Metadata for Managing Grid Resources in Data Mining Applications

Carlo Mastroianni[1], Domenico Talia[1,2] and Paolo Trunfio[2]
[1]*ICAR-CNR, Via P. Bucci, cubo 41-c, 87036 Rende (CS), Italy*
E-mail: mastroianni@icar.cnr.it
[2]*DEIS, Università della Calabria, Via P. Bucci, cubo 41-c, 87036 Rende (CS), Italy*
E-mail: {talia,trunfio}@deis.unical.it

Key words: data mining, discovery service, dynamic scheduling, knowledge grid, metadata management, peer-to-peer, resource categorization, semantic grid

Abstract

The Grid is an infrastructure for resource sharing and coordinated use of those resources in dynamic heterogeneous distributed environments. The effective use of a Grid requires the definition of metadata for managing the heterogeneity of involved resources that include computers, data, network facilities, and software tools provided by different organizations. Metadata management becomes a key issue when complex applications, such as data-intensive simulations and data mining applications, are executed on a Grid. This paper discusses metadata models for heterogeneous resource management in Grid-based data mining applications. In particular, it discusses how resources are represented and managed in the KNOWLEDGE GRID, a framework for Grid-enabled distributed data mining. The paper illustrates how XML-based metadata is used to describe data mining tools, data sources, mining models, and execution plans, and how metadata is used for the design and execution of distributed knowledge discovery applications on Grids.

1. Introduction

The Grid allows sharing and coordinated use of resources in dynamic geographically distributed environments. The effective use of a Grid requires the definition of a model to manage the heterogeneity of the involved resources that can include computers, data, network facilities, sensors, and software tools generally provided by different organizations. Heterogeneity in Grids arises mainly from the large variety of resources available within each category. For example, a software package 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 the use of metadata, whose purpose is to provide information about the features of resources and their effective use. A Grid user needs to know which resources are available, where resources can be found, how resources can be accessed and when resources are available. Metadata can provide answers about involved computing resources such as data repositories (e.g., databases, file systems, web sites), machines, networks, programs, documents, user agents, etc. Therefore, metadata can represent a key element for the effective discovery and utilization of resources on the Grid.

The role of metadata for resource management on Grids is more and more important as Grid applications become more and more complex. Thus, Grids mechanisms and models that define rich metadata schemas able to represent the variety of involved resources are a key element.

This paper discusses heterogeneous resource management in Grid-based data mining applications. We address the problem of locating and allocating computational, data and information resources, and other operations required to use data mining resources in knowledge discovery processes on Grids. In particular, the paper presents an XML-based approach for managing heterogeneous resources in the KNOWLEDGE

GRID environment [1]. The work presented in this paper is an extended version of a work presented in [2]. This new version includes a more detailed description of the proposed system, describes some extensions, and presents its implementation.

The KNOWLEDGE GRID architecture uses basic Grid services and defines a set of additional layers, to support distributed knowledge discovery on world wide connected computers, where each node can be a sequential or a parallel machine. The KNOWLEDGE GRID architecture (see Figure 1) is designed on top of mechanisms provided by Grid environments such as Globus [3]. 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 (PDKD) tools and services.

The KNOWLEDGE GRID services are organized into two layers: the *Core K-Grid* layer, which is built on top of generic Grid services, and the *High level K-Grid* layer, which is implemented over the core layer. The Core K-Grid layer comprises two basic services: the *Knowledge Directory Service* (*KDS*) and the *Resources Allocation and Execution Management Service* (*RAEMS*). The KDS manages metadata describing the characteristics of relevant objects for PDKD applications, such as data sources, data mining software, results of computations, tools for manipulating data and results, execution plans, etc. The information managed by the KDS is stored into three repositories: metadata describing features of data, software and tools expressed in XML documents, is stored in a *Knowledge Metadata Repository* (*KMR*), information about the knowledge discovered after a PDKD computation is stored in a *Knowledge Base Repository* (*KBR*), whereas the *Knowledge Execution Plan Repository* (*KEPR*) stores the execution plans describing PDKD applications over the Grid. The goal of RAEMS is to find a mapping between an execution plan and the available Grid resources that satisfies user, data and algorithms requirements and constraints.

The High level K-Grid layer comprises the services used to build and execute PDKD computations over the Grid. The *Data Access Service* (*DAS*) is used for the search, selection, extraction, transformation and delivery of data to be mined. The *Tools and Algorithms Access Service* (*TAAS*) is responsible for search, selection, and download of data mining tools and algorithms. The *Execution Plan Management Service* (*EPMS*) is used to generate a set of different possible execution plans, starting from the data and the programs selected by the user. Execution plans are stored in the KEPR to allow the implementation of iterative knowledge discovery processes, e.g., periodical analysis of dynamic data sources. The *Results Presentation Service* (*RPS*) specifies how to generate, present and visualize the PDKD results (rules, associations, models, classification, etc.), and offers methods to store those results in the KBR in different formats.

By using services, tools, and repositories provided by the two layers of the KNOWLEDGE GRID, a user can search and identify data sources, data mining tools, and computational resources. Then she/he can combine all these components to build a distributed/parallel data mining application that can be executed on a Grid.

The remainder of the paper discusses how resources are represented and managed in the KNOWLEDGE GRID, how XML-based metadata is used to define data mining tools, data sources, mining models and execution plans, and how metadata is used to design and execute distributed data mining applications on Grids. The paper is organized as follows. Section 2 discusses the management of Grid resources. Section 3 describes the representation of resource metadata. Section 4 discusses the representation of execution plans. Section 5 describes the implementation of the Knowledge Directory Service and a peer-to-peer approach to metadata management. Section 6 discusses related work and Section 7 concludes the paper.

2. Management of the Resources in the KNOWLEDGE GRID

Several approaches have been investigated to represent and manage metadata in Grid environments. Some of them have been proposed in systems such as Globus [3], DICE [4], and the NASA's Information Power Grid [5]. In particular, in the Globus Toolkit 2 – on which the current implementation of the KNOWLEDGE GRID is based – the *Monitoring and Discovery Service* (*MDS*) provides information about the status of the system components [6]. The MDS uses the *Lightweight Directory Access Protocol* (*LDAP*) [7] as a uniform interface to such information. MDS includes a configurable information provider called *Grid Resource Information Service* (*GRIS*) and a configurable aggregate directory service called *Grid Index Information Service* (*GIIS*). A GRIS can answer queries about the resources of a particular Grid node. Examples of information provided by this service include host identity (e.g., operating systems and versions), as well as more dynamic information such as CPU and memory availability. A GIIS combines the information provided by a set of GRIS services managed by an organization, giving a coherent system image that can be explored or searched by Grid applications.

The KNOWLEDGE GRID manages typical resources involved in distributed data mining computations such as:

- 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, that is 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.

This large set of different resources – that in some cases require a complex description – motivated the definition of a metadata model that extends the basic Globus model.

The basic objectives that guided us through the definition of the resource metadata are:

- Metadata should document in a simple and human-readable fashion the features of a data mining application.
- Metadata should allow for the effective search of resources.
- Metadata should provide an efficient way to access resources.
- Metadata should be used by software tools that support a user in building KNOWLEDGE GRID applications. For example, VEGA [8] is a visual toolset for designing and executing data mining applications over the KNOWLEDGE GRID that uses metadata to describe and show the available resources.

The current KNOWLEDGE GRID implementation uses the Globus 2 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.). As mentioned before, the complexity of the information associated to more specific KNOWLEDGE GRID resources (data sources, mining algorithms, knowledge models) led us to design a different model to represent and manage the corresponding metadata.

We adopted the *eXtensible Markup Language* (*XML*), that provides a set of functionalities and capabilities that are making it a common emerging model for describing data structure and data set frameworks:

- XML provides a way to define infrastructure independent representations for information.
- XML allows a user to define complex data structures: for example the XML Schema formalism [9] provides a means for defining a strong control on simple and complex data types in XML documents.
- XML allows for the use of powerful query languages: for instance, the XML Query [10] provides SQL-like query facilities to extract data from real and virtual documents on the Web.
- It is easy to map XML documents into the data structures of an object-oriented programming language: for example, the Xerces library [11] performs the parsing of an XML document in a Java or C++ environment.

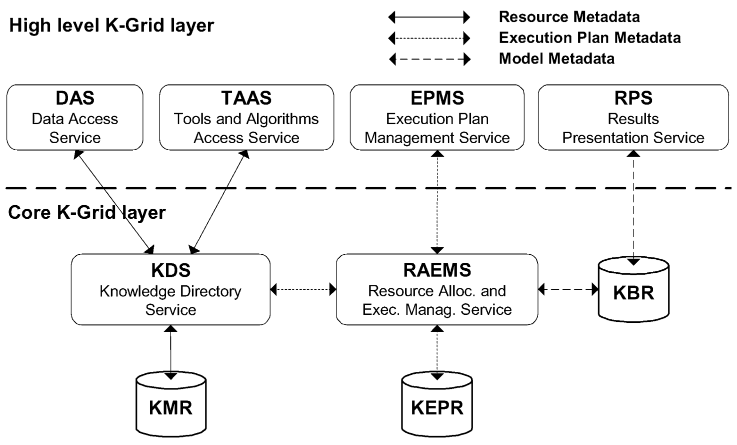
On the basis of these features, we decided to represent metadata by XML documents according to a set of XML schemas defined for the different classes of resources, as we discuss in the next section.

It is worth noting that by using XML for metadata definition we benefit from a standard language that makes our model flexible and extensible. Furthermore, the resulting metadata model could be used to describe other advanced Grid applications.

In the KNOWLEDGE GRID, metadata is accessed and managed by means of a set of services. In particular, the KNOWLEDGE GRID architecture defines, as mentioned above, the KDS, which maintains metadata and allows applications to query and manage it. The information managed by the KDS is stored into three repositories (see Figure 1):

(1) the *Knowledge Metadata Repository* (*KMR*) that stores metadata describing features of data sources, software, and tools;
(2) the *Knowledge Base Repository* (*KBR*) that stores information about the discovered models, and
(3) the *Knowledge Execution Plan Repository* (*KEPR*) that stores the execution plans describing distributed data mining applications over the Grid.

The KNOWLEDGE GRID DAS and TAAS services make use of the KDS for, respectively:



High level K-Grid layer
Resource Metadata
Execution Plan Metadata
Model Metadata
DAS Data Access Service
TAAS Tools and Algorithms Access Service
EPMS Execution Plan Management Service
RPS Results Presentation Service
Core K-Grid layer
KDS Knowledge Directory Service
RAEMS Resource Alloc. and Exec. Manag. Service
KBR
KMR
KEPR

Figure 1. The KNOWLEDGE GRID architecture.

- search, selection (data search services), extraction, transformation and delivery (data extraction services) of data to be mined and
- search, selection and downloading of data mining tools and algorithms.

Figure 1 shows the main metadata flows among KNOWLEDGE GRID services and repositories. The high level DAS and TAAS services use the core level KDS service to manage metadata about algorithm and data sources. In turn, the KDS interacts with the KMR repository to access and store such metadata. The EPMS service manages execution plan metadata, which is accessed through the core level RAEMS service, and stored in the KEPR database. Furthermore, execution results and related model metadata are stored in the KBR repository, and processed by the RPS service.

The metadata management process is a key aspect in the development of data mining applications over the KNOWLEDGE GRID, and on the whole development process of complex applications on Grids. A typical life cycle of metadata consists of the following steps:

(1) Resource metadata is published in the KMRs of the corresponding nodes.
(2) A user specifies the features of the resources she/he needs to design a data mining application.
(3) The DAS and TAAS services search the KMRs of the KNOWLEDGE GRID nodes for the requested resources.
(4) Metadata describing the resources of interest is delivered by such services to the requesting user.
(5) Metadata related to software, data, and operations is combined into an execution plan (see Section 4) to design a complete data mining application.
(6) After application execution, results are stored into the KBR; metadata related to new and/or modified resources is published in the respective KMRs for future use.

3. Resource Metadata Representation

As mentioned before, our goal is to define metadata needed to identify and classify all the heterogeneous resources used in data mining applications on Grids. The first step in designing metadata is the categorization of resources. In the current implementation of the KNOWLEDGE GRID, we focused on the definition of metadata related to data sources, data mining tools, and discovered knowledge. In Subsections 3.1–3.3 a sample metadata document is reported for each resource type. The sample documents are extracted from a real data mining application based on the AutoClass clustering tool [12].

In this section we address the problem of giving metadata a proper structure (schema) that allows for properly representing information about the different types of resources. However, the definition of metadata structure is not sufficient to solve the problem of giving the right meaning to metadata information. To allow applications to automatically manage information represented in metadata documents, it is also necessary to associate semantics to them. This is usually accomplished through the use of ontologies. This issue is discussed in Subsection 3.4.

3.1. *Data Mining Software*

We categorized data mining software on the basis of the following parameters [13]:

Table 1. Classification of data mining software.

Classification parameter	XML tag	Possible values
Kind of data sources	`<KindOfData>`	Relational database, transaction database, object-oriented database, deductive database, spatial database, temporal database, multimedia database, heterogeneous database, active database, legacy database, semi-structured data, flat file.
Kind of knowledge to be mined	`<KindOfKnowledge>`	Association rules, clusters, characteristic rules, classification rules, sequence discovery, discriminant rules, evolution analysis, deviation analysis, outlier detection, regression.
Type of techniques to be utilized	`<KindOfTechnique>`	Statistics, decision trees, neural networks, genetic algorithms, Apriori, fuzzy logic, SVD, bayesian networks, nearest neighbors. . .
Driving method	`<DrivingMethod>`	Autonomous knowledge miner, data-driven miner, query-driven miner, interactive data miner.

- the kind of input data sources;
- the kind of knowledge that is to be discovered;
- the type of techniques that data mining software tools use in the mining process;
- the driving method of the mining process.

Table 1 summarizes a set of values for each classification parameter. This table is mapped on an XML schema which defines the format and the syntax of the XML file that will be used to describe the features of a generic data mining software. The second column of Table 1 reports the XML elements that correspond to the classification parameters.

As an example, Figure 2 reports XML metadata related to the data mining software AutoClass [12]. The XML file is composed of two parts. The first part is the software `Description`, the second one is the software `Usage`. The `Description` section specifies one or more values, among those reported in Table 1, for each classification parameter. The `Usage` section contains information that can be used by a client to access and use the software package. This section is composed of a set of subsections, among which `Syntax`, `Hostname`, `ManualPath`, and `DocumentationURL`. The Syntax subsection describes the format of the command that a client should use to invoke the AutoClass software tool. 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 children of the root node specify the arguments that 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 is a textual description of the argument; the `type` attribute specifies if 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 Figure 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`

3.2. *Data Sources*

Data sources are analyzed by data mining algorithms to extract knowledge from them [13]. They can originate from relational databases, plain files, Web pages 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 the access and search operations on such resources.

The common structure of metadata is composed of two parts:

- an `Access` section that includes information for retrieving a data source;
- a `Structure` section that provides information about the logical and/or physical structure of a data source.

As an example, Figure 3 shows an XML metadata document for a flat file that can be used as an

```
<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>
```

Figure 2. An extract from an XML metadata sample for the AutoClass software.

input by the AutoClass software. The `Access` section includes file system information, e.g., the `Location` and the `Size` of the file, etc. 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.).

Although the high-level XML metadata format is the same for all kinds of data sources, the content of the `Access` and `Structure` sections may depend on the specific characteristics of a given data source. As an example, for relational databases the `Format` subsection is no more needed, since the physical formatting is managed by the database management system. Furthermore, new subsections should be defined; for instance, in the `Access` section, information should be provided for the connection to the database (e.g., the ODBC specifications).

3.3. *Data Mining Models*

The knowledge discovered through a data mining process is represented by “data mining models.” Whereas till today no common language has been proposed for the definition of the data mining resources

```
<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>
```

Figure 3. An extract from an XML metadata sample for a flat file.

discussed before, a standard framework, 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 [14]. PMML provides a *Document Type Definition* (*DTD*) to describe different kinds of models such as classification rules and association rules. We used it to define data mining models in the KNOWLEDGE GRID. As an example, Figure 4 shows an extract from a PMML document that represents the clustering model extracted by AutoClass from the dataset whose metadata is reported in Figure 3. In this example, AutoClass performs a clustering task on records concerning car imports in 1985.

The `MiningSchema` element of the model reported in Figure 4 points out that the clustering process is based on three record attributes: *make*, *num-of-doors* and *body-style*. For the sake of shortness, two clusters (out of 12) are partially shown, the former composed of 28 records and the latter composed of 4 records.

Notice that each cluster record can be reconstructed by taking the values located in the same position within each clustering field. For example, *<bmw two sedan>* is the first record belonging to *Cluster 1*. The portion of the PMML DTD that we used for this clustering model is available at [15].

3.4. *Semantics in the Knowledge Grid*

In the Grid computing community there is an effort to define the so called *Semantic Grid* [16], whose approach is based on the systematic description of resources through metadata and ontologies. The use of ontologies in Grid applications integrates XML-based metadata information models by associating semantics

```
<PMML version = “2.0”>
  ...
    <ClusteringModel modelName = “Clustering on imports-85c”
                     modelClass = “distributionBased” numberOfClusters = “12”>
      <MiningSchema>
        <MiningField name = “make”/>
        <MiningField name = “num-of-doors”/>
        <MiningField name = “body-style”/>
      </MiningSchema>
      ...
      <Cluster name = “Cluster 1”>
        <Partition name = “Partition 1”>
          <PartitionFieldStats field = “make”>
            <Array n = “28” type = “string”>bmw bmw jaguar nissan ...</Array>
          </PartitionFieldStats>
          <PartitionFieldStats field = “num-of-doors”>
            <Array n = “28” type = “string”>two four four four ...</Array>
          </PartitionFieldStats>
          <PartitionFieldStats field = “body-style”>
            <Array n = “28” type = “string”>sedan sedan sedan wagon ...</Array>
          </PartitionFieldStats>
        </Partition>
      </Cluster>
      <Cluster name = “Cluster 2”>
        <Partition name = “Partition 2”>
          <PartitionFieldStats field = “make”>
            <Array n = “4” type = “string”>chevrolet chevrolet chevrolet dodge</Array>
          </PartitionFieldStats>
          ...
        </Partition>
      </Cluster>
      ...
    </ClusteringModel>
</PMML>
```

Figure 4. An extract from a PMML model file.

to them. The Semantic Grid approach exploits the use of

- services for querying over metadata and ontologies,
- tools for knowledge extraction and reasoning, and
- semantic search engines.

Those services and tools represent an evolution with respect to current basic Grid services, such as the Globus MDS pattern-matching based search.

Within the KNOWLEDGE GRID project, an effort is on the way to extend the architecture with ontology-based services. In [17], Cannataro and Comito propose an ontology for the data mining domain (DAMON) to enhance the design of distributed data mining applications on the KNOWLEDGE GRID. DAMON offers a reference model for the different kinds of data mining tasks, methodologies, and software available to solve a given task.

To exploit the DAMON capabilities, some novel components are introduced in the KNOWLEDGE GRID architecture. In particular, the *Ontology Directory Service* (*ODS*) is responsible for maintaining the ontological data and allows applications to query and manage them. Ontological data is represented by RDF schema files and is stored in an *Ontological Data Repository* (*ODR*), whereas metadata regarding availability, location, and configuration of resources is stored into the KMR.

According to this approach, the search and selection of the resources is split into two phases:

(1) Ontology-based selection of the resources. Browsing and searching the ontology allows a user to locate the more appropriate tasks, methods, algorithms and finally data mining software to be used in a certain phase of the knowledge discovery process.

(2) Access to the resource metadata. The ontology gives the URLs of all the instances of the selected resources available on the KNOWLEDGE GRID nodes, i.e. the URLs of the relevant metadata files stored in the KMRs.

4. Execution Plan Representation

A distributed data mining application is a process composed of several steps that are executed sequentially or in parallel. In the KNOWLEDGE GRID framework, the management of complex data mining processes is carried out through the definition of an *execution plan*. An execution plan is a graph that describes the interaction and data flow between data sources, data mining tools, visualization tools, and output models.

Starting from the XML representation of data mining resources, an execution plan defines the high-level logical structure of a data mining process. The KNOWLEDGE GRID provides a visual environment, called VEGA [8], which allows a user to build an execution plan in a semi-automatic way.

An execution plan may contain *concrete resources* and *abstract resources*. A concrete resource is completely specified by its metadata that has been previously retrieved from remote and local KMRs. In metadata describing an abstract resource some features are expressed as constraints. For instance, whereas the metadata described in Section 3 describes an instance of AutoClass, available on a given node, the metadata document shown below describes an abstract data mining software that is able to perform a clustering task on flat files.

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

An abstract resource can be instantiated into an existing concrete resource whose metadata matches the specified constraints. An execution plan that contains at least one abstract resource is an *abstract execution plan*, whereas an execution plan containing only concrete resources is referred to as an *instantiated execution plan*. 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 service attempts to transform it into an instantiated execution plan, by resolving abstract resources into concrete resources. Such an action is performed by a scheduler that allows the generation of the optimal 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.

For long-running applications, it could be useful to evaluate the instantiation of abstract resources not only before the application execution, but also during the execution itself. Indeed, a concrete resource (e.g., a host or a software) chosen for the execution of a task could become unavailable after the starting of the overall application but before the execution of that particular task. On the other hand, novel useful resources might become available, so they could be taken into account at run time for a more efficient execution of the application. We deal with this problem by using an approach based on two elements:

(1) An instantiated execution plan can provide a list of several concrete resources that properly match an abstract resource; the list is ordered so that the first concrete resource is to be considered the most convenient, and so forth. In this way, should a resource become unavailable before task execution, it could be immediately replaced without re-invoking the scheduler.

(2) A number of *resource checkpoints* can be inserted within an execution plan, either by the user or

by the designing tool: when a resource checkpoint is reached, the scheduler is invoked in order to reconsider the instantiation of the abstract resources comprised between that checkpoint and the successive one encountered in the execution plan. Two kinds of re-scheduling operations are defined: a "soft" rescheduling allows for checking the availability of the listed concrete resources and possibly for changing their ordering, without looking for new concrete resources. A "hard" rescheduling is used to search further concrete resources that may have become available during the execution of the application. "Hard" rescheduling provides a better resource selection, but it could be remarkably slower.

Figure 5 shows an extract from a sample instantiated execution plan. An execution plan gives a list of *tasks* and *task links*, which are specified using the XML tags `Task` and `TaskLink`, respectively. The `label` attribute in a `Task` element identifies one basic task in the execution plan, and it is used to link various basic tasks to compose the overall task flow. In general, each `Task` element contains three sub-elements: the `Program` element, which specifies the software to execute, and the `Input` and `Output` elements, that describe input and output files. The `href` attributes of such elements specify, respectively, the location of metadata related to executable, input, and output files. Note that a file transfer is managed exactly as a program execution, where the software to be invoked is a GridFTP client, or another available file transfer agent.

The shown execution plan specifies that the AutoClass software in `task6` should be run on the abstract host `abstract_host1`: it means that the corresponding concrete host, to be selected by the scheduler, must have a copy of the program AutoClass that satisfies the constraints specified in the related XML metadata. On the same host, data source, i.e. the input of AutoClass, has to be transferred using the GridFTP program, as described in the specification of `task2`.

In the execution plan, a checkpoint is denoted as a particular `Task` element having a sub-element `ResourceCheck` that indicates if the checkpoint requests a hard or a soft rescheduling.

A `TaskLink` element represents a relation between two tasks in an execution plan. For instance, the shown `TaskLink` elements indicate, as specified by their `from` and `to` attributes, that the task flow proceeds from `task1` to `task2` (with the checkpoint `check1` between them), then to `task3`, and so on.

Besides the value of the attribute `type` at the root element of the execution plan, the feature that distinguishes an instantiated execution plan from an abstract execution plan, is the presence of the element `ResourceInstantiation`. Within this element, for each abstract resource defined in the execution plan, a list of candidate concrete resources, selected by the scheduler, is given: e.g., Figure 5 shows two hosts that can be used to resolve the `abstract_host1` abstract resource. At the time of execution, the execution manager tries to access the specified concrete resources respecting their ordering. As mentioned before, concrete resources can be reordered at a "soft" resource checkpoint, while further concrete resources can be added within a "hard" checkpoint.

An instantiated execution plan will be translated into the language of a specific Grid resource broker for its execution. The KNOWLEDGE GRID implementation uses the *Globus Resource Allocation Manager* (*GRAM*) [18] whose script language is the *Resource Specification Language* (*RSL*) [19].

The instantiated execution plan is translated into a set of RSL scripts. Each RSL script corresponds to a segment of the instantiated execution plan included between two consecutive `ResourceCheck` tags.

Figure 6 shows an RSL script corresponding to the second segment of the sample execution plan in Figure 5. Such an RSL script includes two *resource descriptions* corresponding respectively to the tasks labeled as `task2` and `task6` in the instantiated execution plan. A typical resource description is composed of several attribute-value relationships in a conjunction.

For instance, the first description in this RSL script specifies that a data transfer is to be performed by using the `globus-url-copy` executable, located in the `minos.cs.icar.cnr.it` node, to copy the `imports-85c.db2` dataset from `minos.cs.icar.cnr.it` to `icarus.cs.icar.cnr.it`. Note that the shown RSL script is generated by substituting the `abstract_host1` of Figure 5 with the first concrete host in the related list of candidate resources.

To allow for the dynamic scheduling of the execution plan, an RSL script related to a given segment is generated after the concrete resources have been chosen at the corresponding `ResourceCheck`.

5. Metadata Management in the KNOWLEDGE GRID

Distributed metadata management in the KNOWLEDGE GRID is performed through the Knowledge

```
<ExecutionPlan type = “instantiated”>
  <Task label = “task1”>
    <Program href = “minos.cs.icar.cnr.it/software/DB2Extractor.xml”
             title = “DB2Extractor on minos.cs.icar.cnr.it”/>
    <Input href = “minos.cs.icar.cnr.it/data/car-imports_db2.xml”
           title = “car-imports.db2 on minos.cs.icar.cnr.it”/>
    <Output href = “minos.cs.icar.cnr.it/data/imports-85c_db2.xml”
            title = “imports-85c.db2 on minos.cs.icar.cnr.it”/>
  </Task>
  <Task label = “check1”>
    <ResourceCheck method = “soft”/>
  </Task>
  <Task label = “task2”>
    <Program href = “minos.cs.icar.cnr.it/software/GridFTP.xml”
             title = “GridFTP on minos.cs.icar.cnr.it”/>
    <Input href = “minos.cs.icar.cnr.it/data/imports-85c_db2.xml”
           title = “imports-85c.db2 on minos.cs.icar.cnr.it”/>
    <Output href = “abstract_host1/data/imports-85c_db2.xml”
            title = “imports-85c.db2 on abstract_host1”/>
  </Task>
  ...
  <Task label = “task6”>
    <Program href = “abstract_host1/software/autoclass3-3-3.xml”
             title = “autoclass on abstract_host1”/>
    <Input href = “abstract_host1/data/imports-85c_db2.xml”
           title = “imports-85c.db2 on abstract_host1”/>
    <Output href = ” abstract_host1/data/classes.xml”
            title = “classes on abstract_host1”/>
  </Task>
  ...
  <TaskLink ep:from = “task1” ep:to = “check1”/>
  <TaskLink ep:from = “check1” ep:to = “task2”/>
  <TaskLink ep:from = “task2” ep:to = “task3”/>
  ...
  <TaskLink ep:from = “task5” ep:to = “task6”/>
  ...
  <ResourceInstantiation abstractResource = “abstract_host1”>
    <candidateResource>icarus.cs.icar.cnr.it</candidateResource>
    <candidateResource>telesio.cs.icar.cnr.it</candidateResource>
  </ResourceInstantiation>
</ExecutionPlan>
```

Figure 5. An extract from an instantiated execution plan.

```
+
(&(resourceManagerContact = minos.cs.icar.cnr.it)
   (subjobStartType = strict-barrier)
   (label = task2)
   (executable = $(GLOBUS_LOCATION)/bin/globus-url-copy)
   (arguments = gsiftp://minos.cs.icar.cnr.it/.../imports-85c.db2
                gsiftp://icarus.cs.icar.cnr.it/.../imports-85c.db2
   )
)
...
(&(resourceManagerContact = icarus.cs.icar.cnr.it)
   (subjobStartType = strict-barrier)
   (label = task6)
   (executable = .../autoclass)
   (arguments = -search .../imports-85c.db2 .../imports-85c.hd2 .../imports-85c.model
                  ...
   )
)
...
```

Figure 6. An extract from an RSL script.

Directory Service (KDS) that allows users to publish and discover metadata about Grid resources. This section describes an implementation of the KDS based on the Globus Toolkit 2 (GT2). In particular, the KDS prototype uses the Monitoring and Discovery Service provided by GT2 which adopts the LDAP protocol and model to represent, publish, and query information.

The KDS implementation based on the MDS-2 offers the opportunity for an easy integration with the other KNOWLEDGE GRID components, which are in turn based on the GT2 services (GSI, GRAM, etc.). The MDS-2 arranges the information servers according to a hierarchical architecture. This approach is well suited for small-medium size Grids in which nodes are organized in a hierarchical fashion, but it requires that information servers belonging to the highest levels of the hierarchy are characterized by a high degree of availability.

In large-scale Grids characterized by a high degree of dynamicity, the MDS-2 architecture, and consequently the discussed KDS implementation, suffers of scalability and reliability problems that are typical in a client/server hierarchical architecture. To overcome such limitations, we are adopting a novel approach for the KDS architecture which is based on the peer-to-peer (P2P) paradigm.

Section 5.1 describes the current KDS implementation based on the GT2, whereas Section 5.2 outlines the envisioned evolution of the KDS architecture according to the peer-to-peer approach.

5.1. *Knowledge Directory Service Implementation*

The Knowledge Directory Service provides two classes of components:

(1) The *KDS publishing system*, which enables to create, modify, and delete XML documents in the local Knowledge Metadata Repository (KMR), and to publish documents stored in the KMR, so that they can be accessed by remote applications.
(2) The *KDS discovery system*, which enables to search, retrieve, and select XML metadata stored in the KMR of remote nodes.

The design of the KDS aims to achieve a clear separation between the metadata representation model (that uses XML) and the metadata access model (that currently uses LDAP), so that redesigning the access model would not affect the representation model.

The main components of the *KDS publishing system* are:

- the Knowledge Metadata Repository (KMR),
- the Grid Resource Information Service (GRIS), and
- the KDS provider.

Table 2. Attributes of the objectclass KGridMetadata.

Attribute	Description
`KGrid-Metadata-url`	URL of the XML document.
`KGrid-Metadata-content`	Content of the XML document, coded in the *base 64* notation.
`KGrid-Metadata-creation`	Creation time of the XML document.
`KGrid-Metadata-last-modified`	Last modification time of the XML document.
`KGrid-Metadata-size`	Dimension in byte of the XML document.

The KMR is implemented as a directory of the file system in which metadata about resources is stored as XML files. Each metadata document is characterized by an URL having the following format:

kds://<*hostname*>/<relative path of the document in the KMR of *hostname*>

As discussed in the previous sections, the KNOWLEDGE GRID provides a set of XML schemas that allow for the description of different types of resources. An XML editor is used to perform the creation, modification, and deletion of XML documents in the KMR, ensuring that such documents are valid with respect to the corresponding XML schemas.

The GRIS is the MDS component used to publish XML metadata on the Grid. As mentioned in Section 2, each Globus node executes a local GRIS which provides information about resources of that node by means of the LDAP protocol [6]. By default, a Globus node owns a set of *information providers* used to generate static and dynamic information about the resources of that node, such as operating system, memory size, CPU speed, file system size, system load, etc., which is published by the GRIS. An information provider is a program that, when invoked by the GRIS, returns a sequence of records that are published through an LDAP server. The records returned by the information providers are expressed in the LDIF format [20], and must respect the *LDAP schema* [7] stored in the GRIS. LDAP data is published in a hierarchical structure named *Directory Information Tree* (*DIT*).

Globus allows for the creation of customized information providers for publishing data through the GRIS [21]. The KDS defines an LDAP schema (the *KDS schema*) and an information provider (the *KDS provider*) to make KMR information accessible by remote applications. The KDS schema specifies how the information in the KMR is represented in the DIT maintained by the GRIS. For each XML document stored in the KMR, a corresponding entry is published in the DIT. To represent the structure of such entries, the KDS schema defines the objectclass `KGridMetadata`, which specifies the attributes described in Table 2.

Figure 7 shows the main steps through which the KDS publishing system replies to a search request. When the GRIS receives an LDAP search request from a remote application (step 1), it invokes the KDS provider (step 2). The KDS provider reads the XML files stored in the KMR directory (step 3) and, for each XML file, returns an LDIF entry that represents the information related to that file using the attributes described in Table 2 (step 4). The LDIF records are validated against the KDS schema (step 5), and are returned to the requesting client (step 6). Steps 2–5 can be skipped if the results of the search operation can be retrieved from the GRIS cache.

As mentioned in Section 2, the information of a set of GRIS can be collected and provided by a GIIS index server. Hence, it is possible to query a single GIIS to search metadata stored in a set of GRIS registered to it.

The KDS discovery system allows for searching, selecting, and retrieving the XML metadata stored in the KMRs of the KNOWLEDGE GRID nodes. The KMR metadata selected as a result of a KDS search is stored in the *Task Metadata Repository* (*TMR*), a local directory that caches metadata needed to compose an application.

The KDS discovery system provides a simple Java library that includes:

- the `KDSRetriever` class, and
- the `KDSSelector` interface.

Figure 8 shows how the KDS discovery system is used by a client application to search metadata in remote KMRs. The `KDSRetriever` class allows for retrieving and selecting metadata stored in the KMRs of remote nodes, and for saving them in the local TMR. A `KDSRetriever` object sends requests to GRIS and/or GIIS servers to perform an LDAP search for `KGridMetadata` entries (step 1). The search operation

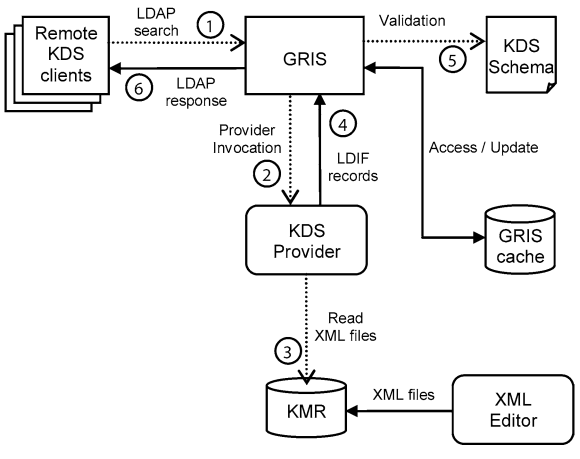


Figure 7. The KDS publishing system.

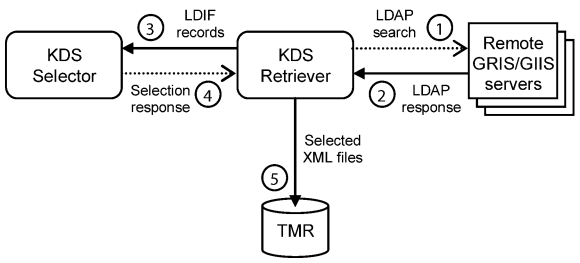


Figure 8. The KDS discovery system.

returns a set of LDIF entries, each one containing the information extracted from a single XML metadata file located in a remote KMR (step 2). The `KDSRetriever` submits the LDIF entries to a `KDSSelector` object (step 3), which responds by indicating the entries that should be selected on the basis of a given criterion (step 4). Selected entries are translated into XML files and are stored in the local TMR (step 5). In essence, a `KDSRetriever` object performs a local copy of the remote XML metadata documents that are selected by a customized `KDSSelector` object.

The interface `KDSSelector` is used to select metadata on the basis of different selection filters and parameters. To this end, a set of different classes implementing the `KDSSelector` interface can be developed to perform several selection tasks, ranging from very simple (e.g., based on the XML document modification time) to very complex ones (e.g., based on advanced content filtering and specified through specialized query languages such as XML Query).

5.2. *A P2P Approach for Metadata Management*

An attractive model that shares same common features with Grid computing systems and applications is the peer-to-peer model. P2P is a class of self-organizing systems or applications that takes advantage of distributed resources – storage, processing, information, and human presence – available at the Internet's edges. The P2P model could thus help to ensure Grid scalability: designers could use the P2P philosophy and techniques to implement non-hierarchical decentralized Grid systems.

In the last few years the Grid community has undertaken a development effort to align Grid technologies with Web Services. The *Open Grid Services Architecture* (*OGSA*) defines Grid Services as an extension of Web Services and lets developers integrate services and resources across distributed, heterogeneous, dynamic environments and communities [24]. The OGSA model provides an opportunity to integrate P2P models in Grid environments since it offers an open cooperation model that allows Grid entities to be composed in a decentralized way. In Globus Toolkit 3 – the current implementation of the OGSA – resource discovery is based, as well as in Globus Toolkit 2, on a hierarchical information service. The key component of such a service is the Index Service, a Grid Service that holds metadata about a set of resources registered to it.

We are designing an alternative architecture for discovering and querying resource metadata in Grids, which is based on OGSA but adopts a P2P approach for managing global queries on multiple Index Services.

Figure 9 shows the main components of this system. Three types of Grid Services are defined: *Index Services*, *Peer Services*, and *Contact Services*. As mentioned before, Index Services can be organized in a hierarchical structure. A top-level Index Service provides information about all the resources in a given Virtual Organization (VO). Peer Services are used to implement resource discovery across different VOs. There is one Peer Service per VO. Each Peer Service has a set of neighbour Peer Services, and exchanges query/response messages with them in a P2P mode. Contact Services are cache servers that store the URLs of known Peer Services. A Peer Service may contact one or more well known Contact Services to obtain the URLs of other Peer Services.

The system supports both local and global queries. A local query attempts to find information about resources in a given VO; it is performed by submitting the query to the Index Service of that VO. A global query aims to discover resources located in possibly different VOs; it is performed by submitting the query

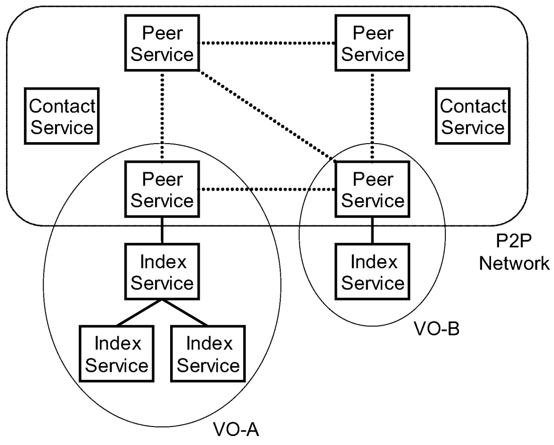


Figure 9. A P2P architecture for resource discovery on OGSA Grids.

to a Peer Service. This Peer Service will process that query locally (through the associated Index Service), and will forward it to its neighbours as in typical P2P networks. Whenever responses are received back by the Peer Service, they are forwarded to the requestor.

6. Related Work

The crucial role of metadata management for the effective designing of distributed data mining systems is widely recognized in the literature [22, 23, 26, 30, 31]. A recent workshop report released within the e-Science project [22] highlights the benefits of adopting standard representation of metadata to face the problems caused both by increasing data volume and by the heterogeneous and distributed nature of scientific data. In particular, the first of twelve recommendations produced by the mentioned workshop report emphasizes the role of XML as a standardization language: XML aids interoperability and flexibility, since data represented with XML combines rich structure, rich choice of tools, and wide acceptance.

In [23] the role of metadata in the context of the Semantic Grid is discussed. Here metadata is used to assist a three level programming model: the lowest level includes traditional code to implement a service; the next level uses agent technology and metadata to choose which services to use; the third level (workflow) links the chosen services to solve domain specific problems. When exploiting the Open Grid Services Architecture, based on the Grid Service technology [24], it is essential to integrate metadata embedded in services (i.e. information stored in the XML-based Service Data Elements provided by Grid Services) and metadata external to Grid Services, which could be stored in distributed databases and repositories with very variable scope and completeness (e.g., LDAP or UDDI indexes).

Several systems that support distributed data mining and data transformation processes have been recently proposed. A few of those systems operate on Grid environments, whereas most of the proposed systems work on clusters of computers or over the Internet.

Discovery Net [25] provides an architecture for building and managing knowledge discovery processes on the Grid. Like in the KNOWLEDGE GRID, metadata is crucial in the Discovery Net infrastructure. An XML language called *Discovery Process Markup Language* (*DPML*) is used to describe simple and compound applications. DPML represents a process as a data flow graph of nodes, each representing a service; a node description is an XML file containing *service parameters*, *history* of past parameter settings, and *notes*. Resources are characterized by the definition of data types stored in a *Meta-Information Server*: service composition is performed by assuring a proper matching of input and output data types. The use of metadata for scheduling is an issue not properly addressed in Discovery Net. Therefore, the approach used in the KNOWLEDGE GRID for managing abstract and instantiated execution plans could be useful in Discovery Net to cope with the dynamic nature of Grid resources.

MyGrid [26] is a project targeted at developing open source high-level middleware to support personalized experiments in biology on a Grid. The semantic complexity of such applications requires a careful definition and integration of metadata about source data, services, workflows etc. In [27], a mechanism is proposed to manage metadata directly related to resources (i.e., created at time of resource publishing), third-party metadata and annotations, and semantic metadata extracted from a domain specific ontology. MyGrid aims at providing and integrating different models for different kinds of metadata: e.g., Web Services Flow Language (WSFL) for workflow and DAML-S profile schemas for domain semantic metadata. The use of these models can be evaluated and exploited also in the KNOWLEDGE GRID. In fact, the Knowledge Discovery Service, as mentioned in Section 5, allows for separating the access method from the XML-based representation model of metadata.

A system implementing software tools and network services for distributed data mining and data intensive computing is Papyrus [28]. Papyrus has been developed for clusters and superclusters of workstations and it is composed of four software layers: data management, data mining, predictive modeling, and agent layer. That system uses PMML for predictive models and an XML language called *Data Space Markup Language* for data and information contained in clusters. Another distributed data mining suite based on Java is PaDDMAS [29], a component-based tool set that integrates pre-developed or custom packages by using a dataflow approach. Each system component is wrapped as a Java or CORBA object whose interface is specified in XML. XML definition may be used in PaDDMAS to automatically derive help on a particular component and to check the suitability of a component for analyzing a particular data set, the type of platforms that may support the component, etc. Finally, the Chimera system [30] proposes a language, called *Virtual Data Language* (*VDL*), to define metadata describing complex applications, in particular data transformation processes. A VDL document is structured in a way similar to an execution plan defined in the KNOWLEDGE GRID.

Whereas most of the above systems mainly focus on the data mining phase, Open DMIX [31] aims to provide OGSA-compliant services to facilitate the data preprocessing phase in distributed data mining applications. In particular, Open DMIX provides *data integration services* allowing for the integration of data from disparate sources, and *data exploration services* that permit the investigation of data sets in order to understand their important characteristics. Open DMIX is designed upon a layered architecture comprising network transport services, data access services, data integration and exploration services, data analysis services, and discovery services. The network transport services and the data access services are based upon the *Data Space Transfer Protocol* [32], a protocol designed to efficiently access and transport data and metadata.

Besides the systems discussed above, other interesting distributed data mining systems have been developed. In such systems metadata management appears to be not a central issue, because they focus on the use of dedicated platforms or make use of homogeneous software components. Among such systems, JAM [33] is an agent-based distributed data mining system developed to mine data stored in different sites for building the so called *meta-models* as a combination of several models learned at the different sites where data is stored. JAM uses Java applets to move data mining agents to remote sites. A sort of meta-learning, called *collective data mining* is implemented also in the BODHI system [34]. BODHI is another agent-based distributed data mining system implemented in Java.

Metadata management models are also deployed in other computer science areas such as problem solving environments (PSEs). Examples of significant PSEs that use XML-based metadata models for representation of heterogeneous resources are WebFlow and the Common Portal Application [35].

In [36] and [37], workflow management on the Grid is examined under the NSF-funded GridPhysics Network (GriPhyN) project. Two levels of workflows are defined: (i) an abstract workflow specifies resources by using logical files and logical component names; (ii) a concrete workflow is a refined workflow, ready to be executed. High-level services are developed to automate the process of building and executing workflows. In [37], several approaches exploitable for workflow scheduling and execution are discussed, ranging from the "full-plan-ahead" approach (scheduling decisions are made statically and globally before the workflow execution) to the "in-time local scheduling" (scheduling decisions are made dynamically and locally before the execution of each workflow task). The management of abstract and concrete workflows in GriPhyN has much in common with our management of abstract and instantiated execution plans. With regard to workflow scheduling, our use of resource checkpoints can be a useful compromise between static/global and dynamic/local approaches. Indeed, rescheduling operations are requested only at some specific and critical points within a workflow and such operations are facilitated by the availability of lists of concrete resources that have been discovered before workflow execution.

7. Conclusions

This paper discussed implementation and use of metadata for management of heterogeneous resources in Grid-based data mining applications. We motivated and presented the use of an XML-based approach to represent metadata and build a distributed information system for the KNOWLEDGE GRID environment that can be also used in different high-level Grid frameworks designed to support complex applications. We introduced and discussed the metadata structure for the main classes of resources involved in a data mining process (data mining software, data sources, mining results, and execution plans). For each resource class we reported a sample metadata document extracted from a real data mining application. Furthermore, we presented the implementation of the service used to publish, search, and retrieve metadata in the KNOWLEDGE GRID. We are currently using the metadata model implementation for developing distributed data mining applications running on the KNOWLEDGE GRID. Finally, we presented the architecture of a P2P model for metadata management on Grids that we are designing for scalable metadata management in future large scale Grids.

As a result of our work we experienced the key role of metadata for the implementation of resource representation, search, and discovery services in heterogeneous computing environments such as Grids and meta-computing systems. In particular, metadata models are at the basis of Grid information systems and they are major players where complex applications, such as knowledge discovery processes or scientific simulations, must be developed in such environments.

Acknowledgements

This work has been partially funded by the projects "MIUR Fondo Speciale SP3: GRID COMPUTING: Tecnologie abilitanti e applicazioni per eScience" and "MIUR FIRB Grid.it: Piattaforme abilitanti per griglie computazionali ad elevate prestazioni orientate ad organizzazioni virtuali scalabili."

References

1. M. Cannataro and D. Talia, "The KNOWLEDGE GRID," *Communications of the ACM*, January 2003, pp. 89–93.
2. C. Mastroianni, D. Talia and P. Trunfio, "Managing Heterogeneous Resources in Data Mining Applications on Grids Using XML-Based Metadata," in *Proceedings IPDPS 2003*, IEEE Computer Society Press, April 2003.
3. Foster, C. Kesselman and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *Internat. J. Supercomputer Applications*, Vol. 15, No. 3, 2001.
4. Reagan W. Moore, "Persistent Archives for Data Collections", UC San Diego SDSC TR-1999-2, October 1999.
5. W. Johnston, "NASA's Information Power Grid: Production Grid Experience with Distributed Computing and Data Management," in *Second Global Grid Forum Workshop (GGF2)*, Washington, DC, 2001.
6. The Globus Project, "The Monitoring and Discovery Service." http://www.globus.org/mds
7. RFC 2251 – Lightweight Directory Access Protocol (v3).
8. M. Cannataro, A. Congiusta, D. Talia and P. Trunfio, "A Data Mining Toolset for Distributed High-Performance Platforms," in *Proceedings 3rd Int. Conference Data Mining 2002*, Bologna, WIT Press, September 2002, pp. 41–50.
9. XML Schema. http://www.w3.org/XML/Schema
10. XML Query. http://www.w3.org/XML/Query
11. Xerces library. http://xml.apache.org
12. P. Cheeseman and J. Stutz, "Bayesian Classification (AutoClass): Theory and Results," in U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth and R. Uthurusamy (eds.), *Advances in Knowledge Discovery and Data Mining*, AAAI Press/MIT Press, pp. 61–83, 1996.
13. M.S. Chen, J. Han and P.S. Yu, "Data Mining: An Overview from a Database Perspective," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 8, No. 6, pp. 866–883, 1996.
14. R.L. Grossman, M.F. Hornick and G. Meyer, "Data Mining Standard Initiatives," *Communications of the ACM*, Vol. 45, No. 8, August 2002.
15. PMML 2.0 – DTD for Clustering Models. http://www.dmg.org/pmmlspecs_v2/ClusteringModel.htm
16. The Semantic Grid project. http://www.semanticgrid.org
17. M. Cannataro and C. Comito, "A Data Mining Ontology for Grid Programming," in *Proc. 1st Int. Workshop on Semantics in Peer-to-Peer and Grid Computing*, Budapest, May 2003.
18. The Globus Project, "The Globus Resource Allocation Manager." http://www.globus.org/gram
19. The Globus Project, "The Globus Resource Specification Language." http://www.globus.org/gram/rsl_spec1.html
20. RFC 2849 – The LDAP Data Interchange Format (LDIF) – Technical Specification.
21. The Globus Project, "MDS 2.2 GRIS Specification Document: Creating New Information Providers." http://www.globus.org/mds/creating_new_providers.pdf
22. B. Mann, R. Williams, M. Atkinson, K. Brodlie, A. Storkey and C. Williams, "Scientific Data Mining, Integration, and Visualization," Report of the workshop held at the e-Science Institute, Edinburgh, October 2002. http://www.cacr.caltech.edu/~roy/papers/sdmiv-ltr.pdf
23. G. Fox, "Data and Metadata on the Semantic Grid," *Computing in Science and Engineering*, Vol. 5, No. 5, September 2003.
24. Foster, C. Kesselman, J. Nick and S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration," Globus Project, 2002. www.globus.org/research/papers/ogsa.pdf
25. V. Curcin, M. Ghanem, Y. Guo, M. Kohler, A. Rowe, J. Syed and P. Wendel, "Discovery Net: Towards a Grid of Knowledge Discovery," ACM KDD 2002.
26. The MyGrid project. http://mygrid.man.ac.uk/myGrid/
27. P. Lord, C. Wroe, R. Stevens, C. Goble, S. Miles, L. Moreau, K. Decker, T. Payne and J. Papay, "Semantic and Personalized Service Discovery," in *Proceedings WI/IAT 2003 Workshop on Knowledge Grid and Grid Intelligence*, Halifax, Canada, October 2003.
28. R. Grossman, S. Bailey, S. Kasif, D. Mon, A. Ramu and B. Malhi, "The Preliminary Design of Papyrus: A System for High Performance, Distributed Data Mining over Clusters, Meta-Clusters and Super-Clusters," in *International KDD'98 Conference*, 1998, pp. 37–43.
29. O.F. Rana, D.W. Walker, M. Li, S. Lynden and M. Ward, "PaDDMAS: Parallel and Distributed Data Mining Application Suite," in *Proc. International Parallel and Distributed Processing Symposium (IPDPS/SPDP)*, IEEE Computer Society Press, 2000, pp. 387–392.

30. Foster, J. Vöckler, M. Wilde and Y. Zhao, "Chimera: A Virtual Data System for Representing, Querying, and Automating Data Derivation," in *SSDBM 2002*, pp. 37–46.
31. R. Grossman, Y. Gu, D. Hanley, X. Hong and G. Rao, "Open DMIX – Data Integration and Exploration Services for Data Grids, Data Web and Knowledge Grid Applications," in *Proceedings WI/IAT 2003 Workshop on Knowledge Grid and Grid Intelligence*, Halifax, Canada, October 2003.
32. R. Grossman and M. Mazzucco, "Dataspace – a Web Infrastructure for the Exploratory Analysis and Mining of Data," *IEEE Computing in Science and Engineering*, pp. 44–51, July/August 2002.
33. S.J. Stolfo, A.L. Prodromidis, S. Tselepis, W. Lee, D.W. Fan and P.K. Chan, "JAM: Java Agents for Meta-Learning over Distributed Databases," in *International KDD'97 Conference*, 1997, pp. 74–81.
34. H. Kargupta, B. Park, D. Hershberger and E. Johnson, "Collective Data Mining: A New Perspective toward Distributed Data Mining," in H. Kargupta and P. Chan (eds.), *Advances in Distributed and Parallel Knowledge Discovery*, AAAI Press, 2000.
35. E. Houstis, A. Catlin, N. Dhanjani, J. Rice, J. Dongarra, H. Casanova, D. Arnold and G. Fox, "Problem-Solving Environments," in *The Parallel Computing Sourcebook*, M. Kaufmann Publishers, 2002.
36. E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, K. Blackburn, A. Lazzarini, A. Arbree, R. Cavanaugh and S. Koranda, "Mapping Abstract Complex Workflows onto Grid Environments," *Journal of Grid Computing*, Vol. 1, No. 1, pp. 25–39, 2003.
37. E. Deelman, J. Blythe, Y. Gil and C. Kesselman, "Workflow Management in GriPhyN," in J. Nabrzyski, J.M. Schopf and J. Weglarz (co-ed.), *Grid Resource Management*, Kluwer Academic Publishers, 2003.