Programming Knowledge Discovery Workflows in Service-Oriented Distributed Systems

Eugenio Cesario[1], Marco Lackovic[2],
Domenico Talia[1,2], Paolo Trunfio[2,*]

[1] *ICAR-CNR, Rende (CS), Italy*
[2] *DEIS, University of Calabria, Rende (CS), Italy*

SUMMARY

In several scientific and business domains very large data repositories are generated. To find interesting and useful information in those repositories, efficient data mining techniques and knowledge discovery processes must be used. The exploitation of data mining techniques in science help scientists in hypothesis formation and give them a support on their scientific practices, whereas in industrial processes data mining can exploit existing data sources as a real value for companies that can take advantage from the knowledge that can be extracted from their large data sources. Data mining tasks are often composed by multiple stages that may be linked each other to form various execution flows. Moreover, data mining tasks are often distributed since they involve data and tools located over geographically distributed environments. Therefore, it is fundamental to exploit effective paradigms, such as services and workflows, to model data mining tasks that are both multi-staged and distributed. This paper discusses data mining services and workflows for analyzing scientific data in high performance distributed environments such as Grids and Clouds. We discuss how it is possible to define basic and complex services for supporting distributed data mining tasks in Grids. We also presents a workflow formalism and a service-oriented programming framework, named DIS3GNO, for designing and running distributed knowledge discovery processes in the Knowledge Grid system. DIS3GNO supports all the phases of a knowledge discovery process, including composition, execution, and results visualization. After introducing DIS3GNO, some relevant use cases implemented by it and a performance evaluation of the system are discussed.

KEY WORDS: Distributed data mining; Workflows; Grid computing; Knowledge Grid

E-mail: cesario@icar.cnr.it (E. Cesario), mlackovic@deis.unical.it (M. Lackovic), talia@deis.unical.it (D. Talia), trunfio@deis.unical.it (P. Trunfio)
[*]Correspondence to: Paolo Trunfio, DEIS, University of Calabria, Via P. Bucci 41C, 87036 Rende (CS), Italy

1. INTRODUCTION

Data mining applications and knowledge discovery in databases (KDD) processes are often composed of multiple stages (e.g., data extraction, data filtering, data analysis, results evaluation) that may be linked each other by different dependencies to form various execution flows. Moreover, data mining tasks are often distributed since they involve data and tools located over geographically distributed environments, like computational Grids. Therefore, it is fundamental to provide formalisms and environments to design and execute knowledge discovery processes that are both multi-staged and distributed.

Workflows are commonly-used formalisms to represent data and execution flows associated to complex data mining applications. A data mining workflow is a graph in which nodes typically represent data sources, filtering tools, data mining algorithms, and visualizers, and edges represent execution dependencies among nodes. An important benefit of workflows is that, once defined, they can be stored and retrieved for modifications and/or re-execution: this allows users to define typical data mining patterns and reuse them in different contexts. We worked in this direction by defining a service-oriented workflow formalism and a visual software environment, named DIS3GNO, to design and execute distributed data mining tasks over the Knowledge Grid [1], a service-oriented framework for distributed data mining on the Grid.

The goal of the workflow formalism discussed in this paper is to allow domain-expert users to design a distributed data mining task without specific expertise about Grid programming. The DIS3GNO system allows users to compose distributed data mining workflows, execute the workflows onto the Knowledge Grid, and visualize the results of the data mining tasks performed. DIS3GNO submits the user-defined workflow to the Knowledge Grid services, which manage the distributed computation onto the Grid infrastructure in a way that is completely transparent to the user.

The remainder of this paper is organized as follows. Section 2 focuses on knowledge discovery in distributed and Grid computing environments. Section 3 outlines the main features of the Knowledge Grid system. Section 4 describes the DIS3GNO system, its workflow formalism, and how distributed data mining workflows are executed on the Knowledge Grid. Section 5 presents some uses cases and a performance evaluation of the system. Section 6 discusses related work. Finally, Section 7 concludes the paper.

2. DISTRIBUTED AND GRID-BASED KNOWLEDGE DISCOVERY

As outlined earlier, scientific applications have to deal with a massive volume of data. Data mining algorithms working on very large datasets take a very long time on conventional computers to get results. In order to improve performances, some distributed approaches have been proposed.

Distributed data mining (*DDM*) works by analyzing data in a distributed fashion and pays particular attention to the trade-off between centralized collection and distributed analysis of data. This approach is particularly suitable for applications that deal with a very large amount of data (e.g., transaction data, scientific simulation and telecommunication data), which cannot be analyzed in a single site on traditional machines in acceptable times.

In the last decade, Grid computing and Cloud computing systems integrated both distributed and parallel computing, representing a privileged infrastructure for high-performance data and knowledge management. In particular, Grid computing was conceived as a paradigm for coordinated resource sharing and problem solving in advanced science and engineering applications. For these reasons, Grids offer an effective support to the implementation and use of knowledge discovery systems by exploiting *Grid-based data mining* approaches.

In the following, distributed and Grid-based data mining models and systems are discussed.

2.1. Distributed Data Mining

Traditional warehouse-based architectures for data mining suppose to have a centralized data repository. Such a centralized approach is inappropriate for most distributed and ubiquitous data mining applications. In fact, the long response time, lack of proper use of distributed resource, and the use of centralized data mining algorithms do not cope well with distributed environments. A scalable solution for decentralized applications calls for distributed processing of data, controlled by the available resources and human factors.

For example, let us consider an ad hoc wireless sensor network where the different sensor nodes are monitoring some time-critical events. Central collection of data from every sensor node may create traffic over the limited bandwidth wireless channels and this may also drain a lot of power from the devices. As another example, let us consider the World Wide Web as it contains distributed data and computing resources. An increasing number of databases (e.g., weather databases, oceanographic data, etc.) and data streams (e.g., emerging disease information, etc.) are currently made available on-line, and many of them change frequently. It is easy to think of many applications that require regular monitoring of these diverse and distributed sources of data.

A distributed approach to analyze this data is aimed at being more scalable, particularly when the application involves a large number of data sites. Hence, in this case we need data mining architectures that pay careful attention to the distribution of data, computing and communication, in order to access and use them in a near optimal fashion. Most DDM algorithms are designed upon the potential parallelism they can apply over the given distributed data. Typically, the same algorithm operates on each distributed data site concurrently, producing one local model per site. Subsequently, all local models are aggregated to produce the final model. This schema is common to several distributed data mining algorithms such as *meta-learning*, *collective data mining* and *ensemble learning.*

The *meta-learning* technique [2] aims at building a global classifier from a set of inherently distributed data sources. Meta-learning is basically a two-step process: first, a number of independent classifiers are generated by applying learning programs to a collection of distributed and homogeneous datasets in parallel. Then, the classifiers computed by local learning programs are collected in a single site and combined to obtain a global classifier.

Collective data mining [3] exploits a different strategy: instead of combining partial local models, it builds the global model through the identification of significant sets of local information. In other words, the local blocks are directly composed to form the global model. This result is based on the fact that any mining function can be expressed in a distributed fashion using a set of appropriate basis functions.

Ensemble learning [4] aims at improving classification accuracy by aggregating predictions of multiple classifiers. An ensemble method constructs a set of base classifiers from training data and performs classification by voting (in the case of classification) or by averaging (in the case of regression) on the predictions made by each classifier. The final result is the ensemble classifier, which tends to have higher classification quality than any single classifier composing it.

2.2. Grid-based Data Mining

The Grid is a computing infrastructure to develop applications over geographically distributed sites, providing for protocols and services enabling the integrated and seamless use of remote computing power, storage, software, and data, managed and shared by different organizations. Some significant examples of Grid applications include biomedicine [5], protein folding studies [6], molecular simulation [7], high-energy physics experiments [8], climate/weather modelling [9], and earthquake simulation [10].

Grid protocols and services are provided by environments such as *Globus Toolkit*[†], *gLite*[‡], *Unicore*[§], *XtreemOS*[¶], and *GridWay*[‖]. In particular, *Globus Toolkit* is the most widely-used middleware in scientific and data-intensive Grid applications, and represents a de-facto standard for implementing Grid systems. It addresses security, information discovery, resource and data management, communication, fault-detection, and portability issues.

Since Grid application areas range widely, specialized services have been implemented to meet the needs of different application contexts. In particular, *data Grids* have been designed to easily store, move, and manage large datasets in distributed data-intensive applications. Besides core data management services, *knowledge-based Grids*, built on top of computational and data Grid environments, offer higher-level services for data analysis, inference, and discovery in scientific and business areas [11]. The availability of *knowledge 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 [12].

Several systems for distributed data mining exploiting the Grid infrastructure have been designed and implemented [13]; among them, some popular systems are DataMiningGrid, Discovery Net, GridMiner, and the Knowledge Grid framework.

DataMiningGrid [14] is a Grid environment for executing data analysis and knowledge discovery tasks in a wide range of different application fields, including the automotive,

[†] http://www.globus.org/toolkit

[‡] http://glite.web.cern.ch/glite

[§] http://www.unicore.eu

[¶] http://www.xtreemos.eu

[‖] http://www.gridway.org

biological and medical, environmental and ICT sectors. It provides functionalities for data manipulation, resource brokering, and application search according to different data mining tasks and methodologies, and supports different types of parameter sweeps.

Discovery Net [15] allows users to integrate data analysis software and data sources made available by third parties. The building blocks are the so-called *Knowledge Discovery Services*, distinguished in *Computation Services* and *Data Services*. Discovery Net provides services, mechanisms and tools for specifying knowledge discovery processes. The functionalities of Discovery Net can be accessed through an interface exposed as Web service.

GridMiner [16] aims at covering the main aspects of knowledge discovery on Grids. Key components in GridMiner are *Mediation Service*, *Information Service*, *Resource Broker*, and *Online Analytical Processing (OLAP) Cube Management*. These are the so called GridMiner Base services, because they provide basic services to GridMiner Core services. GridMiner Core services include services for data integration, process management, data mining, and OLAP.

The *Knowledge Grid* [1] is a software system that we developed for providing services to execute distributed data mining tasks in Grid environments. Workflows play a fundamental role in the Knowledge Grid at different levels of abstraction. A client application submits a distributed data mining application to the Knowledge Grid by describing it through an XML workflow formalism named *conceptual model*. The conceptual model describes data and algorithms to be used, without specifying information about their location or implementation.

The Knowledge Grid creates an execution plan for the workflow on the basis of the conceptual model and executes each node of the workflow as a service by using the resources effectively available. To this end, the Knowledge Grid follows a two-step approach: it initially models an *abstract execution plan* that in a second step is resolved into a *concrete execution plan*. The abstract execution plan may not contain specific information about the involved Grid resources (e.g., the actual site where a data mining tool will be executed), while in the concrete execution plan all the resources must be actually specified, by finding a mapping between requested resources and available ones in the distributed computing infrastructure.

To the best of our knowledge, the Knowledge Grid was the first system for distributed knowledge discovery in Grid environments. Since then, some other systems sharing a similar service-oriented approach have been proposed, the most important ones are mentioned above. One specific feature of the Knowledge Grid is the use of a unified metadata management model, which represents both abstract and concrete resources using a common XML formalism. This is the enabling mechanism for the abstract-to-concrete execution plan mapping, which allows both the re-execution of the same abstract execution plan (i.e., workflow) at different times, and the possible optimization of the execution flow by choosing the most suitable resources among the available ones. Another key aspect in the Knowledge Grid is the possibility for a user to reuse previously-generated models in subsequent analysis.

In the remainder of this paper we focus on the Knowledge Grid system and its programming interface. In particular, Section 3 provides a short overview of the Knowledge Grid architecture and implementation, while Section 4 presents the DIS3GNO system that we recently developed to program distributed data mining applications as service-oriented workflows and for running them over the Knowledge Grid.

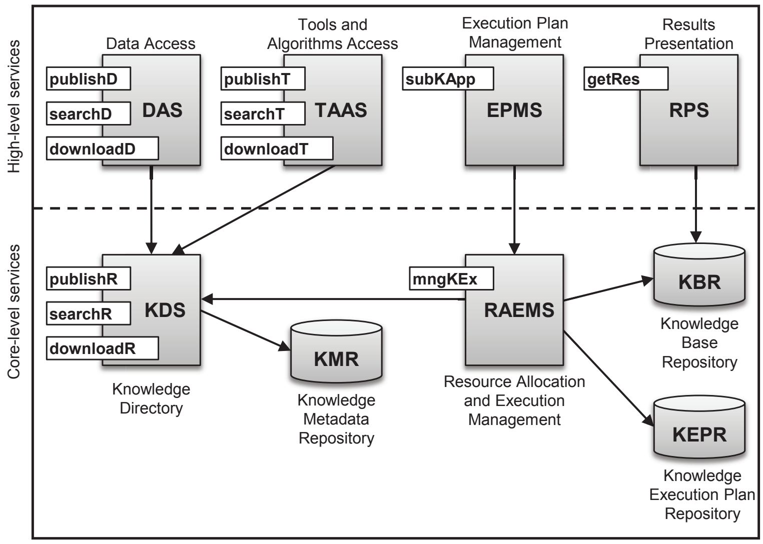


Figure 1. Knowledge Grid layered services.

3. THE KNOWLEDGE GRID

The Knowledge Grid services are organized in two hierarchical layers [1]: the *core layer* and the *high-level layer*, as shown in Figure 1. The design idea is that client applications directly interact with high-level services that, in order to satisfy client requests, invoke suitable operations exported by the core-level services. In turn, core-level services perform their operations by invoking basic services provided by available Grid environments running on the specific host, as well as by interacting with other core-level services.

The high-level layer includes the following services:

- *Data Access Service* (*DAS*), which provides operations for publishing, searching and downloading data to be mined (`publishData`, `searchData`, and `downloadData` operations);
- *Tools and Algorithms Access Service* (*TAAS*), which is responsible for publishing, searching and downloading tools and algorithms for data extraction, pre-processing and mining (`publishTool`, `searchTool`, and `downloadTool` operations);
- *Execution Plan Management Service* (*EPMS*), which receives a conceptual model of the data mining task through the `submitKApplication` operation, translates it into an abstract execution plan, and passes it to the RAEMS service (see below).
- *Results Presentation Service* (*RPS*), which allows to retrieve the results (i.e., the inferred models) of previous data mining computations through the `getResults` operation.

The core-level layer includes two services:

- *Knowledge Directory Service* (*KDS*), which is responsible for managing metadata about the Knowledge Grid resources (data, tools and algorithms). It provides three operations (`publishResource`, `searchResource`, and `downloadResource`) to publish, search and download resource metadata, which are stored in a *Knowledge Metadata Repository* (*KMR*).
- *Resource Allocation and Execution Management Service* (*RAEMS*), which starting from an abstract execution plan (received through the `manageKApplication` operation) generates a concrete execution plan and manages its execution. Generated execution plans are stored in a *Knowledge Execution Plan Repository* (*KEPR*), while the results are stored in a *Knowledge Base Repository* (*KBR*).

Metadata play a key role in the Knowledge Grid since they allow to describe the resources of interest, for subsequently supporting their effective discovery and use in data mining applications. The Knowledge Grid defines metadata models for describing two main types of resources, namely data sources and data mining tools. Such models are expressed using an XML formalism, and their high-level structure is briefly described in the following.

The data source metadata model is composed of two parts: a first section that contains information for retrieving data (file system information, location, database details), and another section providing information about the logical and/or physical structure of data (file format, attribute list, etc.).

The tool metadata model allows to specify various details of a tool such as the type of input data (arff, csv, etc.), the type of knowledge mined (association rules discovery, data classification, clustering, etc.), the type of techniques exploited in the mining process (hierarchical or partitional clustering, decision trees, etc.) and some technical details about its usage and invocation syntax.

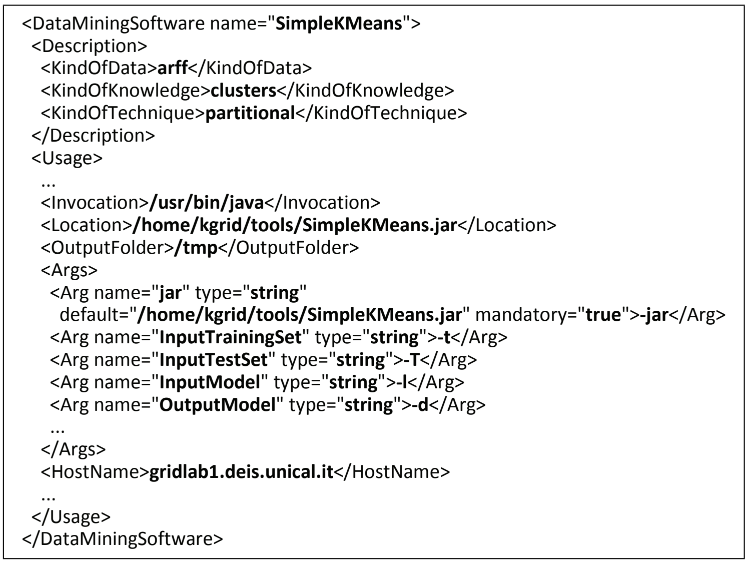
To add new capabilities, such as a new algorithm or dataset available on a certain machine, it is required to submit the corresponding metadata description using the TAAS or DAS publishing operations mentioned above, which results in storing such metadata information into the KMR.

Figure 2 shows an extract of a metadata element that describes a data mining tool in the Knowledge Grid. More details about structure and use of metadata can be found in [17].

All the Knowledge Grid services have been implemented as Web services that comply with the Web Services Resource Framework (WSRF) family of standards, as described in a previous work [18]. In particular, we used the WSRF library provided by Globus Toolkit 4 [19], as well as some basic Grid services (e.g., reliable file transfer, authentication and authorization) provided by the same toolkit. Despite the fact that WSRF and Globus Toolkit 4 are today considered obsolete standards and technologies, both continue to be widely employed by Web services-based Grid designers, especially in the area of stateful resource management due to their effectiveness and reliability.

Within the Knowledge Grid project, a visual software environment named DIS3GNO has recently been implemented to allow a user to:

- program distributed data mining workflows;



```
<DataMiningSoftware name="SimpleKMeans">
 <Description>
  <KindOfData>arff</KindOfData>
  <KindOfKnowledge>clusters</KindOfKnowledge>
  <KindOfTechnique>partitional</KindOfTechnique>
 </Description>
 <Usage>
  ...
  <Invocation>/usr/bin/java</Invocation>
  <Location>/home/kgrid/tools/SimpleKMeans.jar</Location>
  <OutputFolder>/tmp</OutputFolder>
  <Args>
   <Arg name="jar" type="string"
    default="/home/kgrid/tools/SimpleKMeans.jar" mandatory="true">-jar</Arg>
   <Arg name="InputTrainingSet" type="string">-t</Arg>
   <Arg name="InputTestSet" type="string">-T</Arg>
   <Arg name="InputModel" type="string">-l</Arg>
   <Arg name="OutputModel" type="string">-d</Arg>
   ...
  </Args>
  <HostName>gridlab1.deis.unical.it</HostName>
  ...
 </Usage>
</DataMiningSoftware>
```

Figure 2. An example of metadata descriptor for a clustering tool.

- execute the workflow onto the Knowledge Grid;
- visualize the results of a data mining task.

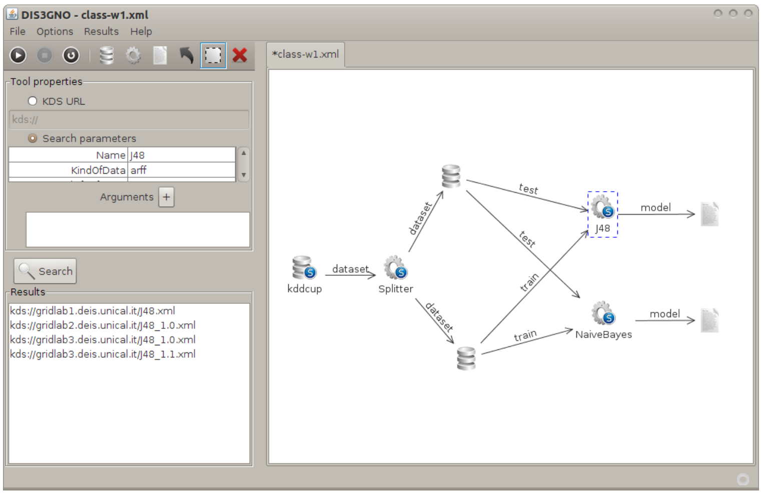
DIS3GNO performs the mapping of the user-defined workflow to the conceptual model and submits it to the Knowledge Grid services, managing the overall computation in a way that is transparent to a user.

4. THE DIS3GNO SYSTEM

In supporting user to develop applications, DIS3GNO is the user interface for two main Knowledge Grid functionalities:

- *Metadata management.* DIS3GNO provides an interface to publish and search metadata about data and tools, through the interaction with the DAS and TAAS services.
- *Design and Execution management.* DIS3GNO provides an environment to program and execute distributed data mining applications as service-oriented workflows, through the interaction with the EPMS service.

The DIS3GNO Graphical User Interface (GUI), depicted in Figure 3, has been designed to reflect this two-fold functionality. In particular, it provides a panel (on the left) devoted to search resource metadata, and a panel (on the right) to compose and execute data mining workflows.



Name: J48
KindOfData: arff

kds://gridlab1.deis.unical.it/J48.xml
kds://gridlab2.deis.unical.it/J48_1.0.xml
kds://gridlab3.deis.unical.it/J48_1.0.xml
kds://gridlab3.deis.unical.it/J48_1.1.xml

Figure 3. A screenshot of the DIS3GNO GUI.

In the top-left corner of the window there is a menu used for opening, saving and creating new workflows, viewing and modifying some program settings and viewing the previously computed results present in the local file system. Under the menu bar there is a toolbar containing some buttons for the execution control (starting/stopping the execution and resetting the nodes statuses) and other for the workflow editing (creation of nodes representing datasets, tools or models, creation of edges, selection of multiple nodes and deletion of nodes or edges).

4.1. Workflow Representation

In DIS3GNO a workflow is represented as a directed acyclic graph whose nodes represent resources and whose edges represent dependencies among the resources.

The types of resources that can be present in a data mining workflow (graphically depicted by the icons in Figure 4) are:

- *Dataset*, representing a dataset;
- *Tool*, representing a tool to perform any kind of operation which may be applied to a dataset (data mining, filtering, splitting, etc.) or to a model (e.g., voting operations);
- *Model*, represents a knowledge model (e.g., a decision tree, a set of association rules, etc.), that is the result produced by a data mining tool.

Tools can be either implemented by the users or taken from third-party software. A reliable source of tools is the Weka toolkit [20], from which we have taken most of the algorithms used in our experiments with DIS3GNO. Besides data mining algorithms, Weka provides a wide set

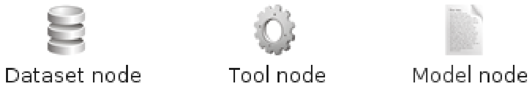


Figure 4. Nodes types.

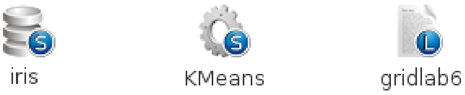


Figure 5. Nodes labels.

of useful filtering tools that can also be used to query remote datasets and to create a local copy of them for subsequent processing.

Each node contains a description of a resource as a set of properties which provide information about its features and actual use. This description may be full or partial: in other words, it is both possible to specify a particular resource and its location in the Grid, or just a few of its properties, leaving to the system the task of finding a resource matching the specified characteristics and its location. In the former case we will refer to the resource as *concrete*, in the latter one as *abstract*.

For example, in the case of a data mining tool, one could be interested in any algorithm, located in any node of the Grid, provided it is a classification algorithm able to handle “arff” files, or could want specifically the algorithm named *NaiveBayes* located in a specified host. Once the workflow will be executed, the Knowledge Grid middleware will find a concrete resource matching the metadata, whether they are completely or partially specified. Clearly only dataset and tool nodes can be either concrete or abstract, the model node can’t be abstract as it represents the result of a computation. The model node has only one property, the *location*, which if left empty will be implicitly set to the same location of the tool node in input.

When a particular resource property is entered, a label is attached below to the corresponding icon, as shown in the example in Figure 5. The property chosen as the label is the one considered most representative for the resource, i.e. the *Name* for the dataset and tool nodes and the *Location* for the model node.

In order to ease the workflow composition and to allow a user to monitor its execution, each resource icon bears a symbol representing the status in which the corresponding resource is at a given time. When the resource status changes, as consequence of the occurrence of certain

Table I. Nodes composition-time statuses.

Symbol	Meaning
⚙	No information provided
⚙S	Abstract resource
⚙K	Concrete resource
L	Location set

Table II. Nodes run-time statuses.

Symbol	Meaning
⚙⏯	Matching resource found
⚙↻	Running
⚙✖	Resource not found
⚙⚠	Execution failed
⚙✓	Task completed successfully

events, its status symbol changes accordingly. The resource statuses can be divided in two categories: the *composition-time* and the *run-time* statuses.

The *composition-time* statuses (shown in Table I) are useful during the workflow composition phase. They are as follows:

1. *No information provided* = no parameter has been specified in the resource properties;
2. *Abstract resource* = the resource is defined through constraints about its features, but it is not known a priori; the *S* in the icon stands for *search*, meaning that the resource has to be searched in the Grid;
3. *Concrete resource* = the resource is specifically defined through its KDS URL; the *K* in the icon stands for *KDS URL*;
4. *Location set* = a location for the model has been specifically set (this status is pertinent to model nodes only).

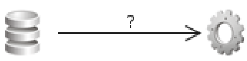
The *run-time* statuses (shown in Table II), useful during the workflow execution phase, are:

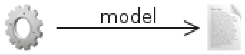
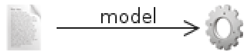
1. *Matching resource found* = a concrete resource matching the metadata has been found;
2. *Running* = the resource is being executed/managed.
3. *Resource not found* = the system hasn't found a resource matching the metadata;
4. *Execution failed* = some error has occurred during the management of the corresponding resource;
5. *Task completed successfully* = the corresponding resource has successfully fulfilled its task.

Each resource may be in one of these run-time statuses only in a specific phase of the workflow execution: i.e. status 1 and 2 only during the execution, status 3 and 4 during or after the execution, status 5 only after the execution.

Table III. Nodes connections.

First resource	**Second resource**	**Label**	**Meaning**	**Graphical representation**
dataset	dataset	`transfer`	Explicit file transfer	

dataset	tool	`dataset,`
		`train,`
		`test`
			Type of input for a tool node	

tool	dataset	`dataset`	Dataset produced by a tool	
tool	model	`model`	Model produced by a data mining algorithm	
model	tool	`model`	Model received by a tool	
model	model	`transfer`	Explicit transfer of a model	

The nodes may be connected with each other through edges, establishing dependency relationships among them using specific patterns. The patterns currently supported in DIS3GNO are: *sequence* (a task is started after the completion of the preceding task), *parallel split* (in any node with multiple outgoing edges, the thread of control is split into multiple ones, thus allowing parallel execution), *synchronization* (any node with multiple incoming edges is implicitly a point of synchronization) [21]. All the possible connections are show in Table III; those not present in Table III are not allowed and the graphical user interface prevents a user to create them.

When an edge is being created between two nodes, a label is automatically attached to it representing the kind of relationship between the two nodes. In most of the cases this relationship is strict but in one case (dataset-tool connection) requires further input from a user to be specified.

The possible edge labels are:

- *dataset*: indicates that the input or output of a tool node is a dataset;
- *train*: indicates that the input of a tool node has to be considered a training set;
- *test*: indicates that the input of a tool node has to be considered a test set;
- *transfer*: indicates an explicit transfer of a dataset, or a result of a computation, from one Grid node to another;
- *model*: indicates a result of a computation of a data mining algorithm.

4.2. Workflow Composition

To outline the main functionalities of DIS3GNO, we briefly describe how it is used to compose and run a data mining workflow. By exploiting the DIS3GNO GUI, a user can compose a

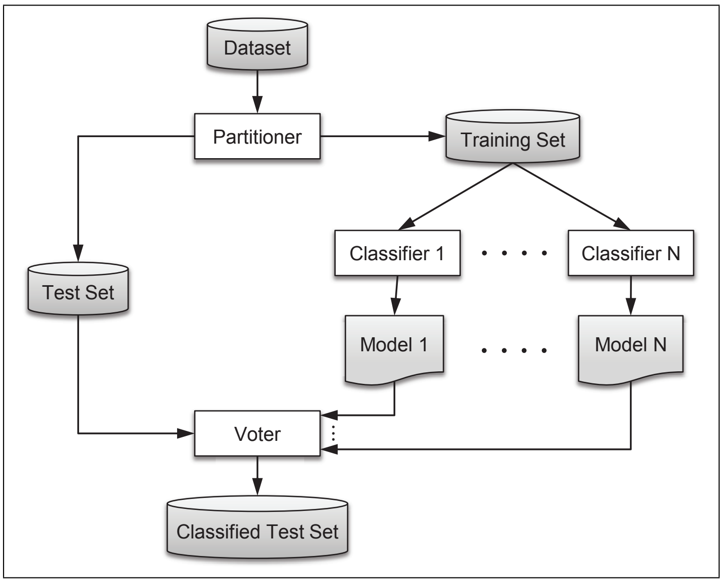
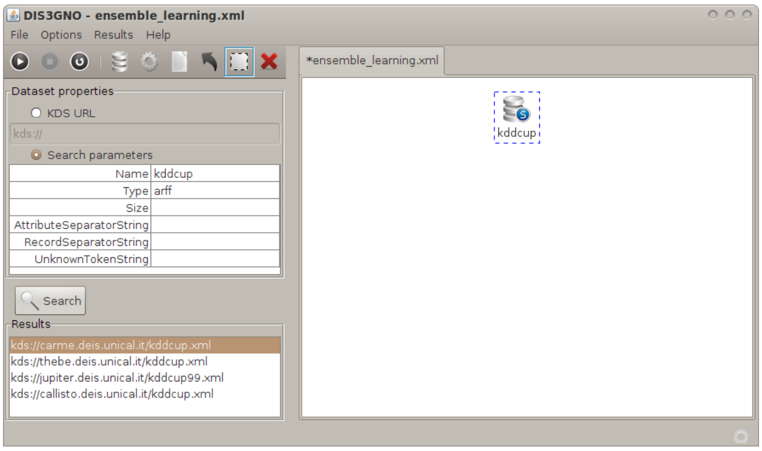


Figure 6. Logical schema of an ensemble learning application.

workflow by selecting from the toolbar the type of resource to be inserted in the workflow (a *dataset*, a *tool* or a *model* node), and clicking on the workflow composition panel. Such operation can be repeated as many times as needed to insert all the required application nodes. Then, she/he has to insert suitable edges by setting, for each one, the specific dependency relationship between the nodes (as described in Section 4.1 and summarized in Table III). Typically, most nodes in a workflow represent abstract resources. In other terms, a user initially concentrates on the application logic, without focusing on the actual datasets or data mining tool to be used.

Let us suppose that a user wants to compose and execute the ensemble learning application depicted in Figure 6. In contrast to ordinary machine learning approaches which try to learn one model from training data, ensemble methods build a set of models and combine them to obtain the final model [22]. In a classification scenario, an ensemble method constructs a set of *base classifiers* from training data and performs classification by taking a vote on the predictions made by each classifier.

As shown in Figure 6, the input dataset is split, using a partitioner tool, into two parts, namely a *training set* and a *test set*. The training set is given in input to N classification algorithms which run in parallel to build N independent classification models from it. Then, a voter tool performs an ensemble classification by assigning to each instance of the test set the class predicted by the majority of the N models generated at the previous step.



Dataset properties

○ KDS URL

kds://

● Search parameters

Name	kddcup
Type	arff
Size	
AttributeSeparatorString	
RecordSeparatorString	
UnknownTokenString	

Results

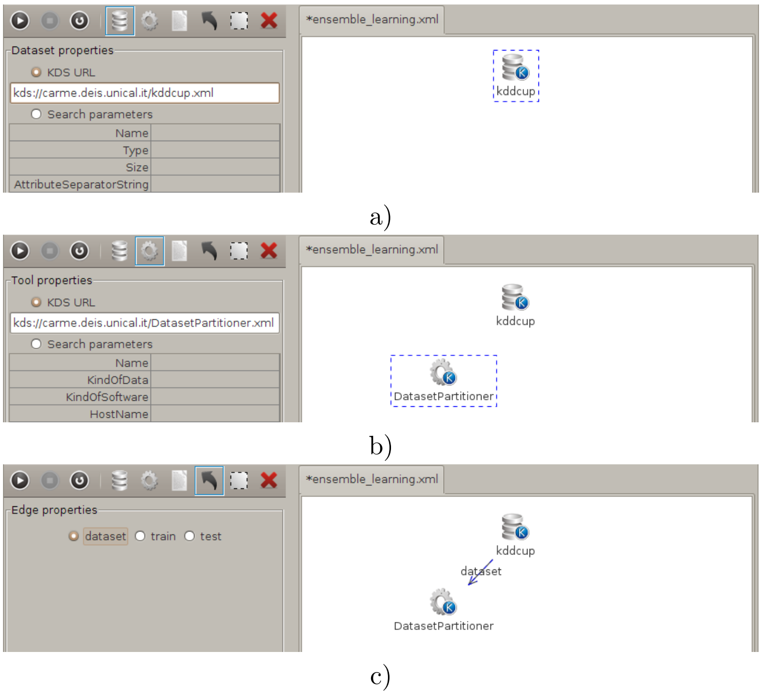
kds://carme.deis.unical.it/kddcup.xml
kds://thebe.deis.unical.it/kddcup.xml
kds://jupiter.deis.unical.it/kddcup99.xml
kds://callisto.deis.unical.it/kddcup.xml

Figure 7. Insertion of the input dataset icon with specification of its properties and search for matching resources.

By using DIS3GNO, the ensemble learning application can be designed as follows. First, a user chooses the input dataset (Figure 7). To do that, she/he selects from the toolbar the *dataset* icon and drags it into the workflow composition panel. In order to associate such an icon to a concrete resource, the user specifies name and format of the desired dataset into the *Search parameters* panel. The search is started by pressing the *Search* button; on completion, the system lists the set of datasets matching the search criteria (left-bottom part of the GUI).

After the selection of one dataset, the URL of such dataset is associated with the dataset icon (Figure 8a). This operation changes the dataset icon mark from S (resource still to be selected) to K (resource identified by a KDS URL). In the same way, the user inserts a *tool* node, which is associated with the KDS URL of the desired partitioner (Figure 8b). Then, the user must specify the relationship between the two nodes. To do that, an edge linking the *dataset* and the *tool* icons is created and labelled appropriately (Figure 8c).

According to the ensemble learning scenario, two dataset icons representing the output of the partitioner are added to the workflow (Figure 9a). Then, the user proceeds by adding the classification algorithms that are required to build the base models. We assume that the user wants to use four classification algorithms (ConjunctiveRule, NaiveBayes, RandomForest and J48) specified as abstract resources (see Section 4.1). For example, Figure 9b shows the insertion of the first classification algorithm (ConjunctiveRule) and the specification of its properties (name of software and type of data supported). The algorithm icon is marked with an S to remind that the corresponding resource will be searched and made concrete at runtime. Similarly, the other three classification algorithms are added, and an edge between the training set and the four algorithms is created (Figure 9c).



Dataset properties

● KDS URL

kds://carme.deis.unical.it/kddcup.xml

○ Search parameters

Name

Type

Size

AttributeSeparatorString

a)

Tool properties

● KDS URL

kds://carme.deis.unical.it/DatasetPartitioner.xml

○ Search parameters

Name

KindOfData

KindOfSoftware

HostName

b)

Edge properties

● dataset ○ train ○ test

c)

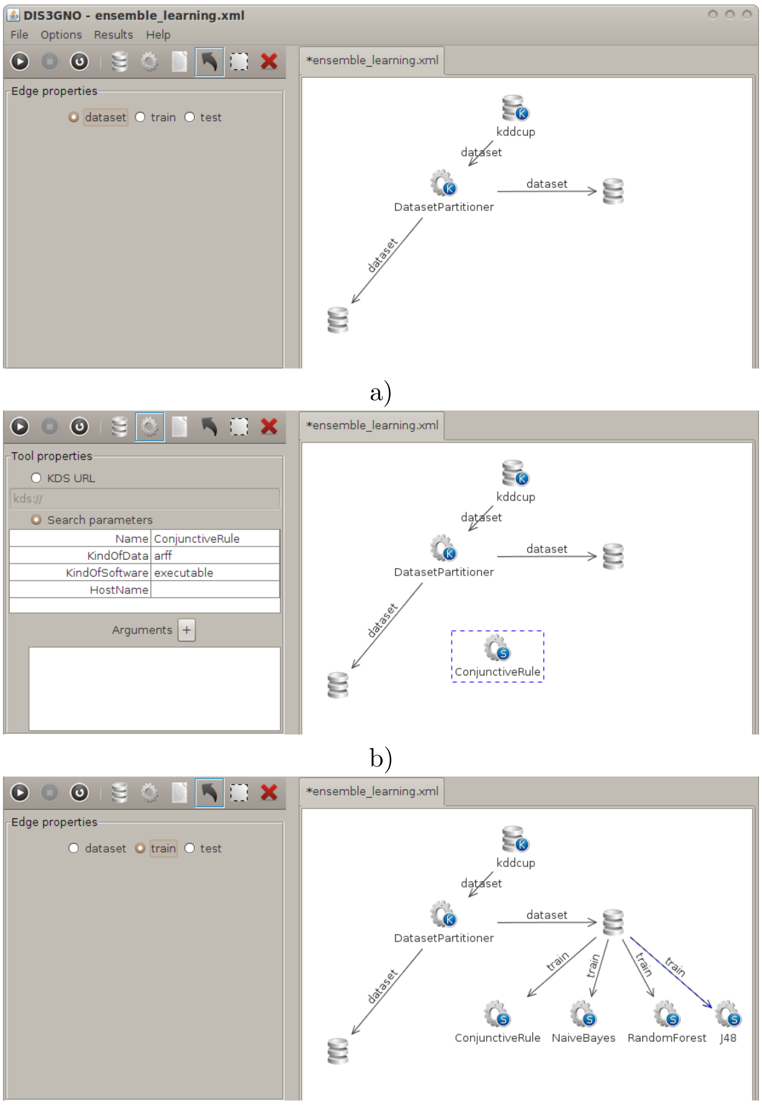
Figure 8. a) Selection of the input dataset; b) Insertion and selection of a partitioner tool; c) Insertion of a labelled edge between dataset and partitioner.

Figure 10 shows the complete workflow. It includes: (*i*) a *model* node connected to each classification algorithm; (*ii*) a *tool* node representing a voter that takes in input the test set and the four base models; (*iii*) the output dataset obtained as output of the voter tool.

The workflow can be submitted to the EPMS service by pressing the *Run* button in the toolbar. As a first action, if user credentials are not available or have expired, a Grid Proxy Initialization window is loaded. After that, the workflow execution actually starts and proceeds as detailed in the next section.

4.3. Execution Management

Starting from the data mining workflow designed by a user, DIS3GNO generates an XML representation of the data mining application referred to as *conceptual model*. DIS3GNO passes the *conceptual model* to a given EPMS, which is in charge of transforming it into an *abstract execution plan* for subsequent processing by the RAEMS. The RAEMS receives the abstract execution plan and creates a *concrete execution plan*. To accomplish this task, the RAEMS needs to evaluate and resolve a set of resources and services, by contacting the KDS and



Edge properties

● dataset ○ train ○ test

a)

Tool properties

○ KDS URL

kds://

● Search parameters

Name	ConjunctiveRule
KindOfData	arff
KindOfSoftware	executable
HostName	

Arguments +

b)

Edge properties

○ dataset ● train ○ test

c)

Figure 9. a) Insertion of two dataset icons representing the partitioner output; b) Insertion and specification of an abstract tool resource; c) Workflow after insertion and specification of all the classification algorithms and associated input edges.

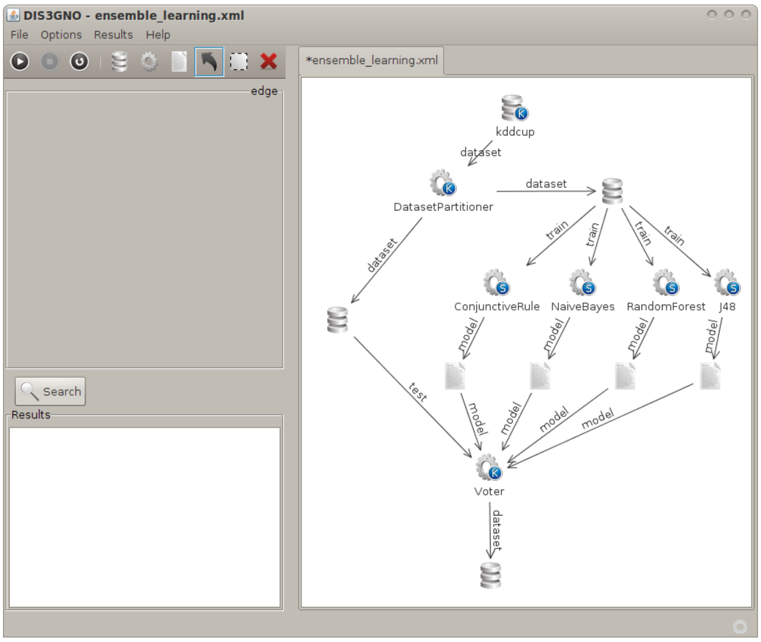


Figure 10. The complete workflow including the base models, a voter tool, and the output dataset.

choosing those matching the requirements specified by the abstract execution plan. In case multiple resources match the requirements, the RAEMS adopts a round-robin strategy to select one of them.

As soon as the RAEMS has built the concrete execution plan, it is in charge of coordinating its execution by invoking the coordinated execution of services corresponding to the nodes of the concrete execution plan. The status of the computation is notified to the EPMS, which in turn forwards the notifications to the DIS3GNO system for visualization.

Figure 11 describes the interactions that occur when an invocation of the EPMS is performed. In particular, the figure outlines the sequence of invocations of others services, and the interchanges with them when a data mining workflow is submitted for allocation and execution. To do this, the EPMS exposes the `submitKApplication` operation, through which it receives a conceptual model of the application to be executed (step 1).

As an example, Figure 13 shows the conceptual model generated by DIS3GNO starting from the simple workflow shown in Figure 12. Basically, the conceptual model is a textual representation of the graph expressed by the visual workflow, which is suitable to be passed to the EPMS for further processing. Conceptual models are expressed using the *GraphML* formalism that is widely employed to represent graphs in XML. Note that the conceptual model reflects the fact that the original workflow contains a concrete resource (the input dataset

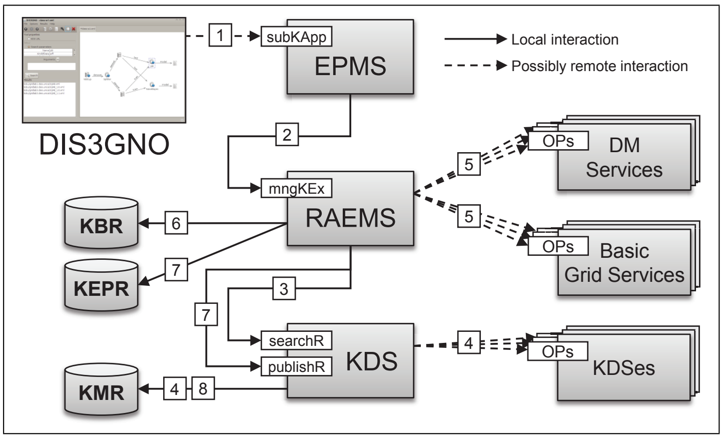


Figure 11. Execution management. The sequence of operation invocations is showed for all services involved in the mapping and execution of a data mining workflow.

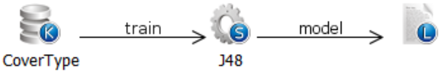
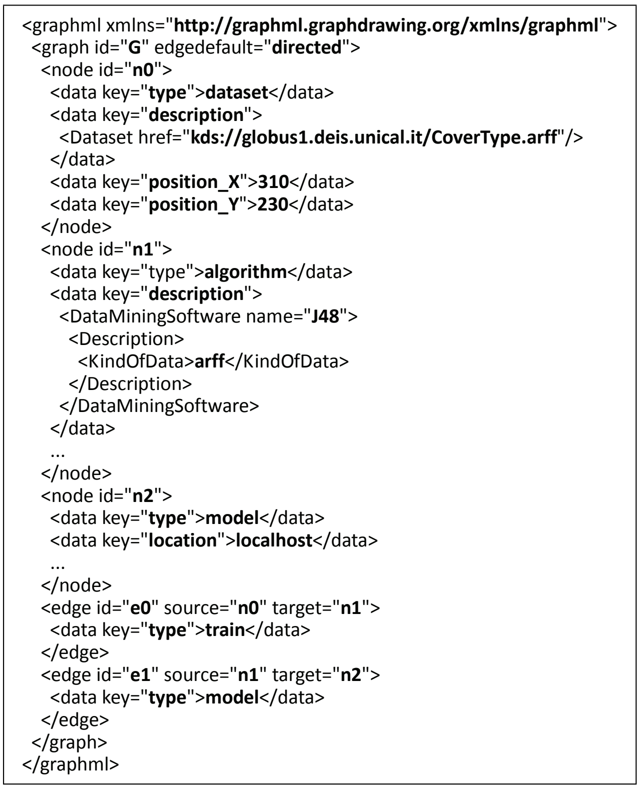


Figure 12. Simple workflow example.

CoverType) and an abstract one (the J48 algorithm). In fact, node $n0$ in the conceptual model is fully specified by its KDS URL, which refers to the metadata descriptor of the CoverType dataset. On the other hand, node $n1$ only specifies the name of the algorithm (J48) and the kind of data to be processed (arff), thus keeping the resource abstract and giving to the system the task of mapping it to a concrete resource.

The basic role of the EPMS is to transform the conceptual model into an abstract execution plan for subsequent processing by the RAEMS. An abstract execution plan is a more formal representation of the structure of the application. Generally, it does not contain information on the physical Grid resources and services to be used, but rather constraints about them. Figure 14 shows an example of abstract execution plan generated by the EPMS starting from the conceptual model displayed in Figure 13.

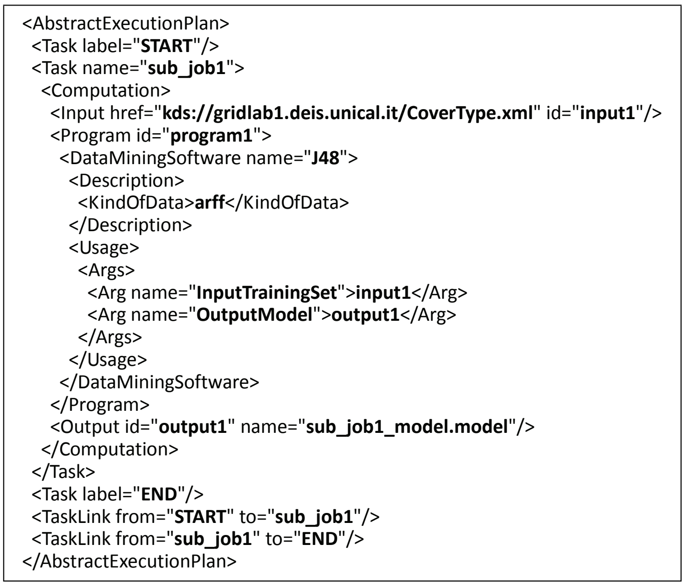
The RAEMS exports the `manageKExecution` operation, which is invoked by the EPMS and receives the abstract execution plan (step 2). First of all, the RAEMS queries the local KDS (through the `searchResource` operation) to obtain information about the resources needed to instantiate the abstract execution plan (step 3). The KDS performs the search both accessing the local Knowledge Metadata Repository (KMR) and querying a set of remote KDSes (step 4).



```
<graphml xmlns="http://graphml.graphdrawing.org/xmlns/graphml">
 <graph id="G" edgedefault="directed">
  <node id="n0">
   <data key="type">dataset</data>
   <data key="description">
    <Dataset href="kds://globus1.deis.unical.it/CoverType.arff"/>
   </data>
   <data key="position_X">310</data>
   <data key="position_Y">230</data>
  </node>
  <node id="n1">
   <data key="type">algorithm</data>
   <data key="description">
    <DataMiningSoftware name="J48">
     <Description>
      <KindOfData>arff</KindOfData>
     </Description>
    </DataMiningSoftware>
   </data>
   ...
  </node>
  <node id="n2">
   <data key="type">model</data>
   <data key="location">localhost</data>
   ...
  </node>
  <edge id="e0" source="n0" target="n1">
   <data key="type">train</data>
  </edge>
  <edge id="e1" source="n1" target="n2">
   <data key="type">model</data>
  </edge>
 </graph>
</graphml>
```

Figure 13. GraphML representation of the simple workflow in Figure 12.

To reach as many remote KDSes as needed, an unstructured peer-to-peer overlay similar to Gnutella [23] is built among the Knowledge Grid nodes. The peer-to-peer overlay is constructed by assigning to each node a small set of neighboring nodes. Each node sends a KDS query to its neighbors, which in turn can forward it to their neighbors to ensure a wider network coverage. As in Gnutella, this query flooding is controlled in two ways: *i*) each time a query is forwarded, an associated time-to-leave (TTL) counter is decremented by one; when the TTL values equals zero, the query forwarding is stopped; *ii*) if a node receives the same query more than once, as a consequence of looping paths that may be present in the overlay, the query is discarded without further processing. In the current implementation, the list of neighbors is static and is created at each node by editing a configuration file. This requires that the node administrator has to know in advance a set of active nodes for the purpose. In dynamic scenarios nodes may be added to or removed from the Grid over time. To cope with these scenarios, we planned, for a future release of the system, to allow nodes to obtain their neighbors by querying some index servers maintaining an updated list of active nodes.



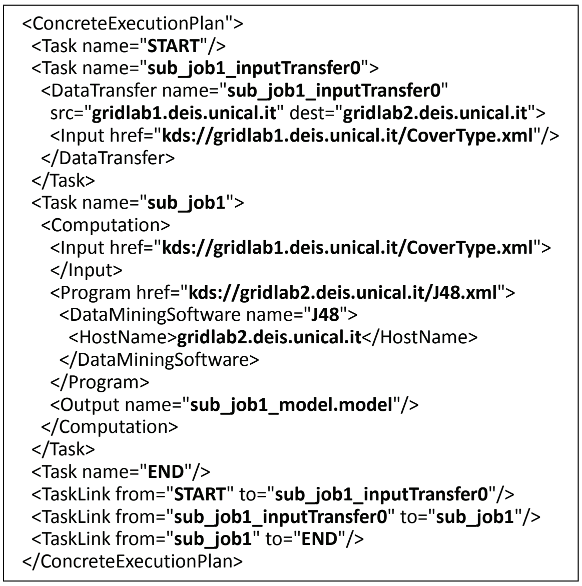
```
<AbstractExecutionPlan>
 <Task label="START"/>
 <Task name="sub_job1">
  <Computation>
   <Input href="kds://gridlab1.deis.unical.it/CoverType.xml" id="input1"/>
   <Program id="program1">
    <DataMiningSoftware name="J48">
     <Description>
      <KindOfData>arff</KindOfData>
     </Description>
     <Usage>
      <Args>
       <Arg name="InputTrainingSet">input1</Arg>
       <Arg name="OutputModel">output1</Arg>
      </Args>
     </Usage>
    </DataMiningSoftware>
   </Program>
   <Output id="output1" name="sub_job1_model.model"/>
  </Computation>
 </Task>
 <Task label="END"/>
 <TaskLink from="START" to="sub_job1"/>
 <TaskLink from="sub_job1" to="END"/>
</AbstractExecutionPlan>
```

Figure 14. Abstract execution plan generated after the conceptual model in Figure 13.

Figure 15 shows an example of concrete execution plan generated by the RAEMS starting from the abstract execution plan in Figure 14. Note that, in the concrete execution plan, the J48 algorithm is instantiated to a concrete resource that is fully specified by its KDS URL. Moreover, compared with the abstract execution plan, the concrete one includes an additional *DataTransfer* operation that consists in copying the input dataset to the node where the selected J48 instance is located. In fact, while the data transfer operation is implicit in the original workflow, it must be explicitly specified in the concrete execution plan.

After the concrete execution plan is obtained, the RAEMS coordinates the actual execution of the overall computation. To this purpose, the RAEMS invokes the appropriate data mining services (DM Services) and basic Grid services (e.g., file transfer services), as specified by the concrete execution plan (step 5). The RAEMS stores the results of the computation into the Knowledge Base Repository (KBR) (step 6), while the execution plan is stored into the Knowledge Execution Plan Repository (KEPR) (step 7). To make available the results stored in the KBR, it is necessary to publish results metadata into the KMR. To this end, the RAEMS invokes the `publishResource` operation of the local KDS (steps 7 and 8).

Figure 16 shows the final screenshot of the DIS3GNO interface when the execution of the data mining workflow has been completed and the final classification result has been created and showed in an ad hoc window.



```
<ConcreteExecutionPlan">
 <Task name="START"/>
 <Task name="sub_job1_inputTransfer0">
  <DataTransfer name="sub_job1_inputTransfer0"
   src="gridlab1.deis.unical.it" dest="gridlab2.deis.unical.it">
   <Input href="kds://gridlab1.deis.unical.it/CoverType.xml"/>
  </DataTransfer>
 </Task>
 <Task name="sub_job1">
  <Computation>
   <Input href="kds://gridlab1.deis.unical.it/CoverType.xml">
   </Input>
   <Program href="kds://gridlab2.deis.unical.it/J48.xml">
    <DataMiningSoftware name="J48">
     <HostName>gridlab2.deis.unical.it</HostName>
    </DataMiningSoftware>
   </Program>
   <Output name="sub_job1_model.model"/>
  </Computation>
 </Task>
 <Task name="END"/>
 <TaskLink from="START" to="sub_job1_inputTransfer0"/>
 <TaskLink from="sub_job1_inputTransfer0" to="sub_job1"/>
 <TaskLink from="sub_job1" to="END"/>
</ConcreteExecutionPlan>
```

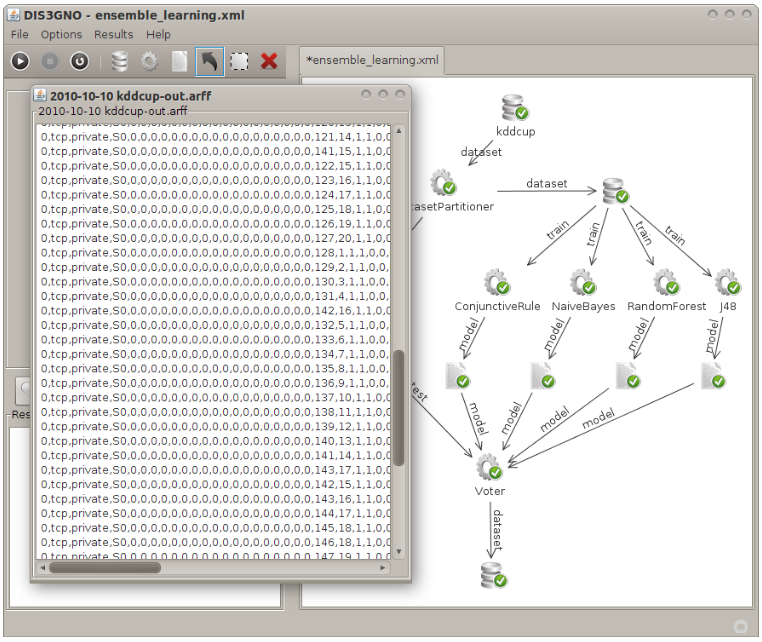
Figure 15. Concrete execution plan generated starting from the abstract execution plan in Figure 14.

5. USE CASES AND PERFORMANCE

In this section we discuss two examples of distributed data mining workflows designed and executed on a Grid using DIS3GNO. The first workflow is a parameter sweeping application in which a dataset is processed using multiple instances of the same classification algorithm with different parameters, with the goal of finding the best classifier based on some accuracy parameters. The second workflow is the ensemble learning application already introduced in the previous section.

Both workflows are representative, in terms of complexity and size, of most distributed data mining applications that can be found in the literature. Indeed, DIS3GNO has been designed for distributed data mining applications, not for general engineering workflows. Therefore, it can manage tens of tasks that is typically a sufficient number for distributed data mining applications both in science and business domain.

To evaluate the effectiveness of the system as well as its performance in terms of scalability, both the workflows have been executed on a Grid including up to 11 nodes. The nodes were equipped with different processors, having a computing power ranging from that of a Pentium IV with 2.4 GHz to that of Xeon 5160 with 3.0 GHz, with RAM size ranging from 2 to 4 GB.



```
0,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,121,14,1,1,0,0
0,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,141,15,1,1,0,0
0,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,122,15,1,1,0,0
0,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,123,16,1,1,0,0
0,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,124,17,1,1,0,0
0,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,125,18,1,1,0,0
0,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,126,19,1,1,0,0
0,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,127,20,1,1,0,0
0,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,128,1,1,1,0,0,
0,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,129,2,1,1,0,0,
0,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,130,3,1,1,0,0,
0,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,131,4,1,1,0,0,
0,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,142,16,1,1,0,0
0,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,132,5,1,1,0,0,
0,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,133,6,1,1,0,0,
0,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,134,7,1,1,0,0,
0,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,135,8,1,1,0,0,
0,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,136,9,1,1,0,0,
0,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,137,10,1,1,0,0
0,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,138,11,1,1,0,0
0,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,139,12,1,1,0,0
0,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,140,13,1,1,0,0
0,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,141,14,1,1,0,0
0,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,143,17,1,1,0,0
0,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,142,15,1,1,0,0
0,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,143,16,1,1,0,0
0,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,144,17,1,1,0,0
0,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,145,18,1,1,0,0
0,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,146,18,1,1,0,0
0,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,147,19,1,1,0,0
```

Figure 16. The final visualization of DIS3GNO after completion of the application.

5.1. Parameter Sweeping Workflow

We used DIS3GNO to compose an application in which a given dataset is analyzed by running multiple instances of the same classification algorithm, with the goal of obtaining multiple classification models from the same data source.

The dataset *covertype*** from the UCI KDD archive, has been used as data source. The dataset contains information about forest cover type for a large number of sites in the United States. Each dataset instance, corresponding to a site observation, is described by 54 attributes that give information about the main features of a site (e.g., elevation, aspect, slope, etc.). The 55th attribute contains the cover type, represented as an integer in the range 1 to 7. The original dataset is made of 581,012 instances and is stored in a file having a size of 72MB. From this dataset we extracted three datasets with 72,500, 145,000 and 290,000 instances and a file size of 9 MB, 18 MB and 36 MB respectively. Then we used DIS3GNO to perform a classification analysis on each of those datasets.

**http://kdd.ics.uci.edu/databases/covertype/covertype.html

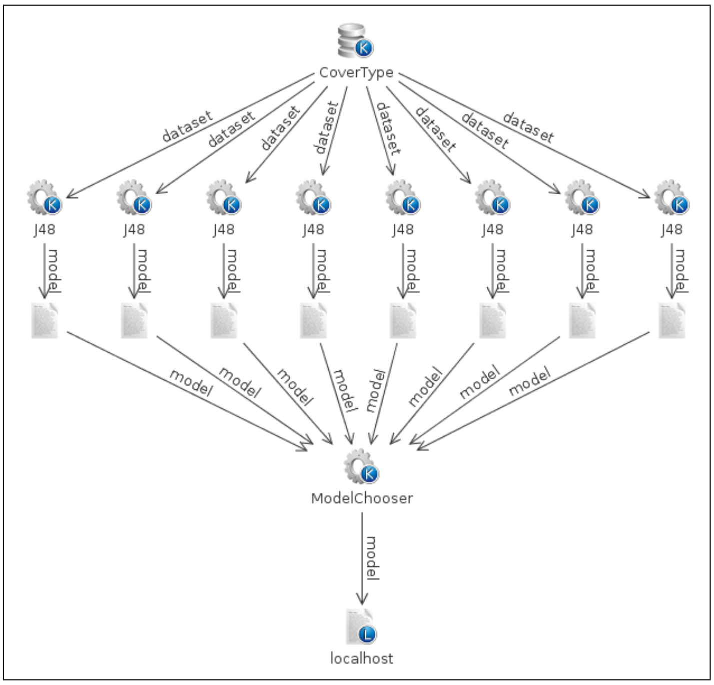


Figure 17. Parameter sweeping workflow.

DIS3GNO has been used to run an application in which 8 independent instances of the J48 algorithm perform a different classification task on the covertype dataset. In particular, each J48 instance has been asked to classify data using a different value of confidence, ranging from 0.15 to 0.50. The same application has been executed using a number of computing nodes ranging from 1 to 8 to evaluate the system speedup.

The workflow corresponding to the application is shown in Figure 17. It includes a *dataset* node (representing the covertype dataset) connected to 8 *tool* nodes, each one associated with an instance of the J48 classification algorithm with a different value of confidence (ranging from 0.15 to 0.50). These nodes are in turn connected to another *tool* node, associated with a model chooser which selects the best classification model among those learnt by the J48 instances. Finally, the node associated with the model chooser is connected to a *model* node having the location set to localhost; this enforces the model to be transferred to the client host for its visualization.

The workflow has been executed using a number of computing nodes ranging from 1 to 8 for each of the three datasets (9 MB, 18 MB and 36 MB) in order to evaluate the speedup of the system. Table IV reports the execution times of the application when 1, 2, 4 and 8 computing nodes are used. The 8 classification tasks that constitute the overall application are indicated

Table IV. Task assignments and execution times for the parameter sweeping workflow (times expressed as hh:mm:ss).

No of nodes	**Task assignments (Node ← Tasks)**	**Exec. time 9 MB**	**Exec. time 18 MB**	**Exec. time 36 MB**
1	$N_1 \leftarrow DM_1, ..., DM_8$	2:43:47	7:03:46	20:36:23
2	$N_1 \leftarrow DM_1, DM_3, DM_5, DM_7$ $N_2 \leftarrow DM_2, DM_4, DM_6, DM_8$	1:55:19	4:51:24	14:14:40
4	$N_1 \leftarrow DM_1, DM_5$ $N_2 \leftarrow DM_2, DM_6$ $N_3 \leftarrow DM_3, DM_7$ $N_4 \leftarrow DM_4, DM_8$	58:30	2:26:48	7:08:16
8	$N_i \leftarrow DM_i$ for $1 \leq i \leq 8$	32:35	1:21:32	3:52:32

as $DM_1..DM_8$, corresponding to the tasks of running J48 with a confidence value of 0.15, 0.20, 0.25, 0.30, 0.35, 0.40, 0.45, and 0.50, respectively. The table shows how the classification tasks are assigned to the computing nodes (denoted as $N_1..N_8$), as well as the execution times for each dataset size.

When the workflow is executed on more than one node, the execution time includes the overhead due to file transfers. For example, in our network scenario, the transfer of a 36 MB dataset from the user node to a computing node takes on average 15 seconds. This value is small as compared to the amount of time required to run a classification algorithm on the same dataset, which takes between 2.5 and 3.9 hours depending on the computing node. The overall execution time also includes the amount of time needed to invoke all the involved services (i.e., EPMS, RAEMS, KDS) as required by the workflow. However, such an amount of time (approximatively 2 minutes) is negligible as compared to the total execution time.

For the 36 MB dataset, the total execution time decreases from more than 20 hours obtained using 1 computing node, to less than 4 hours obtained with 8 nodes. The achieved execution speedup ranged from 1.45 using 2 nodes, to 5.32 using 8 nodes. Similar trends have been registered with the other two datasets. The execution times and speedup values for different number of nodes and dataset sizes are shown in Figure 18.

5.2. Ensemble Learning Workflow

As mentioned earlier, ensemble learning is a machine learning paradigm where multiple learners are trained to solve the same problem. In the following we consider the ensemble learning scenario already introduced in Section 4.2, whose corresponding workflow is shown in Figure 10.

As input dataset we used *kddcup99*[††]. This dataset, used for the KDD'99 Competition, contains a wide set of data produced during seven weeks of monitoring in a military network

[††] http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

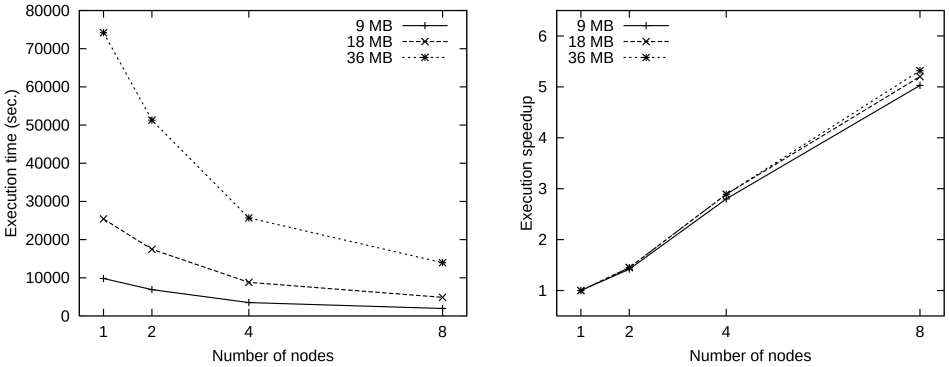


Figure 18. Execution times and speedup values for different numbers of nodes and dataset sizes, for the parameter sweeping workflow.

Table V. Task assignments and execution times for the ensemble learning workflow.

No of nodes	**Task assignments (Node ← Tasks)**	**Exec. time 100 MB**	**Exec. time 140 MB**	**Exec. time 180 MB**
1	$N_1 \leftarrow DM_1, ..., DM_4$	1:30:50	2:31:14	3:34:27
2	$N_1 \leftarrow DM_1, DM_3$ $N_2 \leftarrow DM_2, DM_4$	1:03:47	1:37:05	2:07:05
4	$N_1 \leftarrow DM_i$ for $1 \leq i \leq 4$	46:16	1:13:47	1:37:23

environment subject to simulated intrusions. We extracted three datasets from it, with 940,000, 1,315,000 and 1,692,000 instances and a size of 100 MB, 140 MB and 180 MB.

As shown in Figure 10, DIS3GNO has been used to split the dataset into two parts: a test set (1/3 of the original dataset) and a training set (2/3 of the original dataset). The latter has been processed using four classification algorithms: ConjuctiveRule, NaiveBayes, RandomForest and J48. The models generated by the four classification algorithms are then collected to a node where they are given to a voter component; the classification is performed and evaluated on the test set by taking a vote, for each instance, on the predictions made by each classifier.

The same workflow has been executed, for each of the three datasets, using a number of computing nodes ranging from 1 to 4 (excluding the node where we performed the voting operation) to evaluate the speedup of the system. Table V reports the execution times of the application when 1, 2 and 4 computing nodes are used. The four tasks are indicated as $DM_1..DM_4$, corresponding to ConjuctiveRule, NaiveBayes, RandomForest and J48 respectively. The table shows how the tasks are assigned to the computing nodes, as well as the execution times for each dataset size.

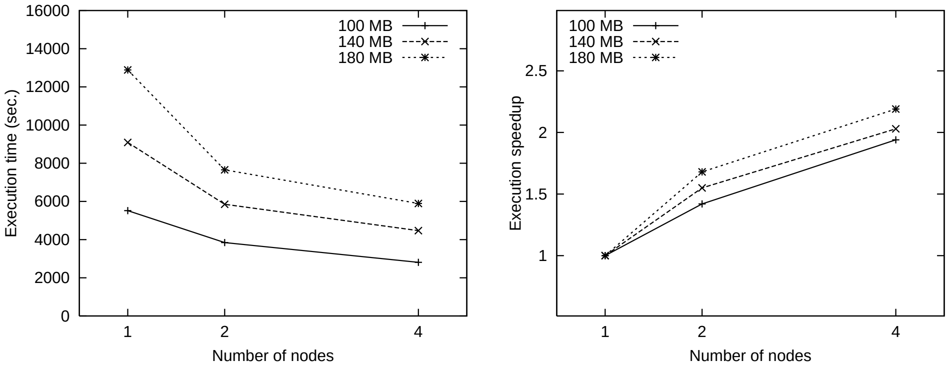


Figure 19. Execution times and speedup values for different numbers of nodes and dataset sizes, for the ensemble learning workflow.

The execution times and speedup values for different number of nodes and dataset sizes are reported in Figure 19. In this case, the speedup is lower than that obtained with the parameter sweeping workflow. This is due to the fact that the four algorithms used require very different amounts of time to complete their execution on a given dataset. In fact, the overall execution time is bound to the execution time of the slowest algorithm, thus limiting the speedup. However, the absolute amount of time saved by running the application on a distributed environment is still significant, particularly for the largest dataset when four computing nodes are used.

6. RELATED WORK

Other workflow systems have been proposed for Grid environments. However, most of them are not specifically designed for distributed data mining applications. Among them the most popular ones are Askalon [24], Kepler [25], Pegasus [26], Taverna [27], Triana [28], and Weka4WS [29].

Askalon [24] is an application development and runtime environment for the Grid. Developed at the University of Innsbruck, Austria, it uses a custom language called Abstract Grid Workflow Language (AGWL) for describing Grid workflow applications at a high level of abstraction. It has a Service Oriented Architecture (SOA)-based runtime environment with stateful services and uses the Globus Toolkit as Grid platform.

Kepler [25] provides a graphical user interface and a run-time engine that can execute workflows (with an emphasis on ecology and geology) either from within the graphical interface or from a command line. It is developed and maintained by a team consisting of several key institutions at the University of California. Kepler works based on the concept of *directors*, which dictate the models of execution used within a workflow. It is a java-based application that is maintained for the Windows, OS X, and Linux operating systems and freely available under the BSD License.

The Pegasus [26] project, developed at the University of Southern California, encompasses a set of technologies to execute workflow-based applications in a number of different environments, i.e., desktops, campus clusters, Grids, and Clouds. The worfklow management system of Pegasus can manage the execution of complex workflows on distributed resources and it is provided with a sophisticated error recovery system.

Taverna [27] is an open source tool for designing and executing workflows, developed at the University of Manchester. Its own workflow definition language is characterized by an implicit iteration mechanism (single node implicit parallelism). The Taverna team has primarily focused on supporting the Life Sciences community (biology, chemistry and medical imaging) although it does not provide any analytical or data services itself. It supports different types of Web services, including WSDL-based, Soaplab, BioMoby and BioMart services.

Triana [28] is a problem solving environment, developed at the Cardiff University, which combines a visual interface with data analysis tools. It can connect heterogeneous tools (e.g. Web services, Java units, JXTA services) on one workflow. Triana uses its own custom workflow language, although it can use other external workflow language representations such as BPEL4WS[‡‡] which are available through *pluggable* language readers and writers. Triana comes with a wide variety of built-in tools for signal-analysis, image-manipulation, desktop publishing, etc.

Weka4WS [29] is a framework developed at the University of Calabria, to extend the widely-used Weka toolkit [20] for supporting distributed data mining on Grid environments. In particular, Weka4WS includes a Grid-enabled version of the Weka Knowledge Flow environment, which allows the parallel and distributed execution of data mining workflows [30]. Weka4WS has been implemented by using WSRF and Globus Toolkit 4.

Differently from all the systems described above (except Weka4WS) that are designed to support generic workflows with particular emphasis on e-science applications, DIS3GNO has been specifically designed to support distributed data mining workflows. For this reason, DIS3GNO provides features that are specific for modelling KDD applications, which facilitate the domain-expert users. This includes specific metadata formalisms for representing and searching data mining resources, and a KDD-oriented workflow checking that ensures the design of consistent data mining applications.

Finally, DIS3GNO differs from the Weka4WS system, which is also KDD-oriented, because it is more general and extendible. In fact, Weka4WS is specifically designed to support the Weka data mining algorithms. DIS3GNO, on the contrary, can use every data mining algorithm (including the Weka ones) provided that it has been previously published in the Knowledge Grid system.

[‡‡] http://www.ibm.com/developerworks/library/specification/ws-bpel

7. CONCLUSIONS

Workflows are effective formalisms to represent data and execution flows associated with complex knowledge discovery processes and data mining tasks. The DIS3GNO system described in this paper provides a set of visual facilities to design, program, and execute distributed service-oriented data mining workflows in Grids.

The DIS3GNO GUI operates as an intermediary between the final user and the Knowledge Grid, a service-oriented system for high-performance distributed KDD. All the Knowledge Grid services for metadata and execution management are accessed transparently by DIS3GNO, thus allowing the domain experts to compose and run complex data mining application without worrying about the underlying infrastructure details.

The experimental evaluation carried out by executing some typical data mining patterns has demonstrated the effectiveness of the DIS3GNO system to support knowledge discovery workflows design and execution in distributed service-oriented environments. The DIS3GNO system and the Knowledge Grid framework are available as open-source software from `http://grid.deis.unical.it/kgrid`.

REFERENCES

1. Cannataro M. and Talia D. The Knowledge Grid. *Communitations of the ACM*, **46**(1), 2003; 89–93.
2. Prodromidis A. L, Chan P. K, Stolfo S. J. Meta-learning in Distributed Data Mining Systems: Issues and Approaches. In: *Advances in Distributed and Parallel Knowledge Discovery*, Kargupta H. and Chan P. (Eds.). AAAI/MIT Press, 2000; 81–87.
3. Kargupta H, Park B, Hershberger D, Johnson E. A New Perspective toward Distributed Data Mining. In: *Advances in Distributed and Parallel Knowledge Discovery*, Kargupta H. and Chan P. (Eds.). AAAI/MIT Press, 2000; 133–184.
4. Tan P. N, Steinbach M, Kumar V. Introduction to Data Mining. Addison-Wesley, 2006.
5. Chen H.-Y, Hsiung M, Lee H.-C, Yen E, Lin S. C, Wu Y.-T. GVSS: A High Throughput Drug Discovery Service of Avian Flu and Dengue Fever for EGEE and EUAsiaGrid. *Journal of Grid Computing*, **8**(4), 2010; 529–541.
6. Natrajan A, Crowley M, Wilkins-Diehr N, Humphrey M. A, Fox A. D, Grimshaw A. S, Books III C. L.. Studying protein folding on the Grid: experiences using CHARMM on NPACI resources under Legion. *Concurrency and Computation: Practice & Experience*, **16**(4), 2004; 385–397.
7. Laganà A, Costantini A, Gervasi O, Faginas Lago N, Manuali C, Rampino S. COMPCHEM: Progress Towards GEMS a Grid Empowered Molecular Simulator and Beyond. *Journal of Grid Computing*, **8**(4), 2010; 571–586.
8. Andreeva J, Campana S, Fanzago F, Herrala J. High-Energy Physics on the Grid: the ATLAS and CMS Experience. *Journal of Grid Computing*, **6**(1), 2008; 3–13.
9. Lagouvardos K, Floros E, Kotroni V. A Grid-Enabled Regional-Scale Ensemble Forecasting System in the Mediterranean Area. *Journal of Grid Computing*, **8**(2), 2010; 181–197.
10. Faerman M, Moore R, Cui Y, Hu Y, Zhu J, Minster B, Maechling P. Managing Large Scale Data for Earthquake Simulations. *Journal of Grid Computing*, **5**(3), 2007; 295–302.
11. Moore R. Knowledge-based Grids. Proc. *18th IEEE Symposium on Mass Storage Systems and 9th Goddard Conference on Mass Storage Systems and Technologies*, 2001.
12. Congiusta A, Talia D, Trunfio P. Using Grids for Distributed Knowledge Discovery. In: *Mathematical Methods for Knowledge Discovery and Data Mining*, Felici G. and Vercellis C. (Eds.). IGI Global, 2007; 284–298.
13. Talia D, Trunfio P. How Distributed Data Mining Tasks can Thrive as Knowledge Services. *Communications of the ACM*, **53**(7), 2010; 132–137.

14. Stankovski V, Swain M. T, Kravtsov V, Niessen T, Wegener D, Kindermann J, Dubitzky W. Grid-enabling data mining applications with DataMiningGrid: An architectural perspective. *Future Generation Computer Systems*, **24**(4), 2008; 259–279.
15. AlSairafi S, Emmanouil F.-S, Ghanem M, Giannadakis N, Guo Y, Kalaitzopoulos D, Osmond M, Rowe A, Syed J, Wendel P. The Design of Discovery Net: Towards Open Grid Services for Knowledge Discovery. *Int. Journal of High Performance Computing Applications*, **17**(3), 2003; 297–315.
16. Brezany P, Hofer J, Tjoa A. M, WoehrerA. GridMiner: An Infrastructure for Data Mining on Computational Grids. *Proc. APAC Conference and Exhibition on Advanced Computing, Grid Applications and eResearch*, 2003.
17. Mastroianni C, Talia D, Trunfio P. Metadata for Managing Grid Resources in Data Mining Applications. *Journal of Grid Computing*, **2**(1), 2004; 85–102.
18. Congiusta A, Talia D, Trunfio P. Distributed data mining services leveraging WSRF. *Future Generation Computer Systems*, **23**(1), 2007; 34–41.
19. Foster I. Globus Toolkit Version 4: Software for service-oriented systems. *Proc. Conf. on Network and Parallel Computing*, 2005; 2–13.
20. Witten H, Frank E. Data Mining: Practical machine learning tools with Java implementations. Morgan Kaufmann, 2000.
21. Van Der Aalst W. M. P, Ter Hofstede A. H. M, Kiepuszewski B, and Barros A. P. Workflow Patterns. Distrib. Parallel Databases **14**(1), 2003; 5–51.
22. Zhoud Z. H, Li M. Semi-supervised learning by disagreement. Knowledge and Information Systems **24**(3), 2010; 415–439.
23. Ripeanu M, Iamnitchi A, Foster I. Mapping the Gnutella Network. *IEEE Internet Computing*, **6**(1), 2002; 50–57.
24. Fahringer T, Jugravu A, Pllana S, Prodan R, Seragiotto Junior C, Truong H. L. ASKALON: A Tool Set for Cluster and Grid Computing. *Concurrency and Computation: Practice & Experience*, **17**(2–4), 2005; 143–169.
25. Altintas I, Berkley C, Jaeger E, Jones M, Ludascher B, Mock S. Kepler: an extensible system for design and execution of scientific workflows. *Proc. 16th International Conference on Scientific and Statistical Database Management*, 2004.
26. Deelman E, Blythe J, Gil Y, Kesselman C, Mehta G, Patil S, Su M.-H, Vahi K, Livny M. Pegasus: Mapping Scientific Workflows onto the Grid. *Proc. Across Grids Conference*, 2004.
27. Hull D, Wolstencroft K, Stevens R, Goble C, Pocock M, Li P, Oinn T. Taverna: a tool for building and running workflows of services. *Nucleic Acids Research*, **34**, 2006; 729–732.
28. Shields M, Taylor I. Programming Scientific and Distributed Workflow with Triana Services. *Proc. GGF10 Workshop on Workflow in Grid Systems*, 2004.
29. Talia D, Trunfio P, Verta O. The Weka4WS framework for distributed data mining in service-oriented Grids. *Concurrency and Computation: Practice & Experience*, **20**(16), 2008; 1933–1951.
30. Lackovic M, Talia D, Trunfio P. A Framework for Composing Knowledge Discovery Workflows in Grids. In: *Foundations of Computational Intelligence Vol 6: Data Mining Theoretical Foundations and Applications, Studies in Computational Intelligence*, Abraham A, Hassanien A, Carvalho A, Snel V. (Eds.). Springer, 2009.