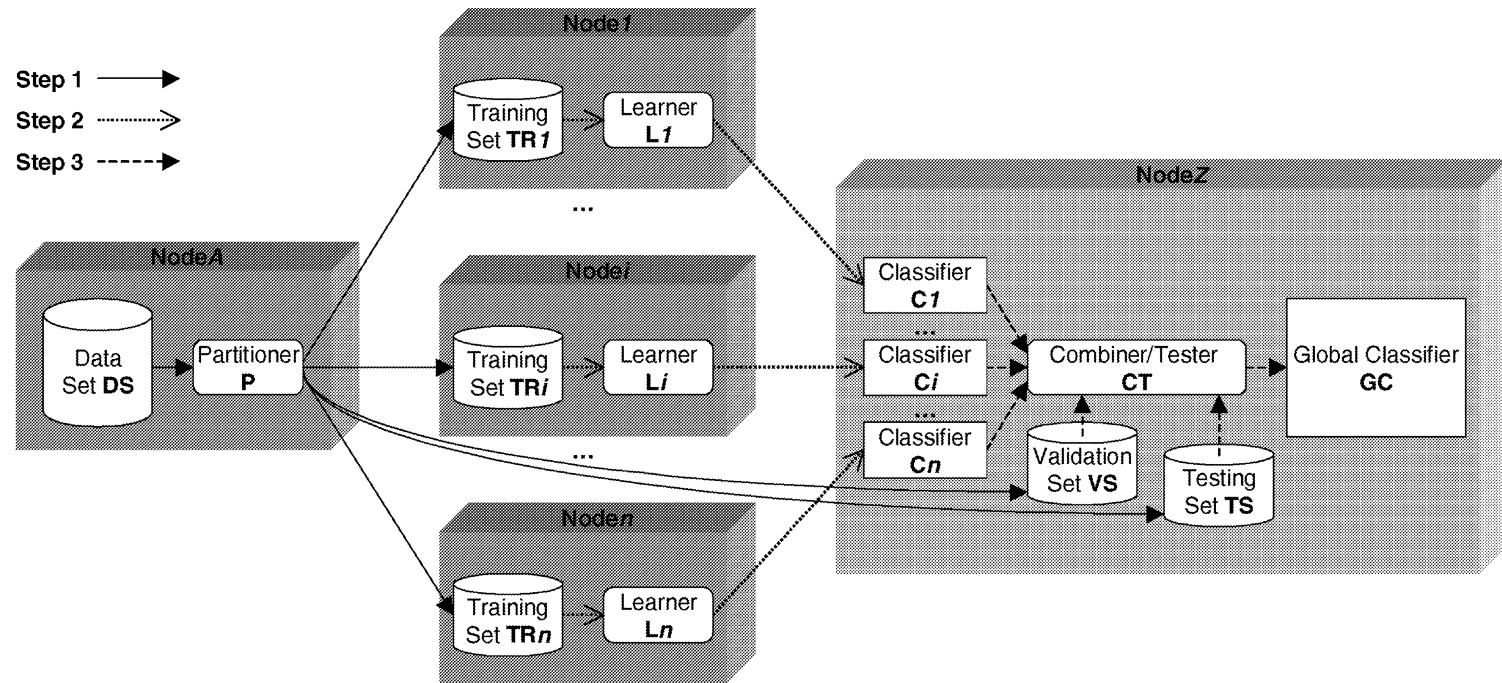


Fig. 2. A distributed meta-learning scenario.

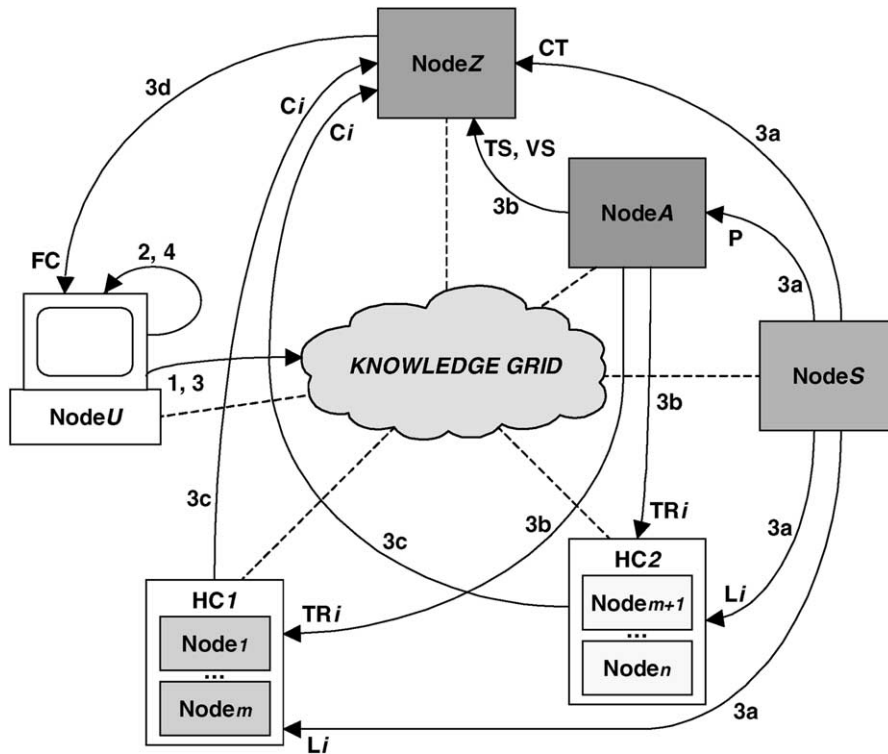


Fig. 3. Meta-learning task over the grid.

programs (i.e., required memory, libraries, etc.) against the features of the *HCS* and *NodeZ* nodes.

Step 2: the GU builds an execution plan for the meta-learning task, specifying strategies for tools and data movement, and for algorithm execution. The execution plan is built by using the EPMS and is stored into the local KEPR.

Step 3: the GU starts the application by submitting the execution plan to the RAEMS. The following steps are then executed:

- a. From *NodeS*, *P* is staged on *NodeA*, *L* (as  $L_i$ ) is staged on each cluster node, and *CT* is staged on *NodeZ*. The staging process (i.e. program movement and eventual installation on a target node) is executed by using the TAAS service on *NodeS*.
- b. On *NodeA*, *P* extracts data subsets  $TR_1, \dots, TR_n$ , *TS* and *VS* and move them,

respectively, to the computing elements of the two clusters *HC1* and *HC2*, and to the *NodeZ*. These data extraction and movement operations are executed by using the DAS service on *NodeA*.

- c. On each computing element of the cluster nodes *HC1* and *HC2*, the learning programs  $L_i$  generate the partial classifiers  $C_i$ , which are then moved on *NodeZ*. The  $C_i$  are partial results of the data mining process, so they are moved to the *NodeZ* by means of the DAS service.
- d. On *NodeZ* the combining and testing processes performed by *CT* generate the global classifier *GC*, which is moved to the KBR of the *NodeU*. This task is executed by the RPS service on *NodeU*.

Step 4: the GU visualizes and evaluates the result of computation (the *GC* stored in the local KBR) by means of the RPS tools.

### 3.2. Task building and execution process

The task building and execution process is shown in Fig. 4.

The *Task Composer* (TC), an internal EPMS module, assists the user in building the execution plan. It presents to the user a set of graphic objects representing resources (e.g., datasets, data mining tools, computational nodes), which metadata were previously stored into the *Task Resource Cache* (TRC). These metadata, containing information about resources selected to perform the computation, are extracted

from local or remote KMRs as result of KDS queries. Therefore, the *TC* allows the user to compose these objects using common visual facilities (e.g., drag and drop), to form a graphic representation of the application data flow. That composition is submitted to the *Execution Plan Generator* (EPG), another internal EPMS module, which validates it by verifying its logical consistency, and hence generates its XML representation, that is the execution plan of computation.

As an example, the execution plan in Fig. 5 describes the KDD computation at a high level, not containing physical information about resources (which

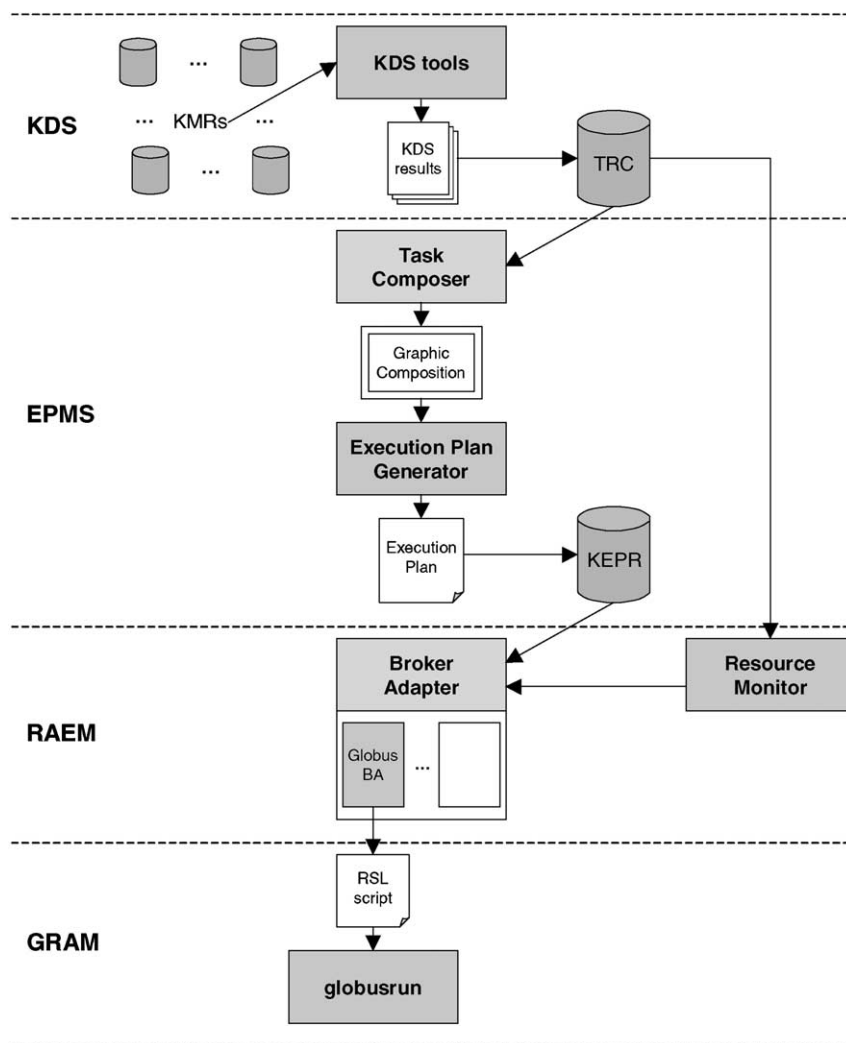


Fig. 4. Task building and execution process.



```

<ExecutionPlan>
...
<Task ep:label="TR11">
  <DataTransfer>
    <Source      ep:href="TR1_A.xml" ep:title="TrainingSet1 on NodeA"/>
    <Destination ep:href="TR1_1.xml" ep:title="TrainingSet1 on Node1"/>
  </DataTransfer>
</Task>
<Task label="L11">
  <Computation>
    <Program ep:href="L1_1.xml" ep:title="Learner1 on Node1"/>
    <Input   ep:href="TR1_1.xml" ep:title="TrainingSet1 on Node1"/>
    <Output  ep:href="C1_1.xml" ep:title="Classifier1 on Node1"/>
  </Computation>
</Task>
<Task ep:label="C1Z">
  <DataTransfer>
    <Source      ep:href="C1_1.xml" ep:title="Classifier1 on Node1"/>
    <Destination ep:href="C1_Z.xml" ep:title="Classifier1 on NodeZ"/>
  </DataTransfer>
</Task>
...
<Task ep:label="CTZ">
  <Computation>
    <Program ep:href="CT_Z.xml" ep:title="CombinerTester on NodeZ"/>
    <Input   ep:href="TS_Z.xml" ep:title="TestingSet on NodeZ"/>
    <Input   ep:href="VS_Z.xml" ep:title="ValidationSet on NodeZ"/>
    <Input   ep:href="C1_Z.xml" ep:title="Classifier1 on NodeZ"/>
    ...
    <Output  ep:href="GC_Z.xml" ep:title="GlobalClassifier on NodeZ"/>
  </Computation>
</Task>
...
<TaskLink ep:from="TR11" ep:to="L11"/>
<TaskLink ep:from="C1Z" ep:to="CTZ"/>
...
</ExecutionPlan>

```

Fig. 5. A fragment of the execution plan for the meta-learning example.

are identified by metadata references), nor about status and current availability of such resources.

In fact, specific information about the involved resources will be included in the next phase, when the execution plan is translated into the particular broker language. Fig. 5 shows a fragment of the execution plan for the meta-learning example described above. The execution plan gives a list of tasks and task links, which are specified using respectively the XML tags `Task` and `TaskLink`. The `label` attribute for `Task` element identifies each basic task

in the execution plan, and is used in linking various basic tasks to form the overall task flow. Each `Task` element contains a task-specific sub-element, which indicates the parameters of the particular represented task. For instance, the task identified by the `TR11` label contains a `DataTransfer` element, indicating that it is a data transfer task. The `DataTransfer` element specifies `Source` and `Destination` of the data transfer. The `href` attributes of such elements specify the location of metadata about source and destination objects. In this example, metadata

about source of data transfer in the  $TR_{11}$  task are provided by the  $TR_{1\_A}.xml$  file, whereas metadata about destination are provided by the  $TR_{1\_1}.xml$  file. The  $TR_{1\_A}.xml$  document provides metadata about the training set  $TR_1$  when stored on  $Node_A$ , whereas the  $TR_{1\_1}.xml$  document provides metadata about  $TR_1$  when stored, after data transfer, on  $Node_1$ . Both  $TR_{1\_A}.xml$  and  $TR_{1\_1}.xml$  files are stored in the local TRC. The  $TaskLink$  elements represent the relations among the tasks of execution plan. For instance, the first  $TaskLink$  specified in Fig. 5 indicates that the task flow proceeds from the task  $TR_{11}$  to the task  $L_{11}$ , as specified by its *from* and *to* attributes.

The *Broker Adapter* (BA) (see Fig. 4) maps the generic execution plan, represented by an XML document, into a specific Globus RSL script, which can be directly executed by means of the *globusrun* program. The mapping process performed by the BA is based on the *Resource Monitor* (RM) tool, which provides physical information of resources referenced in the execution plan. Static information (e.g., CPU speed, memory size, etc.) is provided by the RM directly accessing metadata stored in the TRC (a local cache in the KDS), while dynamic information (e.g., current CPU load and network latency) is provided by means of specific testing tools, for instance based on standard Unix or TCP/IP utilities.

#### 4. KNOWLEDGE GRID implementation

As discussed in Section 2, the *KNOWLEDGE GRID* architecture is composed of two hierarchic layers: the *Core K-grid* and the *High level K-grid* layers. To deploy *K-grid applications* we are implementing the former layer, on top of the Globus services. In particular, we are currently implementing the KDS, the RAEMS, and some basic tools allowing a user for publishing metadata about the *KNOWLEDGE GRID* objects (data sources and data mining algorithms). Moreover, we are implementing the basic *High level K-grid* tools useful to compose and start a *KNOWLEDGE GRID* application.

##### 4.1. *KNOWLEDGE GRID* metadata management

Each node of the grid declares the availability of K-grid objects (resources, components and services)

by publishing specific entries into the Directory Information Tree (DIT) maintained by a LDAP server such as the Grid Resource Information Service (GRIS) provided by Globus Toolkit.

For instance, by publishing the following LDAP entry:

```
dn: kgr=KMR, hn=NodeS.domain...,
   dc=domain,...
objectclass: KGridRepository
kgr: KMR
description: Knowledge Metadata
Repository
hn: NodeS.domain...
storedir: /opt/kgrid/KMR
...
```

The K-grid node *NodeS* declares, according to the definition of the objectclass *KGridRepository*, the availability of a KMR which stores metadata about K-grid resources, such as data sources and data mining tools, in the file system directory specified by the *storedir* attribute provided by that node.

Metadata are implemented by XML documents, on the basis of a set of specified schemas, and provide specific information for the discovery and the use of resources. For instance, metadata about data mining tools provide information about the implemented task (e.g., classification, clustering, regression), complexity of the used algorithm, location of executables and manuals, syntax for running the program, format of input data and results, etc.

##### 4.2. *Knowledge directory services*

The discovery of interesting resources over the *KNOWLEDGE GRID* is accomplished in two steps: the metadata repositories are first located, searching LDAP directories for K-grid entries. Then the relevant XML metadata are downloaded from the specified KMR and stored in the local KMR or directly in the local file system. The collected XML metadata are then analyzed to find more specific information. We implemented the basic tools to find, retrieve and select metadata about K-grid resources (e.g., data sources, data mining software), on the basis of different search parameters and selection filters. The useful metadata (i.e., those satisfying the searching

and filtering criteria) are then stored (see Fig. 4) in the TRC, a local cache which contains information about resources (nodes, data sources and algorithms) selected to perform a computation.

#### 4.3. Development of a KNOWLEDGE GRID application

When the discovery process is completed successfully, the information stored into the TRC is used to build graphic objects representing the resources needed to perform the applications that are then combined, in a graphic way, to compose a KNOWLEDGE GRID application. In particular, we are implementing, the Task Composer and the Execution Plan Generator, two Java programs that respectively implement the composition of an application and the generation of a suitable execution plan. The execution plans are modeled as graphs, where nodes represent basic tasks such as data movements, data filtering and program execution, and arcs represent the link among tasks. The produced execution plan is stored as a XML document (see Fig. 4) into the KEPR repository for a future use.

#### 4.4. Resource allocation and execution management

This component of the Core K-grid layer is responsible to start and manage the execution of a KNOWLEDGE GRID application. Currently, we implemented the Broker Adapter (see Fig. 4) that maps the execution plan, represented by an XML document, into a specific Globus RSL script, which can be directly executed by means of the `globusrun` program. In this way the KNOWLEDGE GRID application can be started as a Globus application by using its basic allocation services. The RAEM module will be completed by implementing dynamic monitoring and management functions for the submitted applications.

In summary, we implemented the basic KNOWLEDGE GRID services allowing the discovery of useful data sources and data mining tools, their graphic composition to form a KNOWLEDGE GRID application, the modeling of such application by means of an abstract execution plan and distributed execution of such application by using the dynamic information and the allocation services provided by Globus.

## 5. Conclusion and future work

The Grid infrastructure is growing up very quickly and is going to be more and more complete and complex both in the number of tools and in the variety of supported applications. Along this direction the Grid services are shifting from generic computation-oriented services to high level information management and knowledge discovery services. The KNOWLEDGE GRID system we discussed here is a significant component for devising and providing those services. It integrates and completes the data grid services by supporting distributed data analysis and knowledge discovery and management services that will enlarge the application scenario and the community of grid computing users [9].

Besides completing the KNOWLEDGE GRID implementation, we are designing a distributed tool, based on the JXTA peer-to-peer technology. JXTA technology is a set of open, generalized peer-to-peer protocols that allow any connected device on a network to communicate and collaborate in a peer-to-peer manner [10,11]. That KNOWLEDGE GRID discovery tool, will be based on the cooperation of mobile agents communicating each ones, in a decentralized manner, by using the peer-to-peer JXTA technology. Whenever a possibly useful resource for a KNOWLEDGE GRID computation is added to the Grid, the mobile agents will broadcast this information to the other agents, that in turn will update the local KMRs of each K-grid node.

## Acknowledgements

This work has been partially funded by the CNR Agenzia 2000 project “GRIGLIE COMPUTAZIONALI E APPLICAZIONI”.

## References

- [1] I. Foster, C. Kesselman, S. Tuecke, The anatomy of the grid: enabling scalable virtual organizations, *Intl. J. Supercomputer Appl.* 15 (3) (2001).
- [2] W. Hoschek, J. Jaen-Martinez, A. Samar, H. Stockinger, K. Stockinger, Data management in an international data grid project, in: *Proceedings of the IEEE/ACM International Workshop on Grid Computing Grid'2000*, LNCS, Vol. 1971, Springer, Berlin, December 2000, pp. 77–90.

- [3] P. Avery, I. Foster, GriPhyN Project Description, 2001. Available at <http://www.griphyn.org/info/index.html>.
- [4] Y. Morita, et al., Grid data farm for atlas simulation data challenges, in: Proceedings of the International Conference on Computing of High Energy and Nuclear Physics, 2001, pp. 699–701.
- [5] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, S. Tuecke, The data grid: towards an architecture for the distributed management and analysis of large scientific datasets, *J. Network Comput. Appl.* 23 (2001) 187–200.
- [6] I. Foster, Building the grid: an integrated services and toolkit architecture for next generation networked applications, Technical Report, 2000. Available at [http://www.gridforum.org/building\\_the\\_grid.htm](http://www.gridforum.org/building_the_grid.htm).
- [7] I. Foster, C. Kesselman, Globus: a metacomputing infrastructure toolkit, *Int. J. Supercomputing Appl.* 11 (1997) 115–128.
- [8] A.L. Prodromidis, P.K. Chan, S.J. Stolfo, Meta-learning in distributed data mining systems: issues and approaches, in: H. Kargupta, P. Chan (Eds.), *Advances in Distributed and Parallel Knowledge Discovery*, AAAI Press/MIT Press, 2000, pp. 81–87.
- [9] F. Berman, From TeraGrid to knowledge grid, *Commun. ACM* 44 (11) (2001) 27–28.
- [10] L. Gong, JXTA: a network programming environment, *IEEE Int. Comput.* 5 (3) (2001) 88–95.
- [11] <http://www.jxta.org/>.



**Mario Cannataro** is a Senior Researcher at the ICAR-CNR, Institute of the Italian National Research Council, and he is a lecturer in Computer Science at University of Calabria. He received his Laurea degree cum Laude in Computing Engineering from the University of Calabria and in 1986 he was awarded a fellowship for 2 years at CRAI. He is working in the area of parallel processing and computer

networks since 1988. His current research interests are in grid and cluster computing, adaptive Web systems, compression and synthesis of semi-structured data. He is a member of the ACM and IEEE Computer Society.



**Domenico Talia** is a Professor of Computer Science at the Faculty of Engineering of the University of Calabria, Italy. Talia received the Laurea degree in Physics at University of Calabria, from 1985 to 1996 he was a Researcher at CRAI (Consortium for Research and Applications of Informatics) and from 1997 to 2001, he was a Senior Researcher at the ISI-CNR, Institute of System Analysis and Information Technology of the Italian National Research Council. His main research interests include parallel computing, parallel data mining, parallel programming languages, cellular automata, computational science, and grid computing. Talia is a member of the Editorial Boards of the IEEE Computer Society Press, the Future Generation Computer Systems journal, the Parallel and Distributed Practices journal, and the Information journal. He is also a member of the Advisory Board of Euro-Par and he served as Distinguished Speaker in the IEEE Computer Society Distinguished Visitors Program. He published three books and more than 110 papers in international journals and conference proceedings. He is member of the ACM and the IEEE Computer Society.



**Paolo Trunfio** is a PhD student in Computer Engineering at the University of Calabria, Italy. Since 2001 he collaborates with the Institute of System Analysis and Information Technology of the Italian National Research Council in the area of grid computing. His current research interests include parallel and distributed computing, and data mining.