2. Grid-Based Data Mining and Knowledge Discovery

Mario Cannataro[1], Antonio Congiusta[2], Carlo Mastroianni[3], Andrea Pugliese[2], Domenico Talia[2], and Paolo Trunfio[2]

[1] Informatics and Biomedical Engineering, University Magna Græcia of Catanzaro, Via T. Campanella, 115 88100 Catanzaro, Italy
[2] DEIS, Università della Calabria, Via P. Bucci, Cubo 41/C 87036 Rende (CS), Italy
[3] ICAR-CNR, Italian National Research Council, Via P. Bucci, Cubo 41/C 87036 Rende (CS), Italy

Abstract

The increasing use of computers in all the areas of human activities is resulting in huge collections of digital data. Databases are common everywhere and are used as repositories of every kind of data. Knowledge discovery techniques and tools are used today to analyze those very large data sets to identify interesting patterns and trends in them. When data is maintained over geographically distributed sites the computational power of distributed and parallel systems can be exploited for knowledge discovery in databases. In this scenario the Grid can provide an effective computational support for distributed knowledge discovery on large data sets. To this purpose we designed a system called *Knowledge Grid*. This chapter describes the *Knowledge Grid* architecture and discusses some related systems and models recently proposed for knowledge discovery on Grids. The chapter presents also how to design and implement distributed data mining applications by using the *Knowledge Grid* tools starting from searching Grid resources, composing software and data elements, and executing the resulting application on a Grid.

2.1 Introduction

Enlarging our knowledge about the world and the secrets of life is one of the strongest motivations of the human activities. The use of computers is changing our way to make discoveries and is improving both speed and quality of the discovery processes. Advances in electronic data gathering, storage, and distribution technologies have far outpaced computational advances in techniques for analyzing and understanding data. This has created the need for models, tools, and techniques for automated data mining and *Knowledge Discovery in Databases (KDD)*. These terms indicate the automated analysis of large volumes of data stored in computers, looking for the relationships and knowledge that are implicit in large volumes of data and are "interesting" for users.

To manage the very large amount of data available today, computer scientists are working on efficient systems, algorithms, and applications that can

handle and analyze very large data repositories. Intensive data-consuming applications are running on massive amounts of data with the task of extracting valuable knowledge. Data mining is one of the key technologies in this scenario. However, these intensive data-consuming applications suffer from performance problems and single database sources.

Distributed architectures supported by high performance networks and knowledge-based middleware offer parallel and distributed databases a great opportunity to support cost-effective everyday applications. Moreover, using distributed computing systems and tools allows users to share large data sources, the mining process building, and the extracted knowledge. Large communities of users can pool their resources from different sites of a single organization or from a large number of institutions and perform all the steps of the KDD process from remote sites according to a cooperative approach.

Grid computing is an innovative distributed computational model focusing on large-scale resource sharing, innovative applications, and high-performance orientation. Grids can be used today as effective infrastructures for distributed high-performance computing and data processing [2.1]. Grid application areas are shifting from scientific computing towards industry and business applications. To meet those needs *data Grids* are designed to store, move, and manage large data sets located in remote sites. Data Grids represent an enhancement of computational Grids, driven by the need to handle large data sets without constant, repeated authentication, aiming to support the implementation of distributed data-intensive applications. Significant examples are the EU DataGrid [2.2], the Particle Physics Data Grid [2.3], the Japanese Grid DataFarm [2.4] and the Globus Data Grid [2.5].

As an advancement of the data Grid concept, it is imperative to develop knowledge-based Grids that may offer tools and environments to support the process of analysis, inference, and discovery over data available in many scientific and business areas. These environments will 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 can occur in industry and commerce, where analysts need to be able to mine the large volumes of information that can be distributed over different plants to support corporate decision making.

Knowledge Grids 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, 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 chapter is as follows. Section 2.2 discusses related work. Section 2.3 briefly describes the *Knowledge Grid* architecture and the main features of its components. Section 2.4 describes the metadata model adopted to describe the resources used for developing the data mining applications. Sections 2.5–2.6 discuss how the tools of the *Knowledge Grid* support a user in designing, building, and executing a distributed data mining application. Section 2.7 concludes the chapter.

2.2 Knowledge Discovery on Grids

Berman [2.6], Johnston [2.7], and some of us [2.8, 2.9] claimed that the creation of knowledge Grids on top of computational Grids is the enabling condition for developing high-performance knowledge discovery processes and meeting the challenges posed by the increasing demand of power and abstractness coming from complex Problem-Solving Environments. The design of knowledge Grids can benefit from the layered Grid architecture, with lower levels providing middleware support for higher level application-specific services.

Whereas some high-performance parallel and distributed knowledge discovery (*PDKD*) systems recently appeared [2.10] (see also [2.8]), there are few projects attempting to implement and/or support knowledge discovery processes over computational Grids. A main issue here is the integration of two main demands: synthesizing useful and usable knowledge from data, and performing sophisticated large-scale computations leveraging the Grid infrastructure. Such integration must pass through a clear representation of the knowledge base used to translate moderately abstract domain-specific queries into computations and data analysis operations able to answer such queries by operating on the underlying systems [2.6].

In the remainder of this section we review the most significant systems oriented at supporting knowledge discovery processes over distributed/Grid infrastructures. The systems discussed here provide different approaches to supporting knowledge discovery on Grids. We discuss them starting from general frameworks, such as the TeraGrid infrastructure, then outlining data-intensive oriented systems, such as DataCutter and InfoGrid, and, finally, describing KDD systems similar to the *Knowledge Grid*, such as Discovery Net, and some significant data mining testbed experiences.

The *TeraGrid* project is building a powerful Grid infrastructure, called *Distributed TeraScale Facility* (*DTF*), connecting four main sites in the USA (the San Diego Supercomputer Center, the National Center for Supercomputing Applications, Caltech, and Argonne National Laboratory). Recently, the NSF funded the integration into the DTF of the *TeraScale Computing System* (*TCS-1*) at the Pittsburgh Supercomputer Center; the resulting Grid environment will provide, besides tera-scale data storage, 21 TFLOPS of

computational capacity [2.11]. Furthermore, the TeraGrid network connections, whose bandwidth is in the order of tenths of Gbps, have been designed in such a way that all resources appear as a single physical site. The connections have also been optimized to support peak requirements rather than an average load, as is natural in Grid environments. The TeraGrid adopts Grid software technologies and, from this point of view, appears as a "virtual system" in which each resource describes its own capabilities and behavior through *Service Specifications*. The basic software components are called *Grid Services*, and are organized into three distinct layers. The *Basic* layer comprises authentication, resource allocation, data access, and resource information services; the *Core* layer comprises services such as advanced data management, single job scheduling, and monitoring; the *Advanced* layer comprises superschedulers, resource discovery services, repositories, etc. Finally, *TeraGrid Application Services* are built using Grid Services. The definition of such services is still under discussion, but they should comprise, for example, the support of on-demand/interactive applications, the support of GridFTP interface to data services, etc.

The most challenging application on the TeraGrid will be the synthesis of knowledge from very large scientific data sets. The development of knowledge synthesis tools and services will enable the TeraGrid to operate as a Knowledge Grid. A first application is the establishment of the Biomedical Informatics Research Network to allow brain researchers at geographically distributed advanced imaging centers to share data acquired from different subjects and using different techniques. Such applications make a full use of a distributed data Grid with hundreds of terabytes of data online, enabling the TeraGrid to be used as a knowledge Grid in the biomedical domain. The use of the *Knowledge Grid* services can be potentially effective in these applications [2.12].

InfoGrid is a service-based data integration middleware engine designed to operate on Grids. Its main objective is to provide information access and querying services to knowledge discovery applications [2.13]. The information integration approach of InfoGrid is not based on the classical idea of providing a "universal" query system: instead of abstracting everything for users, it gives a personalized view of the resources for each particular application domain. The assumption here is that users have enough knowledge and expertise to handle the absence of "transparency". In InfoGrid the main entity is the *Wrapper*; wrappers are distributed on a Grid and each node publishes a directory of the wrappers it owns. A wrapper can wrap information sources and programs, or can be built by composing other wrappers (*Composite Wrapper*). Each wrapper provides: (*i*) a set of query construction interfaces that can be used to query the underlying information sources in their native language, and (*ii*) a set of administration interfaces that can be used to configure its properties (access metadata, linkage metadata, configuration files). In summary, InfoGrid puts the emphasis on delivering metadata

describing resources and providing an extensible framework for composing queries.

DataCutter is another middleware infrastructure that aims to provide specific services for the support of multi-dimensional range querying, data aggregation, and user-defined filtering over large scientific data sets in shared distributed environments [2.14]. DataCutter has been developed in the context of the *Chaos* project at the University of Maryland; it uses and extends features of the *Active Data Repository* (*ADR*), that is a set of tools for the optimization of storage, retrieval, and processing of very large multi-dimensional data sets. In ADR, data processing is performed at the site where data is stored, whereas in Grid environments this is naturally unfeasible, due to inherent data distribution and resource sharing at servers, which may lead to inefficiencies.

To overcome this, in the DataCutter framework an application is decomposed into a set of processes, called *filters*, that are able to perform a rich set of queries and data transformation operations. Filters can execute anywhere but are intended to run on a machine close (in terms of connectivity) to the storage server. DataCutter supports efficient indexing. In order to avoid the construction of a huge single index that would be very costly to use and keep updated, the system adopts a multi-level hierarchical indexing scheme, specifically targeted at the multi-dimensional data model adopted.

Different from the two environments discussed above, the *Datacentric Grid* is a system directed at knowledge discovery on Grids designed for mainly dealing with immovable data [2.15]. The system consists of four kinds of entities. The nodes at which computations happen are called *Data/Compute Servers* (*DCS*). Besides a compute engine and a data repository, each DCS comprises a *metadata tree*, that is a structure for maintaining relationships among raw data sets and models extracted from them. Furthermore, extracted models become new data sets, potentially useful at subsequent steps and/or for other applications.

The *Grid Support Nodes* (*GSNs*) maintain information about the whole Grid. Each GSN contains a directory of DCSs with static and dynamic information about them (e.g., properties and usage), and an execution plan cache containing recent plans along with their achieved performance. Since a computation in the Datacentric Grid is always executed on a single node, execution plans are simple. However, they can start at different places in the model hierarchy because, when they reach a node, they may find already computed models. The *User Support Nodes* (*USNs*) carry out execution planning and maintain results. USNs are basically proxies for user interface nodes (called *User Access Points* or *UAPs*). This is because user requests (i.e., task descriptions) and their results can be small in size, so in principle UAPs could be simple devices not always online, and USNs could interact with the Datacentric Grid when users are not connected.

An agent-based data mining framework, called *ADaM* (*Algorithm Development and Mining*), has been developed at the University of Alabama [2.16]. Initially, this framework was adopted for processing large data sets for geophysical phenomena. More recently, it has been ported to the NASA's *Information Power Grid* (*IPG*) environment, for the mining of satellite data [2.17]. In this system, the user specifies *what* is to be mined (data set names and locations) and *how* and *where* to perform the mining (sequence of operations, required parameters, and IPG processors to be used). Initially, "thin" agents are associated with the sequence of mining operations; such agents acquire and combine the needed mining operations from repositories that can be public or private, i.e., provided by mining users or private companies. Data is acquired "on-the-fly", through SRB/MCAT and GridFTP, in order to minimize storage requirements at the mining site. ADaM comprises a moderately rich set of interoperable operation modules, comprising *data readers* and *writers* for a variety of formats, *preprocessing modules*, for example for data subsetting, and *analysis modules* providing data mining algorithms.

The InfoGrid system mentioned before has been designed as an application-specific layer for constructing and publishing knowledge discovery services. In particular, it is intended to be used in the *Discovery Net* (*D-NET*) system. D-NET is a project of the Engineering and Physical Sciences Research Council at Imperial College [2.18, 2.19] whose main goal is to design, develop, and implement an infrastructure to effectively support scientific knowledge discovery processes from high-throughput informatics. In this context, a series of testbeds and demonstrations are being carried out for using the technology in the areas of life sciences, environmental modeling, and geo-hazard prediction.

The building blocks in Discovery Net are the so-called *Knowledge Discovery Services* (*KDSs*), comprising *Computation Services* and *Data Services*. The former typically comprise algorithms, e.g., data preparation and data mining, while the latter define relational tables (as queries) and other data sources. Both kinds of services are described by means of *adapters* that provide information about input and output data types, allowed parameters, location, platform constraints, and available *factories*, i.e. objects allowing for the creation of service instances. KDSs are used to compose moderately complex data-pipelined processes. The composition may be carried out by means of a GUI which provides access to a library of services. The XML-based language used to describe processes is called *Discovery Process Markup Language* (*DPML*). Each composed process can be deployed and published as a new process. Typically, process descriptions are not bound to specific servers since the actual resources are later resolved by lookup servers (see below).

Discovery Net is based on an open architecture using common protocols and infrastructures such as the Globus Toolkit. Servers are distinguished into (*i*) *Knowledge Servers*, allowing storage and retrieval of knowledge (meant as raw data and knowledge models) and processes; (*ii*) *Resource Discovery*

Servers, providing a knowledge base of service definitions and performing resource resolution; and (*iii*) *Discovery Meta-Information Servers*, used to store information about the *Knowledge Schema*, i.e., the sets of features of known databases, their types, and how they can be composed with each other.

Finally, we outline here some interesting data mining testbeds developed at the National Center for Data Mining (NCDM) at the University of Illinois at Chicago (UIC) [2.20]:

- *The Terra Wide Data Mining Testbed* (*TWDM*). TWDM is an infrastructure for the remote analysis, distributed mining, and real time exploration of scientific, engineering, business, and other complex data. It consists of five geographically distributed nodes linked by optical networks through *StarLight* (an advanced optical infrastructure) in Chicago. These sites include StarLight, the Laboratory for Advanced Computing at UIC, SARA in Amsterdam, and Dalhousie University in Halifax. In 2003 new sites will be connected, including Imperial College in London. A central idea in TWDM is to keep generated predictive models up-to-date with respect to newly available data in order to achieve better predictions (as this is an important aspect in many "critical" domains, such as infectious disease tracking). TWDM is based on *DataSpace*, another NCDM project for supporting real-time streaming data. In DataSpace the *Data Tranformation Markup Language* (*DTML*) is used to describe how to update "profiles", aggregate data which are inputs of predictive models, on the basis of new "events", i.e., new bits of information.
- *The Terabyte Challenge Testbed.* The Terabyte Challenge Testbed is an open, distributed testbed for DataSpace tools, services, and protocols. It involves a number of organizations, including the University of Illinois at Chicago, University of Pennsylvania, University of California at Davis, Imperial College. The testbed consists of ten sites distributed over three continents connected by high performance links. Each site provides a number of local clusters of workstations which are connected to form wide area *meta-clusters* maintained by the *National Scalable Cluster Project*. So far, meta-clusters have been used by applications in high energy physics, computational chemistry, nonlinear simulation, bioinformatics, medical imaging, network traffic analysis, and digital libraries of video data. Currently, the Terabyte Challenge Testbed consists of approximately 100 nodes and two terabytes of disk storage.
- *The Global Discovery Network* (*GDN*). The GDN is a collaboration between the Laboratory for Advanced Computing of the National Center for Data Mining and the Discovery Net project (see above). It will link the Discovery Net to the Terra Wide Data Mining Testbed to create a combined global testbed with a critical mass of data.

In summary, many of the recent knowledge discovery-oriented systems have been designed for specific domains, and have later been extended to

support more general applications. Some of such systems are essentially advanced interfaces for integrating, accessing, and elaborating large data sets. Furthermore, they provide specific functionalities for the support of typical knowledge discovery processes. The *Knowledge Grid* we designed is one of the first attempts to build a domain-independent knowledge discovery environment on the Grid. Moreover, our system provides services specifically designed for the integration of parallel and sequential data mining algorithms, and the management of base data sets and extracted knowledge models.

2.3 The *Knowledge Grid* Architecture

The *Knowledge Grid* architecture [2.8] 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 [2.21], these services can be developed in different ways using the available Grid toolkits and services. The current implementation is based on the Globus toolkit [2.22]. 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.

2.3.1 *Knowledge Grid* Services

The *Knowledge Grid* services are organized in two hierarchic levels: the Core K-grid layer and the High level K-grid layer depicted in Fig. 2.1. The figure shows layers (as implemented on top of Globus services), the *Knowledge Grid* data, and metadata repositories. In the following the term *K-grid node* will denote a Globus node implementing the *Knowledge Grid* services.

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 goal is 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 the application requirements and the available Grid resources. The core K-grid layer comprises two main services:

– The *Knowledge Directory Service* (*KDS*) extends the basic Globus MDS service and is responsible for maintaining metadata describing data and tools used in the *Knowledge Grid*. They comprise repositories of data to be mined (data sources), tools, and algorithms used to extract, analyze,

and manipulate data, distributed knowledge discovery execution plans, and knowledge obtained as result of the mining process, i.e., learned models and discovered patterns. The metadata information is represented by *eXtensible Markup Language* (*XML*) documents stored in a *Knowledge Metadata Repository* (*KMR*).
– The *Resource Allocation and Execution Management Service (RAEMS)* is 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 is obtained by co-allocating resources. After the execution plan activation, this layer manages and coordinates the application execution and the storing of knowledge results in the *Knowledge Base Repository* (*KBR*). Resource requests of each single data mining process are expressed using the *Resource Specification Language* (*RSL*) [2.23].

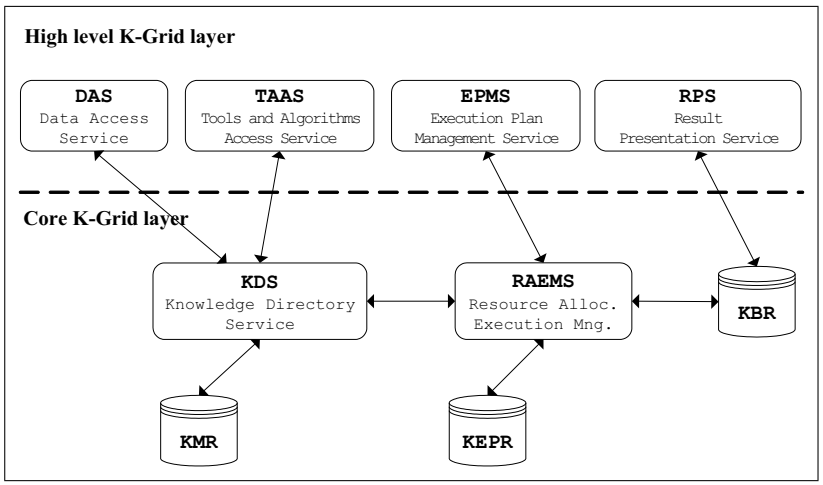

Fig. 2.1. The *Knowledge Grid* architecture

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 here are:

– The *Data Access Service* (*DAS*) is responsible for the search, selection (data search services), extraction, transformation, and delivery (data extraction services) of data to be mined.
– The *Tools and Algorithms Access Service* (*TAAS*) is responsible for searching, selecting, and downloading data mining tools and algorithms.

- The *Execution Plan Management Service* (*EPMS*). An execution plan is represented by a graph describing interactions and data flows between data sources, extraction tools, DM tools, and visualization tools. The Execution Plan Management Service 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 *Knowledge Execution Plan Repository* (*KEPR*).
- The *Results Presentation Service* (*RPS*) specifies how to generate, present, and visualize the knowledge models extracted (e.g., association rules, clustering models, classifications). The results metadata are stored in the KMR to be managed by the KDS.

2.4 An XML-Based Metadata Model for the *Knowledge Grid*

The large heterogeneity of the resources involved in a distributed data mining computation is tackled in the *Knowledge Grid* through the definition of a suitable metadata model that is exploited by core and high level services to manage resources in a standard and efficient way. The resources involved in a typical distributed data mining computation are:

- computational resources (computers, storage devices, etc.);
- data to be mined, such as databases, plain files, semi-structured 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 mine data, i.e., data mining tools available on the Grid nodes;
- knowledge obtained as a result of the mining process, i.e., learned models and discovered patterns;
- tools and algorithms used to visualize, store, and manipulate discovered models.

Heterogeneity arises mainly from the large variety of resources within each category. For instance, software can run only on some particular host machines whereas data can be extracted from different data management systems such as relational databases, semi-structured databases, plain files, etc. The management of such heterogeneous resources requires an intense use of metadata, whose purpose is to provide information about the features of resources and their effective use. Since metadata represents a key element for effective resource discovery and utilization, Grids need to use mechanisms and models that define rich metadata schemas able to represent the variety of resources involved.

The *Knowledge Grid* uses the Globus MDS, and therefore the LDAP protocol, to publish, discover, and manage information about the generic resources of the underlying Grid (e.g., CPU performance, memory size, etc.). However, the complexity of the information associated with specific *Knowledge Grid* resources (data sources, mining algorithms, models), led us to design a more effective model to represent and manage the corresponding metadata.

The basic objectives that guided us through the definition of the *Knowledge Grid* metadata model are the following:

- Metadata should document in a simple and human-readable fashion the features of a data mining application and of the resources involved.
- Metadata should allow an effective search of resources.
- Metadata should provide an efficient way to access resources.

To satisfy such requirements, we chose to express metadata in XML, as it provides a set of functionalities and capabilities that are making it a common model for data description and exchange.

XML metadata documents are defined according to a set of XML schemas properly defined to describe and categorize the different classes of resources. In the remainder of this section, we will give some hints on the definition of metadata related to software, data sources, data mining tools, and discovered knowledge. Furthermore, we will show how metadata can be used to distinguish abstract from concrete resources, and we will introduce execution plans, also defined with an XML formalism, used by the *Knowledge Grid* to manage complex data mining applications. More information on the topical issue of metadata management in the *Knowledge Grid* can be found in [2.24].

2.4.1 Data Mining Software

Categorization of data mining software is based on the following classification parameters [2.25]:

- the kind of data sources the software works on;
- the kind of knowledge that is to be discovered by the software;
- the type of techniques that the software uses in the data mining process; and
- the driving method, i.e., whether the mining process is autonomous, driven by data or queries, or driven by the user (interactive).

As an example, Fig. 2.2 reports the XML metadata related to the data mining software *AutoClass*. The XML document is composed of two parts. The first part is the software `Description`, the second one is the software `Usage`. The `Description` section specifies, for each classification parameter, one or more values that characterize the software. The `Usage` section contains all the information that can be used by a client to access and use the software. This section is composed of a set of subsections, among which are

`Syntax`, `Hostname`, `ManualPath`, and `DocumentationURL`. The `Syntax` subsection describes the format of the command that the client should use to invoke the software. This subsection is defined as a tree, where each node is an `Arg` element and the root is the name of the software itself. The root children specify the arguments that should follow the software name in the software invocation, and these arguments can in turn have children, i.e., sub-arguments, and so on. Each `Arg` element has the following attributes: the `description` attribute, which is a textual description of the argument and the `type` attribute, which specifies whether the argument is `optional`, `required`, or `alternative`. In the last case, all the sibling arguments should have the same value for this attribute, meaning that only one of the siblings should be used in the software invocation. Finally, the `value` attribute (optional) specifies the fixed value of the argument. If the `value` attribute is omitted, the value is to be provided by the client. In the example shown in Fig. 2.2, in the `AutoClass` execution command the executable name should be followed by the `-search` argument, to ask for a classification, or by the `-reports` argument, to obtain the model file. If the `-search` argument is chosen, it should be followed by four sub-arguments, all `required`. Therefore, `AutoClass` can be invoked with the command `/usr/autoclass/autoclass -search aFile.db2 aFile.hd2 aFile.model aFile.s-params`.

```
<DataMiningSoftware name="AutoClass">
  <Description>
    <KindOfData>flat file</KindOfData>
    <KindOfKnowledge>clusters</KindOfKnowledge>
    <KindOfTecnique>statistics</KindOfTecnique>
    <DrivingMethod>autonomous knowledge miner</DrivingMethod>
  </Description>
  <Usage>
    ...
    <Syntax>
      <Arg description="executable" type="required" value="/usr/autoclass/autoclass">
        <Arg description="make a classification" type="alternative" value="-search">
          <Arg description="a .db2 file" type="required"/>
          <Arg description="a .hd2 file" type="required"/>
          <Arg description="a .model file" type="required"/>
          <Arg description="a .s-params file" type="required"/>
        </Arg>
        <Arg description="create a report" type="alternative" value="-reports">
          <Arg description="a .results-bin file" type="required"/>
          ...
        </Arg>
        ...
      </Arg>
    </Syntax>
    <Hostname>icarus.cs.icar.cnr.it</Hostname>
    <ManualPath>/usr/autoclass/read-me.text</ManualPath>
    <DocumentationURL>http://ic-www.arc.nasa.gov/ic/projects/...</DocumentationURL>
    ...
  </Usage>
</DataMiningSoftware>
```

Fig. 2.2. An extract from an XML metadata sample for the AutoClass software

2.4.2 Data Sources

Data sources are the input on which data mining algorithms work to extract new knowledge. They can be provided by relational databases, plain files, and other structured and semi-structured documents. In spite of the wide variety of the possible data source types, we aim to define a common structure of data source metadata in order to standardize access and search operations on such resources.

The common structure of source metadata is composed of two parts: (*i*) an `Access` section including information for retrieving the data source (location, size, etc.), and (*ii*) a `Structure` section providing information about the data source logical and/or physical structure.

As an example, Fig. 2.3 shows an XML metadata document for a flat file that can be used as an input by the `AutoClass` software. The `Structure` section includes two subsections, `Format` and `Attributes`. The `Format` subsection contains information about the physical structure of the flat file, e.g., the strings that are used to separate the records and the attributes within a record. The `Attributes` subsection contains information about the logical structure, i.e., it lists the table attributes and provides the relative specifications (such as the name of the `Attribute`, its type, etc.).

The high-level metadata model is the same for all kinds of data sources, but some details can be added or omitted depending on the specific kind of data to be represented. For example, in a relational database no information is needed to describe the structure of data, since it is directly managed by the database management system.

2.4.3 Abstract and Concrete Resources

The resources discussed so far (software, data sources, models) can be either *concrete* or *abstract*. A concrete resource is a resource, published on a KMR of a *Knowledge Grid* node, which is completely specified by its metadata. In abstract resource metadata, some features are expressed as constraints and not as well known values: an abstract resource is not used to identify a specific resource, but to define a set of requirements on the resources that should be discovered by the KDS service.

For instance, whereas the metadata described in Fig. 2.2 describes the concrete software `Autoclass` available on a given node, the metadata document shown in Fig. 2.4 describes an abstract data mining software able to perform a clustering computation on flat files.

An abstract resource can be instantiated into an existing concrete resource whose metadata matches the specified constraints.

2.4.4 Execution Plans

A distributed data mining computation is a process composed of several steps which are executed sequentially or in parallel. In the *Knowledge Grid* frame-

```
<FlatFile>
  <Access>
    <Location>/usr/share/imports-85c.db2</Location>
    <Size>26756</Size>
    ...
  </Access>
  <Structure>
    <Format>
      <AttributeSeparatorString>,</AttributeSeparatorString>
      <RecordSeparatorString>#</RecordSeparatorString>
      <UnknownTokenString>?</UnknownTokenString>
      ...
    </Format>
    <Attributes>
      <Attribute name="symboling" type="discrete">
        <SubType>nominal</SubType>
        <Parameter>range 7</Parameter>
      </Attribute>
      <Attribute name="normalized-loses" type="real">
        <SubType>scalar</SubType>
        <Parameter>zero_point 0.0</Parameter>
        <Parameter>rel_error 0.01</Parameter>
      </Attribute>
      ...
    </Attributes>
  </Structure>
</FlatFile>
```

Fig. 2.3. An extract from an XML metadata sample for a flat file

```
<DataMiningSoftware name="genericSoftware">
  <Description>
    <KindOfData>flat file</KindOfData>
    <KindOfKnowledge>clusters</KindOfKnowledge>
  </Description>
</DataMiningSoftware>
```

Fig. 2.4. An XML metadata sample for a generic clustering software

work, the management of complex data mining processes is carried out by defining an execution plan. An execution plan is a graph that describes interactions and data flows between data sources, data mining tools, visualization tools, and output models.

Execution plans are also described through an XML formalism. Such descriptions contain information about relationships among atomic tasks (e.g., computations and data transfers), along with references to the XML documents that describe the involved resources.

Execution plans can be either *abstract* or *instantiated*. An abstract execution plan contains at least one abstract resource, whereas an instantiated execution plan contains only concrete resources. Such a distinction is made to take into account the dynamic nature of a Grid environment, in which resources fail and become available, data gets deleted, software gets updated,

etc. In general, a user builds an abstract execution plan, and the RAEMS attempts to transform it into an instantiated execution plan, by mapping abstract resources into concrete resources. Such an action is performed by a scheduler that allows the generation of a suitable execution plan. From an abstract execution plan, different instantiated execution plans could be generated, depending on the resources that are available on the *Knowledge Grid* at different times. This topic will be further discussed in Sect. 2.5, where a sample execution plan will be shown.

2.4.5 Data Mining Models

The knowledge discovered through the data mining process is represented by "data mining models". Whereas so far no common models have been defined for the definition of the data mining resources discussed before, a standard model called *Predictive Model Markup Language* (*PMML*) has been defined to describe data mining results. PMML is an XML language which provides a vendor-independent method for defining data mining models [2.26]. The PMML provides a *Document Type Definition* (*DTD*) to describe different kinds of models such as classification rules and association rules. We use it to define data mining models in the *Knowledge Grid*.

2.4.6 Metadata Management

As stated before, the metadata management process is a key aspect in the development of data mining applications over the *Knowledge Grid*. A typical life cycle of metadata consists of the following steps:

1. Resource metadata are published on the KMRs of the corresponding nodes.
2. The user specifies the features of the resources needed to design a data mining application.
3. The DAS and TAAS services search the KMRs of the *Knowledge Grid* nodes for the requested resources, using the core level KDS service (which directly interacts with local and remote KMRs) to manage resource metadata.
4. Metadata describing the resources of interest are delivered by such services to the requesting user. Figure 2.5 shows the flows of resource-related metadata in the *Knowledge Grid* architecture.
5. Metadata related to software, data, and operations are combined into an execution plan to design a complete data mining application. The execution plan metadata are managed by the EPMS service; they are accessed through the core level RAEMS service and stored in the KEPR (see Fig. 2.6).

6. After application execution, results and related model metadata are stored in the KBR and processed by the RPS service. Moreover, metadata related to new and/or modified resources are published in the corresponding KMRs for future use (see Fig. 2.7).

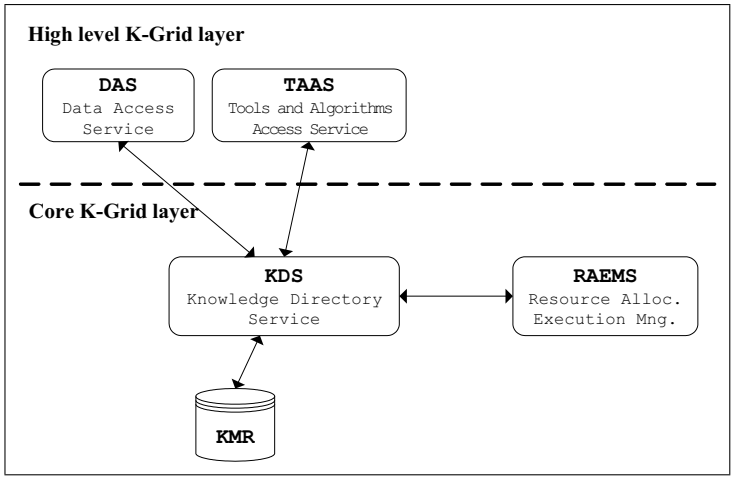

Fig. 2.5. Resource metadata flows and services involved

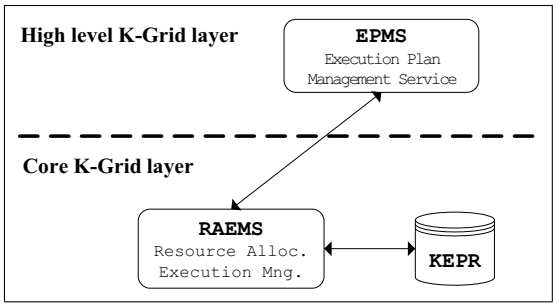

Fig. 2.6. Execution plan metadata flows and services involved

2.5 Design of a PDKD Computation

The design of a PDKD computation on the *Knowledge Grid* is performed as shown in Fig. 2.8. The process starts by searching and selecting the resources to be used in a PDKD computation. This step is accomplished by

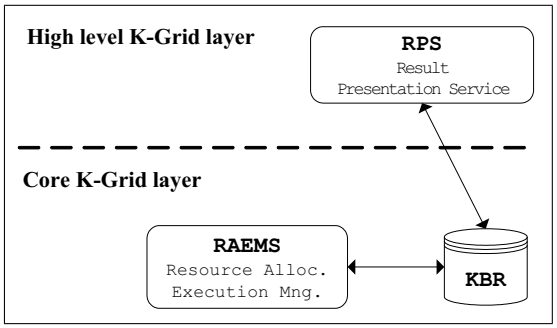

Fig. 2.7. Knowledge metadata flows and services involved

means of the DAS and TAAS tools that analyze the XML metadata documents stored in the KMRs of the participant K-grid nodes, which represent the resources available on those nodes. Such an analysis attempts to find information about useful resources (e.g., software implementing a specific data mining algorithm, data sources about a specific argument, etc.), and is carried out on the basis of the search parameters and selection filters chosen by the user. Metadata about the resources selected for the computation (i.e., those satisfying the searching and filtering criteria) are then stored in the *Task Metadata Repository* (*TMR*), a local storage space that contains information about resources (computational nodes, data sources, and software) selected to perform a computation. The TMR is organized as a set of directories: each one is named with the fully qualified hostname of a Grid node, and contains metadata documents about its resources.

The design of a PDKD computation is performed by means of the EPMS. To allow a user to build the computation in a simple way, we developed a toolset named *VEGA* (*Visual Environment for Grid Applications*). Its architecture is depicted in Fig. 2.9. VEGA integrates functionalities of the EPMS and other K-grid services; in particular, it provides the following EPMS operations:

- *task composition*, i.e., definition of the entities involved in the computation and specification of the relationships among them;
- *checking* of the consistency of the planned task;
- *generation* of the execution plan for the task.

2.5.1 Task Composition

The task composition phase is performed by means of a graphical interface (see Fig. 2.10), which provides a user with a set of graphical objects representing the resources (data sets, data mining tools, Grid nodes). These objects can be composed using visual facilities that allow a user to insert links among them, forming a graphical representation of the computation. In particular,

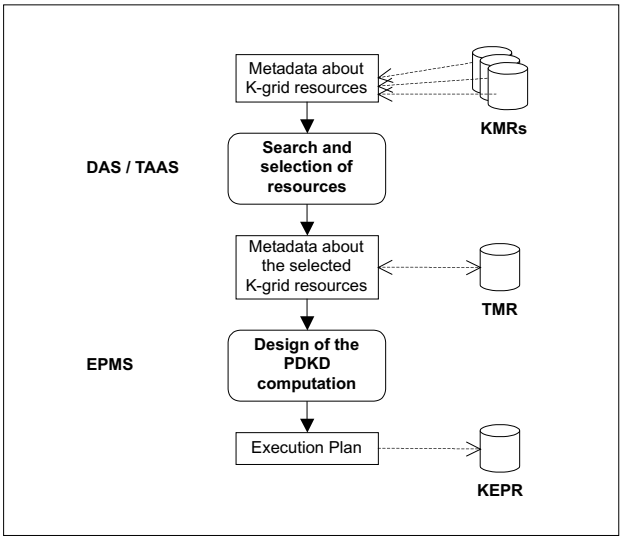

Fig. 2.8. The PDKD computation design process

such a phase is realized by the *Resource Manager*, the *Object Manager*, and the *Workspace Manager*.

The Resource Manager permits a user to browse the TMR in order to search and choose the resources to be used in the computation. Selected hosts are displayed into the *Hosts* panel, and the user can explore resources of each one by clicking on its label. Those resources are displayed by categories in the *Resources* panel.

The Object Manager deals with the graphical objects during the visual composition. Each graphical object is associated with information about the related resources; this information is used for the creation of the internal model and for the execution plan generation. The Object Manager handles three kinds of objects: data, software, and hosts. It allows the user to drag the objects presented in the hosts and resources panels into a workspace. After this, those objects can be linked to indicate the interactions among them. Links can represent different actions, such as data transfer, program execution, and input and output relationships. The Object Manager performs the labeling of the links and the attribution of other properties characterizing them. The *data transfer* link is used to move resources among different locations of the Grid. The *execute* link is used to run an application on a Grid host; the *input* and *output* links are used to respectively indicate input and output of a program. For each link type it is possible to set related parameters (e.g., protocol and destination path of the data transfer, job manager of the execution, etc.).

A complex computation is composed of several jobs. The design environment is organized into *workspaces*. Jobs present in a given workspace can

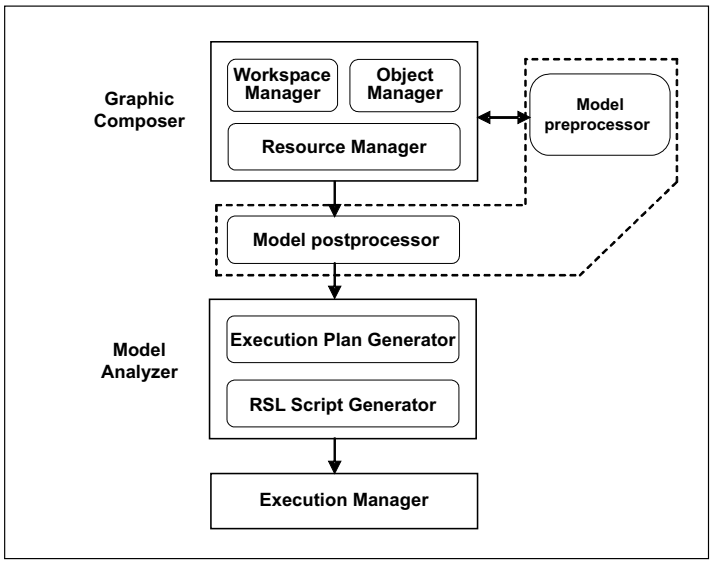

Fig. 2.9. The VEGA software modules

be executed concurrently, whereas workspaces are executed sequentially. To this end, an ordering between the workspaces is defined. In addition, the Workspace Manager handles an internal model of the graphical representation shown to the user. We describe here an example that shows the task composition process.

A user logged on the K-grid node `g1.isi.cs.cnr.it` intends to perform a data mining application composed of two data mining steps, clustering and classification, on the data set `Unidb` stored on the same node. The data set must be clustered using three different algorithms running in parallel on three copies of the data set. Clustering results must be analyzed by a classification algorithm that will be executed in parallel on three different nodes, generating three classification models of the same data set. Finally, the three different models will be shown to the user, who will select the more accurate one. The user has located the K-grid nodes `k1.deis.unical.it`, `k2.deis.unical.it`, and `k3.deis.unical.it` offering, respectively, the clustering algorithms K-Means [2.27], Intelligent Miner [2.28], and AutoClass [2.29], and the node `g2.isi.cs.cnr.it`, offering the C5.0 classifier [2.30].

Figures 2.10–2.13 show the sequence of the four workspaces composed by the user to design such a computation:

- *Workspace 1* (Fig. 2.10). The data set `Unidb` (which is located on the node `g1`) and the classifier C5.0 (which is located on `g2`) are copied to the nodes `k1`, `k2`, and `k3`.
- *Workspace 2* (Fig. 2.11). The data set `Unidb` is analyzed on `k1` by K-Means, producing as output `K-Means.out`; on `k2`, the data set `Unidb` is analyzed

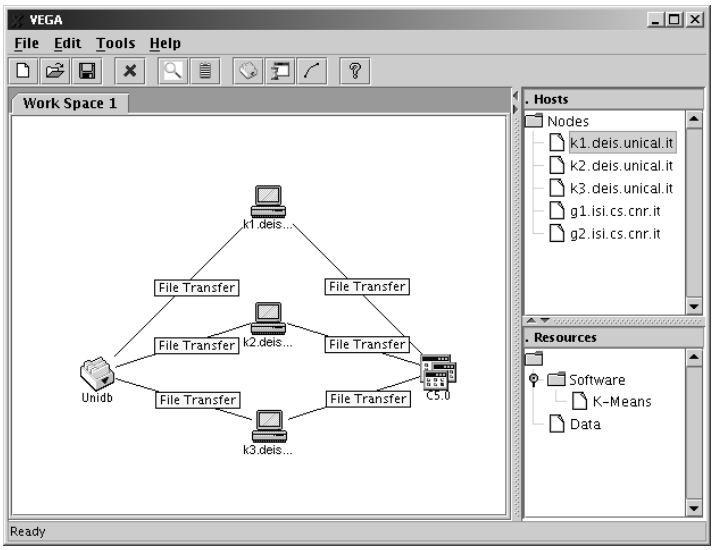

Fig. 2.10. VEGA: Example workspace 1

by Intelligent Miner, producing `Iminer.out`; on `k3`, the data set `Unidb` is analyzed by AutoClass, producing `Autoclass.out`.

– *Workspace 3* (Fig. 2.12). On `k1`, `K-Means.out` is analyzed by C5.0, producing `K-Means_c5.out`; on `k2`, `IMiner.out` is analyzed by C5.0, producing `Iminer_c5.out`; on `k3`, `Autoclass.out` is analyzed by C5.0, producing `Autoclass_c5.out`.
– *Workspace 4* (Fig. 2.13). `K-Means_c5.out`, `IMiner_c5.out`, and `Autoclass_c5.out` are moved from `k1`, `k2`, and `k3` to `g1`.

Since the set of workspaces represents a unique logical computation, the Workspace Manager must deal with the case in which a task in a given workspace needs to operate on resources generated by tasks in previous workspaces. Such resources are not physically generated when the user starts composing a subsequent workspace of the same computation because all the workspaces are processed for the execution only at the end of the design session. The Workspace Manager recognizes such a situation during the composition of a workspace, generates the needed *virtual* resources, and makes them available, through the Resource Manager, to the subsequent workspaces. For instance, in *workspace 1* (Fig. 2.10), the data set `Unidb` is copied to the node `k1.deis.unical.it`; therefore, a new metadata document is created for `Unidb` and stored in the directory `k1.deis.unical.it` of the TMR. The document specifying the new location of `Unidb` is marked as temporary until the data transfer is performed. However, in *workspace 2* (Fig. 2.11), the data set `Unidb` is displayed as already available under the resources of `k1.deis.unical.it`.

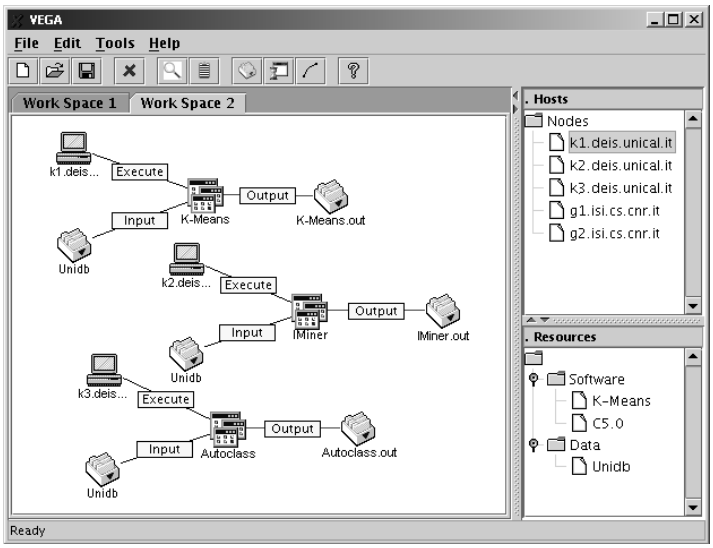

Fig. 2.11. VEGA: Example workspace 2

2.5.2 Task Consistency Checking

The goal of this phase is to obtain a correct and consistent model of the computation. The validation process is performed by means of two components: the model preprocessor the model postprocessor.

The preprocessing of the computation model takes place during the graphical composition. The model preprocessor verifies the composition consistency, allowing the user, with a context-sensitive control, to create links only if they represent actions that can really be executed. For instance, given a data set, it allows the user to link it to a software through an input/output link, not to a host through an execution link.

The checking is completed by the model postprocessor, which is responsible for catching error occurrences that cannot be recognized during the preprocessing phase. For example, it indicates accordingly if the graphical composition in a workspace does not contain at least one host.

2.5.3 Execution Plan Generation

In this phase, the computation model is translated into an execution plan represented by an XML document. This task is performed by the *Execution Plan Generator*.

Basically, the Execution Plan Generator is a parser that analyzes the computation model produced during the graphical composition, and is able to generate its equivalent XML representation, taking into account the properties of the involved resources and the parameters of the links. The XML

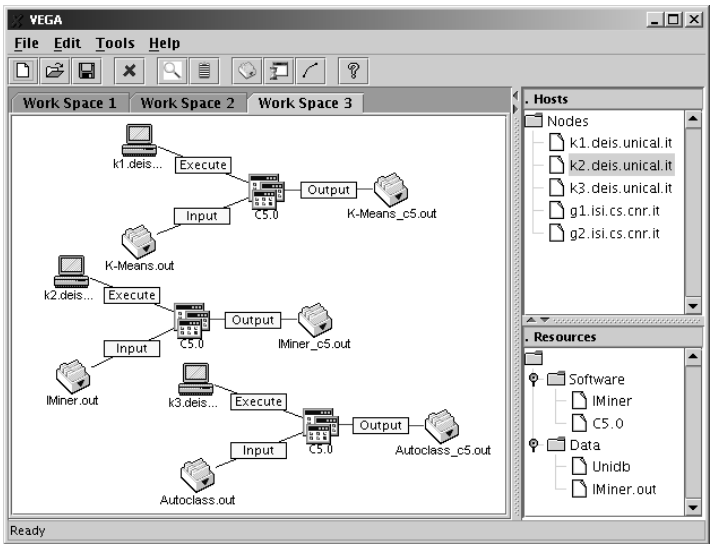

Fig. 2.12. VEGA: Example workspace 3

execution plan describes a data mining computation at a high level, containing physical information neither about resources (which are identified by metadata references) nor about the status and current availability of such resources. In fact, specific information about the resources involved will be included in the RSL generation phase, when the computation model is translated into this language. Figure 2.14 shows an extract of the execution plan for the example described above. The execution plan gives a list of tasks and task links, which are specified using the XML tags `Task` and `TaskLink`, respectively. The `label` attribute of a `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 task represented. For instance, the task identified by the `ws1_dt2` label contains a `DataTransfer` element, indicating that it is a data transfer task. The `DataTransfer` element specifies `Protocol`, `Source`, and `Destination` of the transfer. The `href` attributes of such elements specify the location of metadata about protocol, source, and destination objects. In this example, metadata about the source of the data transfer in the `ws1_dt2` task are provided by the `Unidb.xml` file stored in the directory named `g1.isi.cs.cnr.it` of the TMR, whereas metadata about the destination are provided by the `Unidb.xml` file stored in the directory named `k2.deis.unical.it` of the same TMR. The first of such XML documents provides metadata about the `Unidb` data set when stored on `g1.isi.cs.cnr.it`, whereas the second one provides metadata about `Unidb` when, after the data transfer, it is stored on `k2.deis.unical.it`. The `TaskLink` elements represent

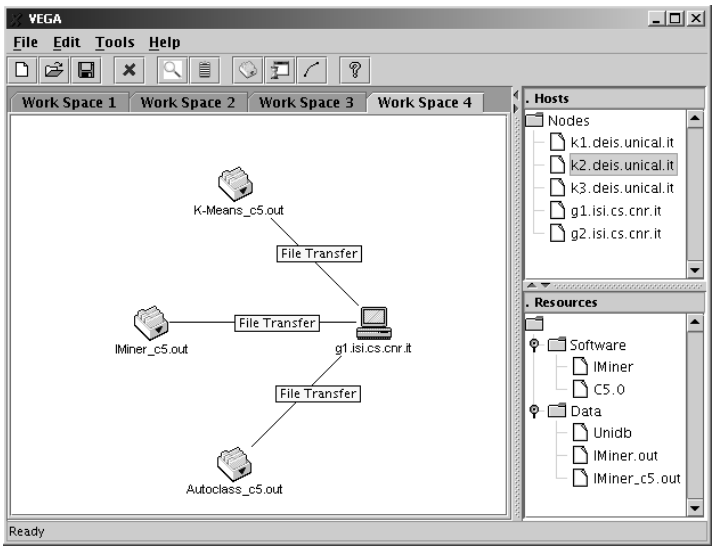

Fig. 2.13. VEGA: Example workspace 4

```
<ExecutionPlan>
 ...
 <Task ep:label="ws1_dt2">
  <DataTransfer>
   <Protocol ep:href="g1../GridFTP.xml"
           ep:title="GridFTP on g1.isi.cs.cnr.it"/>
   <Source ep:href="g1../Unidb.xml"
           ep:title="Unidb on g1.isi.cs.cnr.it"/>
   <Destination ep:href="k2../Unidb.xml"
                ep:title="Unidb on k2.deis.unical.it"/>
  </DataTransfer>
 </Task>
 ...
 <Task ep:label="ws2_c2">
  <Execution>
   <Program ep:href="k2../IMiner.xml"
            ep:title="IMiner on k2.deis.unical.it"/>
   <Input ep:href="k2../Unidb.xml"
          ep:title="Unidb on k2.deis.unical.it"/>
   ...
   <Output ep:href="k2../IMiner.out.xml"
           ep:title="IMiner.out on k2.deis.unical.it"/>
  </Execution>
 </Task>
 ...
 <TaskLink ep:from="ws1_dt2" ep:to="ws2_c2"/>
 ...
</ExecutionPlan>
```

Fig. 2.14. The extract of an execution plan

the relationships among the tasks of the execution plan. For instance, the `TaskLink` shown indicates that the `ws2_c2` task follows `ws1_dt2`, as specified by its `from` and `to` attributes.

As stated in Sect. 2.4, execution plans are abstract if some of the resources are not completely determined. They are specified by means of one or more constraints. In this case, the Execution Plan Generator produces an abstract execution plan, i.e., an XML document similar to the example shown in Fig. 2.14, except that at least one resource is an abstract resource. For example, the XML `Program` element within the description of the computation task `ws2_c2`, in Fig. 2.14, could contain a reference to an abstract resource which specifies the features of a data mining software able to perform the requested type of analysis.

2.6 Execution of a PDKD Computation

Figure 2.15 shows the steps of a PDKD computation execution. The execution plan optimization and translation are performed by the RAEMS, whose basic functionalities are provided by the VEGA components (see Fig. 2.9).

In the optimization phase, the RAEMS scheduler searches the *Knowledge Grid* for concrete resources that satisfy the constraints specified in the corresponding abstract resources. The searching is performed by means of the KDS service. Once a proper number of candidate resources have been found, the RAEMS selects those resources that allow the generation of the optimal execution plan.

Currently, VEGA integrates an *RSL Generator* module, which produces an RSL script that can be directly submitted to the *Globus Resource Allocation Manager* (*GRAM*) of a Grid node running Globus. The RSL (Resource Specification Language) is a structured language with which resource requirements and parameters can be outlined by a user [2.23]. In contrast with the XML execution plan, the RSL script entirely describes an instance of the designed computation, i.e., it specifies all the physical information needed for the execution (e.g., name and location of resources, software parameters, etc.). Figure 2.16 shows an extract of a sample RSL script.

The execution of the computation is performed by means of the VEGA *Execution Manager* module. The Execution Manager allows the system to authenticate a user to the Grid, using the Globus GSI (Grid Security Infrastructure) services, and to submit the RSL script to the Globus GRAM for its execution. The Execution Manager is also responsible for the monitoring of the jobs that compose the overall data mining computation during their life cycle. Finally, the Execution Manager collects results of the PDKD computation and presents them to the user.

2.7 Conclusions

Grid computing is enlarging its scope and is going to be more and more complete and complex both in the number of developed tools and in the va-

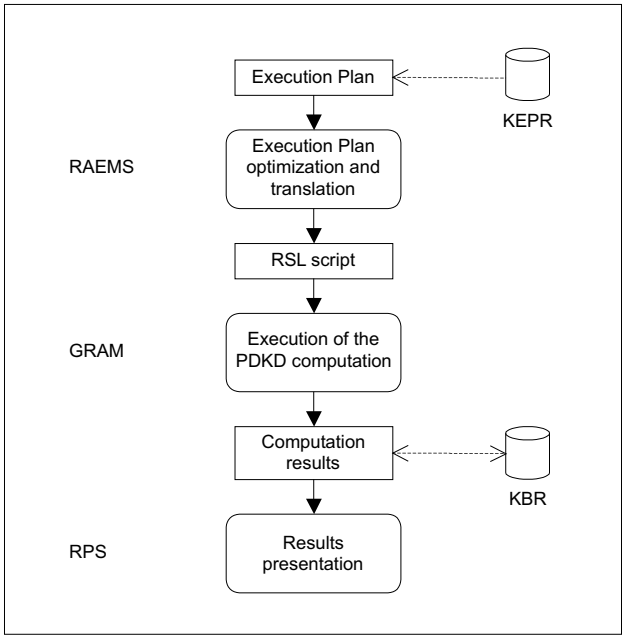

Fig. 2.15. The PDKD computation execution process

```
+
...
(&(resourceManagerContact=g1.isi.cs.cnr.it)
  (subjobStartType=strict-barrier)
  (label=ws1_dt2)
  (executable=$(GLOBUS_LOCATION)/bin/globus-url-copy)
  (arguments=-vb -notpt gsiftp://g1.isi.cs.cnr.it/.../Unidb
             gsiftp://k2.deis.unical.it/.../Unidb
  )
)
...
(&(resourceManagerContact=k2.deis.unical.it)
  (subjobStartType=strict-barrier)
  (label=ws2_c2)
  (executable=.../IMiner)
  ...
  )
)
...
```

Fig. 2.16. The extract of a sample RSL script

riety of supported applications. Deployed Grids are growing up very quickly and support high-performance applications in science, industry, and business. According to this trend, Grid services are shifting from generic computation-oriented services to high-level information management and knowledge discovery services. It is vitally important to design and develop knowledge-based systems for supporting sophisticated Grid applications. The *Knowledge Grid* system we discussed here is a significant system that, according to this approach, aims to support the implementation of knowledge discovery processes

on Grids. 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 [2.6].

In this chapter we also presented the *Knowledge Grid* features and tools by showing how a user through a step-by-step process can compose and execute a knowledge discovery application on a Grid in a simple way. Besides completing the *Knowledge Grid* implementation we are developing real distributed data mining applications in different domains that exploit the system features and give us feedback on the user needs in terms of capabilities and performance. These experiments will help us to improve and extend the system functionalities.

Acknowledgements

This work has been partially funded by the project MIUR Fondo Speciale SP3: Grid Computing: Tecnologie abilitanti e applicazioni per eScience.

References

2.1 I. Foster, C. Kesselman, S. Tuecke: The Anatomy of the Grid: Enabling Scalable Virtual Organizations. Int. J. of Supercomputing Applications, 15(3) (2001)
2.2 W. Hoschek, J.J. Martinez, A. Samar, H. Stockinger, K. Stockinger: Data Management in an International Data Grid Project. *Proc. IEEE/ACM Int. Workshop on Grid Computing, Grid 2000* (LNCS Vol. 1971, Springer Verlag) pp. 77-90
2.3 P. Avery, I. Foster: GriPhyN Project Description. Available at http://www.griphyn.org/info/index.html
2.4 Y. Morita et al.: Grid Data Farm for Atlas Simulation Data Challenges. *Proc. of Int. Conf. on Computing of High Energy and Nuclear Physics, 2001* pp. 699-701
2.5 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 Data sets. Journal of Network and Computer Applications, 23, 187-200 (2001)
2.6 F. Berman: From TeraGrid to Knowledge Grid. Communications of the ACM, 44(11), 27-28 (2001)
2.7 W.E. Johnston: Computational and Data Grids in Large-Scale Science and Engineering. Future Generation Computer Systems, 18 (8), 1085-1100 (2002)
2.8 M. Cannataro, D. Talia, P. Trunfio: Knowledge Grid: High Performance Knowledge Discovery Services on the Grid. *Proc. GRID 2001* (Springer-Verlag, 2001) pp. 38-50
2.9 M. Cannataro, A. Congiusta, D. Talia, P. Trunfio: A Data Mining Toolset for Distributed High-performance Platforms. *Proc. Conf. Data Mining 2002* (Wessex Inst. Press, Bologna, Italy, 2002)
2.10 H. Kargupta, P. Chan (eds.): *Advances in Distributed and Parallel Knowledge Discovery* (AAAI/MIT Press, 2000)

2.11 C. Catlett: The TeraGrid: a Primer. Available at http://www.teragrid.org/
2.12 F. Berman: Private communication (November 2001)
2.13 N. Giannadakis, A. Rowe, M. Ghanem, Y. Guo: InfoGrid: Providing Information Integration for Knowledge Discovery. To Appear in the Journal of Information Science
2.14 The DataCutter project: http://www.cs.umd.edu/projects/hpsl/chaos/ResearchAreas/dc/
2.15 D. Skillicorn, D. Talia: Mining Large Data Sets on Grids: Issues and Prospects. Computing and Informatics, 21, 347-362 (2002)
2.16 The ADaM system. http://datamining.itsc.uah.edu/adam/.
2.17 T. Hinke, J. Novonty: Data Mining on NASA's Information Power Grid. *Proc. Ninth IEEE Int. Symposium on High Performance Distributed Computing, 2000*
2.18 Discovery Net. http://ex.doc.ic.ac.uk/new/
2.19 V. Curcin, M. Ghanem, Y. Guo, M. Kohler, A. Rowe, J. Syed, P. Wendel: Discovery Net: Towards a Grid of Knowledge Discovery. *Proc. Eighth ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining* (Edmonton, Canada, 2002)
2.20 Testbeds, National Center for Data Mining, Laboratory for Advanced Computing, University of Illinois at Chicago. http://www.ncdm.uic.edu/testbeds.htm
2.21 I. Foster: Building the Grid: An Integrated Services and Toolkit Architecture for Next Generation Networked Applications. Technical Report, available at http://www.gridforum.org/building_the_grid.htm (2000)
2.22 I. Foster, C. Kesselman: Globus: a metacomputing infrastructure toolkit. Int. J. of Supercomputing Applications, 11, pp. 115-128 (1997)
2.23 The Globus Project. The Globus Resource Specification Language RSL v1.0. Available at http://www.globus.org/gram/rsl_spec1.html
2.24 C. Mastroianni, D. Talia, P. Trunfio: Managing Heterogeneous Resources in Data Mining Applications on Grids Using XML-based Metadata. To appear on *Heterogeneous Computing Workshop* (HCW 2003, Nice, France, April 2003)
2.25 M.S. Chen, J. Han, P.S. Yu: Data Mining: An Overview from a Database Perspective. IEEE Trans. Knowledge and Data Engineering, 8(6), 866-883 (1996)
2.26 R.L. Grossman, M.F. Hornick, G. Meyer: Data Mining Standard Initiatives. Communications of the ACM, 45(8) (August 2002)
2.27 J. MacQueen: Some Methods for Classification and Analysis of Multivariate Observations. *Proc. 5th Symp. on Mathematical Statistics and Probability, 1967*, pp. 281-297
2.28 IBM Intelligent Miner. http://www.software.ibm.com/data/iminer/
2.29 P. Cheeseman, J. Stutz: Bayesian Classification (AutoClass): Theory and Results. In: U.M. Fayyad, G.P. Shapiro, P. Smyth, R. Uthurusamy (eds.), *Advances in Knowledge Discovery and Data Mining* (AAAI Press/MIT Press, 1996) pp. 61-83
2.30 J.R. Quinlan: See5/C5.0, version 1.16. http://www.rulequest.com/see5-info.html (2002)