European Research Network on Foundations, Software Infrastructures and Applications for large scale distributed, GRID and Peer-to-Peer Technologies

A Network of Excellence funded by the European Commission

Designing data analysis services in the Knowledge Grid

Eugenio Cesario
`cesario@icar.cnr.it`
ICAR-CNR, Rende (CS), Italy

Antonio Congiusta, Domenico Talia, Paolo Trunfio
{`acongiusta|talia|trunfio`}`@deis.unical.it`
University of Calabria, Rende (CS), Italy

CoreGRID Technical Report
Number TR-0118
November 26, 2007

Institute on Knowledge and Data Management

CoreGRID - Network of Excellence
URL: http://www.coregrid.net

CoreGRID is a Network of Excellence funded by the European Commission under the Sixth Framework Programme

Project no. FP6-004265

Designing data analysis services in the Knowledge Grid

Eugenio Cesario
cesario@icar.cnr.it
ICAR-CNR, Rende (CS), Italy

Antonio Congiusta, Domenico Talia, Paolo Trunfio
{acongiusta|talia|trunfio}@deis.unical.it
University of Calabria, Rende (CS), Italy

CoreGRID TR-0118
November 26, 2007

Abstract

Grid environments were originally designed for dealing with problems involving compute-intensive applications. Today, however, grids enlarged their horizon as they are going to manage large amounts of data and run business applications supporting consumers and end users. To face these new challenges, grids must support adaptive data management and data analysis applications by offering resources, services, and decentralized data access mechanisms. In particular, according to the service-oriented architecture model, data mining tasks and knowledge discovery processes can be delivered as services in grid-based infrastructures. By means of a service-based approach it is possible to define integrated services supporting distributed business intelligence tasks in grids. These services can address all the aspects involved in data mining and knowledge discovery processes: from data selection and transport, to data analysis, knowledge models representation and visualization. We worked along this direction for providing a grid-based architecture supporting distributed knowledge discovery named Knowledge Grid. This report discusses how the Knowledge Grid framework has been developed as a collection of grid services and how it can be used to develop distributed data analysis tasks and knowledge discovery processes exploiting the service-oriented architecture model.

1 Introduction

Computer science applications are becoming more and more network centric, ubiquitous, knowledge intensive, and computing demanding. This trend will result soon in an ecosystem of pervasive applications and services that professionals and end users can exploit everywhere. A long term perspective can be envisioned where a collection of services and applications will be accessed and used as public utilities, like water, gas and electricity are used today.

Key technologies for implementing that perspective vision are SOA and Web services, semantic Web and ontologies, pervasive computing, P2P systems, grid computing, ambient intelligence architectures, data mining and knowledge discovery tools, Web 2.0 facilities, mashup tools, and decentralized programming models. In fact, it is mandatory to develop solutions that integrate some or many of those technologies to provide future knowledge-intensive software utilities.

In the area of grid computing a proposed approach in accordance with the trend outlined above is the *service-oriented knowledge utilities (SOKU)* model [16] that envisions the integrated use of a set of technologies that are considered as a solution to information, knowledge and communication needs of many knowledge-based industrial and business applications. The SOKU approach stems from the necessity of providing knowledge and processing capabilities to everybody, thus promoting the advent of a competitive knowledge-based economy. The SOKU model captures three key notions:

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

- Service Oriented: services which must be instantiated and assembled dynamically, so that the structure, behavior and location of software is changing at run-time;
- Knowledge: SOKU services are knowledge-assisted to facilitate automation and advanced functionality;
- Utility: directly and immediately useable services with established functionality, performance and dependability; the emphasis is on user needs and issues such as trust, ease of use, and standard interface.

Although the SOKU model is not yet implemented, grids are increasingly equipped with data management tools, semantic technologies, complex workflows, data mining features and other Web intelligence approaches. These technologies can facilitate the process of having grids as a strategic element for pervasive knowledge intensive applications and utilities.

Grids were originally designed for dealing with problems involving large amounts of data or compute-intensive applications. Today, however, grids enlarged their horizon as they are going to run business applications supporting consumers and end users [8]. To face those new challenges, grid environments must support adaptive data management and data analysis applications by offering resources, services, and decentralized data access mechanisms. In particular, according to the service-oriented architecture model, data mining tasks and knowledge discovery processes can be delivered as services in grid-based infrastructures.

Through a service-based approach it is possible to define integrated services for supporting distributed business intelligence tasks in grids. These services can address all the aspects involved in data mining and in knowledge discovery processes: from data selection and transport, to data analysis, knowledge models representation and visualization. Along this direction, a grid-based architecture supporting distributed knowledge discovery is the *Knowledge Grid* [7, 9]. This report discusses how the Knowledge Grid framework has been developed as a collection of grid services and how it can be used to develop distributed data analysis tasks and knowledge discovery processes using the SOA model.

The rest of the report is organized as follows. Section 2 introduces the Knowledge Grid approach for the design of distributed knowledge discovery services. Section 3 describes the services implemented in the Knowledge Grid framework. Section 4 describes the rationale and features of the data analysis services as higher-level services built on top of the basic Knowledge Grid services. Section 5 discusses models and tools for designing service-based data mining tasks and some application examples. Finally, Section 6 concludes the report.

2 Approach

The Knowledge Grid framework makes use of basic grid services to build more specific services supporting distributed knowledge discovery in databases (KDD) on the grid [7]. Such services allow users to implement knowledge discovery applications that involve data, software, and computational resources available from distributed grid sites. To this end, the Knowledge Grid defines mechanisms and higher-level services for publishing and searching information about resources, representing and managing KDD applications, as well as managing their output results.

Previous work on the Knowledge Grid has been focused on the development of the system by using early grid middleware [6], as well as the design and evaluation of distributed KDD applications [6, 4, 5]. The *Web services resource framework* (*WSRF*) has been adopted for re-implementing the Knowledge Grid services following the SOA approach [9]. Such services will be used either to execute distributed data mining applications or to build more specific data analysis services, which can be in turn exploited to compose and execute higher-level KDD applications.

Such an approach can be described through the layered architecture shown in Figure 1, which represents the relationship among basic grid services, Knowledge Grid services, data analysis services, and KDD applications:

- **Basic grid services**: core functionalities provided by standard grid environments such as Globus Toolkit [14], UNICORE [12], and gLite [13]. Such functionalities include *security services* (mechanisms for authentication, authorization, cryptography, etc.), *data management services* (data access, file transfer, replica management, etc.), *execution management services* (resource allocation, process creation, etc.), and *information services* (resource representation, discovery, and monitoring).
- **Knowledge Grid services**: services specifically designed to support the implementation of data mining services and applications. They include *resource management services*, which provide mechanisms to describe, publish, and retrieve information about data sources, data mining algorithms, and computing resources, and *execution*

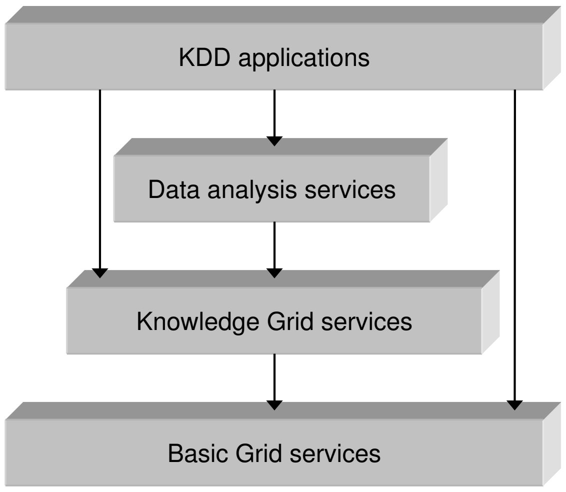


Figure 1: The Knowledge Grid layered approach.

management services, which allow users to design and execute distributed KDD applications. Both resource and execution management services include core-level components and higher-level services, as described in Section 3

- **Data analysis services**: ad hoc services that exploit the Knowledge Grid services to provide high-level data analysis functionalities. A data analysis service can expose either a single data pre-processing or data mining task (e.g., classification, clustering, etc.), or a more complex knowledge discovery task (e.g., parallel classification, meta-learning, etc.). Data analysis services are discussed in Section 4.
- **KDD applications**: knowledge discovery applications built upon the functionalities provided by the underlying grid environments, the Knowledge Grid framework, or higher-level data analysis services. Different models, languages, and tools can be used to design distributed KDD applications in this framework, as discussed in Section 5.

The discussed architectural approach is flexible in allowing KDD applications to be built upon data analysis services and upon Knowledge Grid services. Moreover, those services can be composed together with basic services provided by a generic grid toolkit to develop KDD applications.

3 Knowledge Grid services

The Knowledge Grid is a software framework for implementing knowledge discovery tasks in a wide range of high-performance distributed applications. It offers to users high-level abstractions and a set of services by which they can integrate grid resources to support all the phases of the knowledge discovery process.

The Knowledge Grid architecture is designed according to the *service-oriented architecture (SOA)*, which is a model for building flexible, modular, and interoperable software applications. The key aspect of SOA is the concept of *service*, a software block capable of performing a given task or business function. Each service operates by adhering to a well defined interface, defining required parameters and the nature of the result. Once defined and deployed, services are like “black boxes” that is, they work independently of the state of any other service defined within the system, often cooperating with other services to achieve a common goal. The most important implementation of SOA is represented by Web services, whose popularity is mainly due to the adoption of universally accepted technologies such as XML, SOAP, and HTTP. Also the grid provides a framework whereby a great number of services can be dynamically located, balanced, and managed, so that applications are always guaranteed to be securely executed, according to the principles of on-demand computing.

The grid community has adopted the *open grid services architecture (OGSA)* as an implementation of the SOA model within the grid context. In OGSA every resource is represented as a Web service that conforms to a set of conventions and supports standard interfaces. OGSA provides a well-defined set of Web service interfaces for the development of interoperable grid systems and applications [15]. Recently the WSRF has been adopted as an evolution of early OGSA implementations [10]. WSRF defines a family of technical specifications for accessing and managing stateful resources using Web services. The composition of a Web service and a stateful resource is termed as WS resource. The possibility to define a state associated to a service is the most important difference between WSRF-compliant Web services, and pre-WSRF ones. This is a key feature in designing grid applications, since WS resources provide a way to represent, advertise, and access properties related to both computational resources and applications.

3.1 The Knowledge Grid architecture

Basically, the Knowledge Grid architecture is composed of two groups of services, classified on the basis of their roles and functionalities. Indeed, two main aspects characterize a knowledge discovery process performed in accordance with the Knowledge Grid philosophy. The first is the management of data sources, data sets and tools to be employed in the whole process. The second is concerned with the design and management of a knowledge flow that is the sequence of steps to be executed in order to perform a complete knowledge discovery process by exploiting the advantages coming from a grid environment. Notice that such a division has also a functional meaning, because a Knowledge Grid user first looks for data sets and tools to be used in the knowledge discovery process, then defines the knowledge flow to be executed (by using the resources found in the first step). On the basis of such rationale, there are two distinct groups of services composing the Knowledge Grid architecture: the *Resource Management Services* and the *Execution Management Services*.

The *Resource Management Services (RMSs)* group is composed of services that are devoted to the management of resources, like data sets, tools and algorithms involved in a KDD process performed by the Knowledge Grid, as well as models inferred by previous analysis. Due to the heterogeneity of such resources, metadata represent a suitable descriptive model for them, in order to provide information about their features and their effective use. In this way, all the relevant objects for knowledge discovery applications, such as data sources, data mining software, tools for data preprocessing, results of computation, are described by their metadata. RMSs comprise services that make a user able to store resource metadata and perform different kind of resource discovery tasks. Such tasks may typically include queries on resources availability, resource location, as well as their access properties.

The *Execution Management Services (EMSs)* group is concerned with the design and management of the knowledge discovery flow. A knowledge flow in the Knowledge Grid is modeled by a so-called *execution plan* that can be represented by a direct graph describing interactions and data flows between data sources, extraction tools, data mining tools and visualization tools. The execution graph describes the whole flow of the knowledge discovery process, the resources involved, the data mining steps, etc. The EMS module contains services that are devoted to create an execution plan, to elaborate and to execute it, on the basis of the available resources and the basic grid services running on the host. It is worth noticing that when the user submits a knowledge discovery application to the Knowledge Grid, she has no knowledge about all the low-level details needed by the execution plan. More precisely, the client submits to the Knowledge Grid a high-level description of the KDD application, named *conceptual model*, more targeted to distributed knowledge discovery aspects than to grid-related issues. A specific function of the EMS is to create an execution plan on the basis of the conceptual model received from the user, and execute it by using the resources effectively available. To realize this logic, the EMS follows a two-step approach: it initially models an *abstract execution plan* that in a second step is resolved into a *concrete execution plan*.

Figure 2 shows the architecture of the Knowledge Grid. A client application that wants to submit a knowledge discovery computation to the Knowledge Grid, does not need to interact with all of these services, but just with a few of them. Inside each group, in fact, there are two layers of services: *high-level services* and *core-level services*. The design idea is that user-level applications directly interact with high-level services that, in order to perform 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. In other words, operations exported by high-level services are designed to be invoked by user-level applications, whereas operations provided by core-level services are thought to be invoked both by high-level and core-level services.

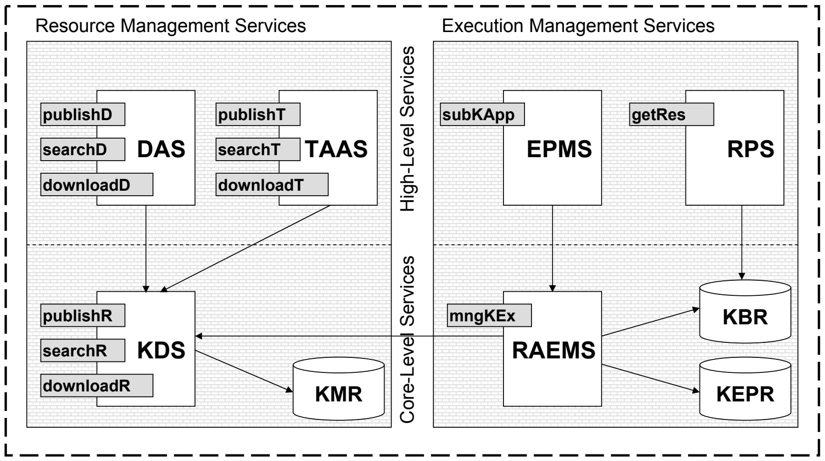


Figure 2: The Knowledge Grid architecture.

3.1.1 Client Interface

Even though the client component is not a module integrated in the Knowledge Grid, it is useful to introduce its role and interaction with the architecture in order to better describe functionalities of the whole framework. A user of the Knowledge Grid can request the execution of basic tasks (e.g searching of data and software, data transfers, simple job executions), as well as distributed data mining applications described by arbitrarily complex conceptual models/execution plans. As previously mentioned, a user-level application performs its tasks by invoking the appropriate operations provided by the different high-level services. Generally, a client running on a grid node may invoke services that are running on a different grid node; therefore, the interactions between the client interface and high-level services are possibly remote.

3.1.2 Resource Management Services

This group of services includes basic and high-level functionalities for managing different kinds of Knowledge Grid resources. Among such resources, data sources and algorithms are of main importance, for this reason the Knowledge Grid architecture provides ad-hoc components, namely *DAS* and *TAAS*, for dealing with data, tools, and algorithms.

DAS. The *Data Access Service* is concerned with the publishing, searching and transferring of data sets to be used in a KDD application, as well as the search for inferred models (resulting from the mining process). The DAS exports the `publishData`, `searchData` and `downloadData` operations. The `publishData` is invoked by a user-level application to publish metadata about a data set; as soon as a publish operation is requested, it passes the corresponding metadata to the local *KDS* by invoking the `publishResource` operation. The `searchData` operation is invoked by a client interface that needs to locate a data set on the basis of a specified set of criteria. The DAS submits its request to the local KDS, by invoking the corresponding `searchResource` and, as soon as the search is completed, it receives the result from the KDS. Such a result consists in a set of references to the data sets matching the specified search criteria. It is worth noticing that the search operation is not just handled on the local host, but it is remotely forwarded also to other hosts. Finally, the `downloadData` operation works similarly to the previous ones: it receives the reference of the data set to be download and forwards the request to the `downloadResource` of the local KDS component. The DAS is, together with the TAAS, a high-level service of the RMS group.

TAAS. The *Tools and Algorithms Access Service* is concerned with the publishing, searching and transferring of tools to be used in a knowledge discovery application. Such tools can be extraction tools, data mining tools and visualization tools. It has the same basic structure as the DAS and performs the main tasks by interacting with a local instance of the KDS, which in turn may invoke one or more other remote KDS instances. The operations that are exported by the TAAS are `publishTool`, `searchTool` and `downloadTool`. They have similar functionalities and similar behavior as DAS operations, with the difference that TAAS operations are concerned with tools and not with data.

KDS. The *Knowledge Directory Service* is the only core-level service of the RMS group. It manages metadata describing Knowledge Grid resources. Such resources comprise hosts, repositories of data to be mined, tools and algorithms used to extract, analyze and manipulate data, knowledge models obtained as results of mining processes.

Such information are stored in a local repository, the *Knowledge Metadata Repository* (*KMR*). The KDS exports the `publishResource`, `searchResource` and `downloadResource` operations. As previously noticed, they are invoked by the high-level services DAS or TAAS (to perform operations on data sets or tools respectively). The `publishResource` operation is invoked to publish information (metadata) about a resource; the publish operation is made effective by storing their metadata in the local KMR. The `searchResource` operation is invoked to retrieve a resource on the basis of a given set of criteria represented by a query. An important aspect to be pointed out is that the KDS performs such a searching task both locally, by accessing the local KMR, and remotely, by querying other remote KDSs (that in turn will access their local KMR). The `downloadResource` operation asks that a given resource is downloaded to the local site; also in this case, if the requested resource is not stored on the local site, the KDS will forward such a request to other remote KDS instances.

3.1.3 Execution Management Services

Services belonging to this group have the functionality of allowing the user to design and execute KDD applications, as well as delivering and visualizing the computation result.

EPMS. The *Execution Plan Management Service* allows for defining the structure of an application by building the corresponding execution graph and adding a set of constraints about resources. This service, on the basis of a *conceptual model* received from the client, generates its corresponding *abstract execution plan* that is a more formal representation of the structure of the application. Generally, it does not contain information on the physical grid resources to be used, but rather constraints and other criteria about them. Nevertheless, it can include both well identified resources and abstract resources, i.e., resources that are defined through logical names and quality-of-service requirements. The EPMS exports the `submitKApplication` operation, through which it receives a conceptual model of the application to be executed and transforms it into an abstract execution plan for subsequent processing by the RAEMS. The mapping between the abstract execution plan and a more concrete representation of it, as well as its execution, is delegated to the RAEMS.

RPS. The *Results Presentation Service* offers facilities for presenting and visualizing the extracted knowledge models (e.g., association rules, clustering models, and decision trees), as well as to store them in a suitable format for future reference. It is pointed out that the results stage-out, often needed for delivering results to the client, is an important task of this service. Indeed, a user can remotely interacts with the Knowledge Grid and, as soon as the final result is computed, it is delivered to the client host and made available for visualization.

RAEMS. The *Resource Allocation and Execution Management Service* is used to find a suitable mapping between an abstract execution plan (received from the EPMS) and available resources, with the goal of satisfying the constraints (CPU, storage, memory, database, network bandwidth requirements) imposed by the execution plan. The output of this process is a concrete execution plan, which explicitly defines the resource requests for each data mining process. In particular, it matches requirements specified in the abstract execution plan with real names, location of services, data files, etc. From the interface viewpoint, the RAEMS exports the `manageKExecution` operation, which is invoked by the EPMS and receives the abstract execution plan. Starting from it, the RAEMS queries the local KDS (through the `searchResource` operation) to obtain information about the resources needed to create a concrete execution plan from the abstract execution plan. As soon as the concrete execution plan is obtained, the RAEMS coordinates the actual execution of the overall computation. For this purpose, the RAEMS invokes the appropriate services (data mining services and basic grid services) as specified by the concrete execution plan. As soon as the computation terminates, the RAEMS stores its results into the *Knowledge Base Repository (KBR)*, while the execution plans are stored into the *Knowledge Execution Plan Repository (KEPR)*. As a final operation, in order to publish results obtained by the computation, the RAEMS publishes their metadata (result metadata) into the KMR (by invoking the `publishResource` operation of the local KDS).

3.2 Implementation

This section describes the WSRF implementation of the Knowledge Grid, pointing out the adopted technologies, the service structure, along with the main implemented mechanisms.

The Knowledge Grid architecture uses grid functionalities to build specific knowledge discovery services. Such services can be implemented by using different available grid environments, such as Globus Toolkit[1], UNICORE[2],

[1]http://www.globus.org
[2]http://www.unicore.org

and gLite[3]. The Knowledge Grid framework and its services are being developed by using the Globus Toolkit [14]. The Globus Toolkit, currently at version 4, is an open source framework provided by the Globus Alliance for building computational grids based on WSRF. Currently, a growing number of projects and companies are using it to develop grid services and to exploit grid potentials. Such a phenomenon is contributing to the diffusion of the framework within the international community.

As a preliminary consideration, it has to be noticed that the implementation of each Knowledge Grid service follows the *Web service resource factory pattern*, a well known design pattern in software design and especially in object-oriented programming. Such a pattern, recommended by the WSRF specifications, requires having one service, the *factory service*, in charge of creating the resources and another one, the *instance service*, to actually operate on them. A resource maintains the state (e.g., results) of the computation performed by the service specific operations. The factory service provides a `createResource` operation that is used (by a client) to create a stateful resource. As soon as a new resource has been created, the factory service returns an *endpoint reference* that identifies univocally the resource for successive accesses to it. The instance service exports all the other methods useful for the functionalities of the service, by accessing the resources through their endpoint reference. Both services interact with a third external entity (not a service), the *Resource Home*, in charge of managing the resources (creates them and returns their reference). For completeness, it has to be specified that each service of the Knowledge Grid, in addition to the service specific operations and the aforementioned `createResource`, exports two other operations: `destroy` and `subscribe`. Even though they are not reported in Figure 2 and were not mentioned before, such operations are part of the service interface as well as other methods explicitly reported. The `destroy` operation is used to destroy a resource (previously created by `createResource`). The `subscribe` operation subscribes a client for notifications about a specific published topic.

In order to implement services in a highly-decentralized way, a common design pattern is adopted. It allows a client to be notified when interesting events occur on the service side. The *WS-Notification* specification defines a *publish-subscribe* notification model for Web services, which is exploited to notify interested clients and services about changes that occur to the status of a WS resource. In other words, a service can *publish* a set of topics that a client can subscribe to; as soon as the topic changes, the client receives a notification of the new status.

3.2.1 Resource Management Services.

The RMSs group contains a set of services devoted to the publishing, searching and downloading of resources (data sets, tools, etc.) involved in Knowledge Grid applications. In particular, the high-level DAS and TAAS services interact, through the core-level KDS service, with the KMR repository to manage metadata about data sets and algorithms respectively.

KDS, **DAS**, and **TAAS**. The KDS, DAS and TAAS have been implemented according to the factory-instance pattern. Therefore, the development of each Knowledge Grid service is split into two basic services: factory service and instance service. All the services are implemented in Java. The service interfaces have been described in Section 2, as well as their qualitative functionalities.

The interactions between a client, the DAS (or TAAS), the KDS, and the KMR to manage a publish, search, or download operation are pictorially presented in Figure 3. When a client needs to publish a data set it invokes the `publishData` operation of the DAS, passing the URL of the metadata describing the data set. In turn, the DAS forwards such a request to the KDS by invoking its `publishResource` operation. The KDS stores the metadata info into the KMR, by adding an entry corresponding to the newly available resource. There is one resource property associated to each service: the *DAS_RP* is the resource property of the DAS, whereas the *KDS_RP* is the resource property of the KDS. Both such resource properties are string variables. Additionally, they are also published in the *topic list*, in order to be available as "items of interest for subscription" for notification purposes. The sample scenario reproduce a general case where the client is running on Node A, while a Knowledge Grid instance (the client directly interacts with) is running on Node B. Moreover, there are some Knowledge Grid instances running on different nodes (W, X, Y, and Z).

Whenever a client wants to submit the request, it invokes the `createResource` operation of the DAS to create a resource property DAS_RP, subscribes a listener (a Java class inside the client) as consumer of notifications on the topic DAS_RP (through the DAS `subscribe` method), and invokes the `publishData` operation exported by the DAS (step 1). In turn, the DAS dispatches this operation in a similar way: it subscribes a listener as consumer of notifications

[3]http://www.cern.ch/glite

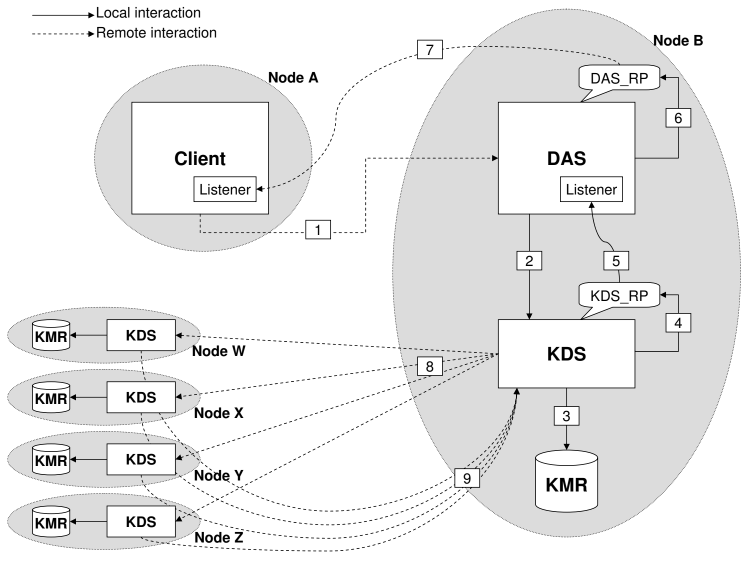


Figure 3: Services interactions for resource management.

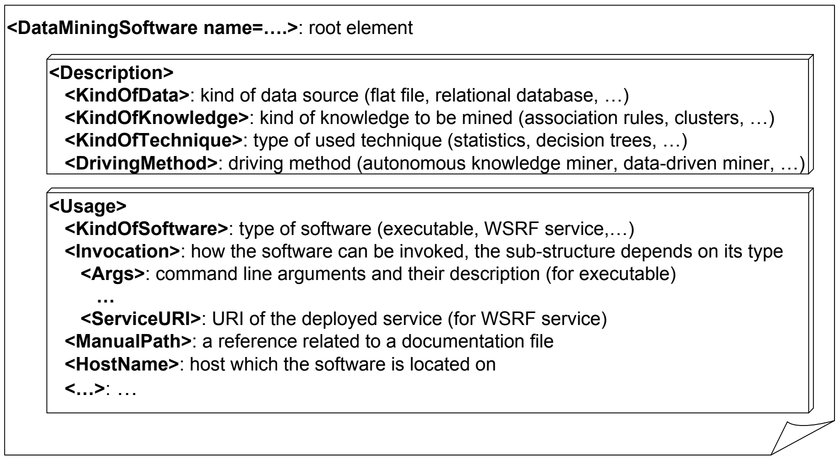
on the resource property KDS_RP (after having created it), and invokes the local KDS `publishResource` operation (step 2). As consequence of this invocation, the metadata file referred to by the URL (and describing the data set to be published) is inserted into the KMR (step 3). As soon as such operation is terminated, the value of the KDS_RP is set by the *kdsURL* of the published resource (step 4), or it assumes an error message, if an error occurs (e.g., the metadata file is corrupted, the resource has been already published, etc.). The kdsURL is a reference identifying a published resource in the KMR. The change in the KDS_RP value is automatically notified to the *DAS Listener* (step 5), which in turn stores such value on the DAS_RP (step 6) and asks the destroying of KDS_RP. The value change of DAS_RP throws a notification for the *Client Listener* (step 7), and finally to the client.

The operation of publishing a tool is similar to that described above, with the difference that the TAAS is involved, instead of the DAS. Analogously, also the search and download operations are executed in a similar way, with the addition that the KDS forwards (step 8) the requests also to other remote KDSs and waits for their response (step 9). For completeness, it is worth noticing that in the case of a download request, the data transferring is handled by a request to the *RFT service*, provided as basic service by the Globus Toolkit.

KMR. The KMR stores metadata needed to identify and classify all the heterogeneous resources used in data mining applications on grids. They include data mining software and data sources.

Data Mining Software. There are two main aspects in the definition of metadata describing a data mining algorithm: its *description* (what the algorithm does) and its *usage* (how it can be invoked). Data mining software can be categorized on the basis of the following parameters: the kind of input data source, the kind of knowledge to be discovered, the type of techniques used, and the driving method of the mining process [19]. For what concerns its usage, metadata should specify if it is an executable file or it is an available WSRF service, and the parameters to be used for its invocation. Such information can be mapped on an XML schema which defines the format and the syntax of the XML file that is used to describe the features of a generic data mining software. In Figure 4 is reported a sketch of the XML file containing such metadata information. It is composed of two parts. The first part is the software description, containing some tags specifying the features of the data mining algorithm as mentioned above. The second part is the software usage, describing how to invoke or execute the algorithm. In particular, the `<invocation>` subsection specifies the way the software can be invoked. This feature depends on the kind of software: if it is in the form of an executable, this section specifies details of the software invocation (executable name and its arguments), whereas in the case of a software available through a service, within the tag is specified the service URI (from which is possible to retrieve the WSDL file and information about the service interface).

Data Source. Data sources are analysed by data mining algorithms to extract knowledge from them. They can originate from text files, relational databases, Web pages and other semi-structured documents. Metadata information describing


<DataMiningSoftware name=….>: root element

<Description>
<KindOfData>: kind of data source (flat file, relational database, …)
<KindOfKnowledge>: kind of knowledge to be mined (association rules, clusters, …)
<KindOfTechnique>: type of used technique (statistics, decision trees, …)
<DrivingMethod>: driving method (autonomous knowledge miner, data-driven miner, …)

<Usage>
<KindOfSoftware>: type of software (executable, WSRF service,…)
<Invocation>: how the software can be invoked, the sub-structure depends on its type
<Args>: command line arguments and their description (for executable)
…
<ServiceURI>: URI of the deployed service (for WSRF service)
<ManualPath>: a reference related to a documentation file
<HostName>: host which the software is located on
<…>: …

Figure 4: Software Metadata Schema.

a data source should specify its *structure* (what attributes describe it, their types, etc.) and its *access* (how it is retrieved). This allows the Knowledge Grid to deal with different kind of data sources as long as they are properly characterized by appropriate metadata descriptions. For instance, data sources represented by flat files (arff[4] and csv[5] formats) are described by metadata information as detailed in [19].

After having briefly described the structure of the metadata modeling algorithms and data sources, let us briefly sketch the implementation of the KMR. Currently it is implemented as a directory of the file system in which metadata about resources are stored as XML files. In such a directory an XML file for each resource published on the local host is stored. In addition to all these files, there is also a *registry* file, containing an entry "*<LogicalName,PhysicalName>*" for each published resource. The *LogicalName* (the so called kdsURL) is in the form "*kds://<hostname>/<relative path of the document in the KMR of hostname>*", while the *PhysicalName* is in the form "*<absolute path of the document in the KMR of hostname>*". As soon as a new resource has to be published on a host, its metadata file is stored in the KMR and an entry is inserted in the registry (that makes the publishing operation effectively concluded). As an evolution of such current solution based on file system, there is an ongoing implementation of a storage back-end based on database. Both relational (PostgreSQL, MySQL) and XML (XIndice, Exist) databases have been chosen, in order to evaluate and compare their suitability and efficiency.

3.2.2 Execution Management Services.

Interfaces and communication patterns involving such services have been defined, the implementation of the software is in a preliminary status. As previously mentioned, applications are described at high level using a *conceptual model* generated by the user (e.g., through design facilities provided by the user-level module). It must specify, at some level of abstraction, the tasks that compose the process, as well as the logic ruling their integration. In the current approach *UML activity diagrams* are used to represent the conceptual model of an application. Such an activity diagram represents the high-level flow of service invocations that constitute the application logic. The *conceptual model* is then passed to the EPMS, which is in charge of transforming it into an *abstract execution plan* for subsequent processing by the RAEMS. The execution plan is represented by *business process execution language (BPEL)* [2], that is an XML-based language to express the business logic of the application being modeled. The abstract execution plan may not refer to specific implementations of services, application components and input data. The RAEMS receives the abstract execution plan and creates a *concrete execution plan*. In order to do such a task, the RAEMS needs to evaluate and resolve a set of grid resources, by contacting the KDS and choosing the most appropriate ones. Another important difference between the abstract and concrete execution plans is that in the first the WSDL of services are used without specifying the service location, while in the second the location on which to execute the service is concretely decided (w.r.t. locations currently available, processor speed, quality of service, etc.). In other words, the RAEMS attempts to locate suitable service instances that match the service requirements specified as WSDL definitions. As

[4]attribute-relation file format
[5]comma-separated values

soon as the RAEMS has built the concrete execution plan, it is in charge of submitting it to the workflow engine that coordinates services execution. The RAEMS and the workflow engine cooperate for managing the execution of the overall application taking into account relevant scheduling decisions. Such decisions are generated by specific sub-components of the RAEMS that monitor the status of the involved grid resources and jobs, and enforce optimization policies also based on appropriate heuristics (more details are provided in [21]).

Whenever a legacy software is to be integrated within the execution plan, this is obtained by including the submission of a grid job to the GRAM service of the Globus Toolkit. In order to do that, the RAEMS takes care of generating the needed *resource specification language* document to be referred by the concrete execution plan as a GRAM service parameter. Finally, the notification mechanism will be used to send back information to the client about the execution status of the running jobs.

4 Data Analysis services

Distributed data mining includes a variety of application scenarios relying on many different patterns, techniques, and approaches. Their implementation may involve multiple independent or correlated tasks, as well as sharing of distributed data and knowledge models [17].

An example of distributed data mining technique is *meta-learning*, which aims to build a global classifier from a set of inherently distributed data sources [20]. 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 data sets in parallel. Then, the classifiers computed by learning programs are collected in a centralized site and combined to obtain the global classifier.

Another example of distributed data mining is *collective data mining* [18]. Instead of combining incomplete local models, collective data mining builds the global model through the identification of significant sets of local information. In other terms, the local blocks directly compose the global model. This result is based on the fact that any function can be expressed in a distributed fashion using a set of appropriate basis functions. If the basis functions are orthogonal the local analysis generates results that can be correctly used as components of the global model.

When such kind of applications are to be deployed on a grid, in order to fully exploit the advantages of the environment, many issues must be faced, including heterogeneity, resource allocation, and execution coordination. The Knowledge Grid provides services and functionalities that permit to cope with most of the above mentioned issues and to design complex applications using high-level abstractions such as service composition facilities (i.e., workflows).

In some cases, however, end users are not familiar with distributed data mining patterns and complex workflow composition. Therefore, higher-level data analysis services, exposing common distributed data mining patterns as single services, might be useful to broaden the accessibility of the Knowledge Grid features.

In general, data analysis services may implement either:

- single steps of the overall KDD process, or
- complex KDD patterns involving multiple independent or correlated data mining tasks.

The former include tasks such as data filtering, classification, clustering, and association rules discovery. The latter refer to patterns such as parallel classification, meta-learning, collective data mining, and so on. Both types of data analysis services can be in turn used as basis for other data analysis services, or directly employed in the implementation of distributed KDD applications over the Knowledge Grid.

As an example scenario in which data analysis services may be profitably employed, let us consider the case of a user willing to perform a meta-learning analysis over distributed data. In order to execute this application on the Knowledge Grid, the user must design the overall application by defining a workflow that coordinates the execution of different classifiers on multiple nodes, followed by a final combining of the models to obtain a global classifier.

While this task can be performed using the Knowledge Grid facilities (see Section 5), it requires some expertise that in some cases is not owned by the final user. Thus, a specific data analysis service, here called *meta-learning service (MLS)*, could be provided to support seamless execution of meta-learning applications over the Knowledge Grid.

Like the underlying Knowledge Grid services, the MLS data analysis service should be implemented as a WSRF-compliant Web service. Among the other WSRF-specific operations, the MLS interface should provide a `submitMLApplication` operation for submitting the meta-learning task, with parameters specifying the URLs of the data sources to be analyzed, the names of the classifiers algorithms to be used, and the name of the combining algorithm.

Note that the MLS `submitMLApplication` differs from the EPMS `submitKApplication`, because the latter receives a conceptual model of generic KDD application, while the former receives parameters exclusively targeted to the execution of a meta-learning task. The role of the MLS is thus to translate the request submitted to its `submitMLApplication` into a request for the `submitKApplication` of the EPMS, releasing the user from the task of designing the conceptual model of the application.

By exploiting the MLS, the meta-learning application could be thus executed through the following steps:

1. The user invokes `submitMLApplication` of the MLS passing the URLs of the data sources to be analyzed and the names of the algorithms to be used.

2. The MLS defines the appropriate conceptual model for the application, and passes it to EPMS of a Knowledge Grid node.

3. The Knowledge Grid services then proceed by: $i)$ generating the abstract execution plan; $ii)$ creating the concrete execution plan by finding the needed algorithms and resources; $iii)$ coordinating the overall execution; $iv)$ passing the results to the MLS.

4. The MLS then returns the obtained results (i.e., the global model inferred by the meta-learning process) to the submitting user.

While in the example above the MLS acts as a broker between end users and the Knowledge Grid services, in other cases data analysis services may in turn invoke lower-level data analysis services (e.g, simple classification services). In general, the use of data analysis services can foster the reuse of KDD applications and patterns over the Knowledge Grid, by fully exploiting the service composition paradigm.

5 Design of Knowledge Grid Applications

The knowledge discovery design phase is the process of selecting data sets, algorithms, and related configurations for performing the needed steps leading to the extraction of knowledge models from data. To such purpose, a task-flow involving the selected data sets and algorithms can be used for specifying how the several activities of the knowledge discovery process are organized. Such a task-flow typically involves several atomic tasks (such as pre-processing, classification, etc.), to be performed sequentially and/or concurrently (and in general on different grid nodes) with specified relationships among them, as well as data constraints.

The application design is also a fundamental process through which it is possible to exploit the facilities offered by the underlaying grid environment. The application design process in the Knowledge Grid can be described as follows.

- Starting points are the resource descriptions provided by the Knowledge Grid information system (namely DAS and TAAS services).

- It is necessary to integrate resources in a task-flow (or workflow) representing the structure of the application. This step includes the possible addition of abstract tasks. The way the task-flow is expressed is an abstract and generic way, as to assure flexibility, easiness of design and comprehension, and portability over time and space of the application model. Such model is referred to as conceptual model.

As already discussed, the design phase is followed by an application-management phase, in which the application model is processed by the EPMS and the RAEMS for refining the specification, generating execution plans, and submitting the application for execution.

Designing and executing a distributed KDD application within the Knowledge Grid is a multi-step process that involves interactions and information flows between services at the different levels of the architecture. A key aspect in the Knowledge Grid is how applications are modeled, and how the application models are represented and processed through the different services.

As already mentioned, applications are described at a high level using a conceptual model, generated, for instance, by the user through the design facilities provided by the client interface. The conceptual model (or application model) must specify, at some level of abstraction, the tasks that compose the application, as well as the logic ruling their integration.

The Knowledge Grid embraces user-friendliness as a basic design principle. For this reason significant efforts have been devoted to the provision of design tools and abstractions able to fill the gap between the user perspective and the grid infrastructure. The application design support offered by the Knowledge Grid has evolved over the time from a basic visual language to a UML-based grid-workflow formalism.

5.1 The VEGA visual language

VEGA has been the first programming interface of the Knowledge Grid [6]. The graphical interface provided by VEGA proposes to users a set of graphical objects representing grid resources. These objects can be manipulated using visual facilities allowing for inserting links among them, thus composing a graphical representation of the set of jobs included in the application. A complex application is generally composed by several jobs, some of which must be executed concurrently and others sequentially. To reflect this, VEGA provides the *workspace* abstraction. An application is designed as a sequence of workspaces which are executed sequentially; while each workspace contains a set of jobs to be executed concurrently.

The insertion of links between resources must follow precise rules, in order to produce logically correct application models. Table 1 summarizes such basic composition rules.

Table 1: Composition rules in VEGA

Resource A	**Resource B**	**Link type**	**Meaning**
Data	Software	Input	Computation input
Data	Software	Output	Computation output
Data	Host	File Transfer	Data transfer to the host
Software	Host	File Transfer	Software staging to the host
Software	Host	Execute	Computation execution on the host

Before the generation of abstract and instantiated execution plans the application model is validated by VEGA in order to assert both basic composition rules and other consistency properties.

5.2 UML application modeling

In the latest implementation of the Knowledge Grid the conceptual model of an application is expressed under the form of a grid workflow, since it represents a more general way for expressing grid computations able to overcome some limitations of the VEGA model.

Many formalisms have been traditionally used for modelling workflows, such as directed acyclic graphs, Petri nets, and UML activity diagrams. Many grid workflow systems adopt standard coordination languages such as BPEL and WSCI [3], or XML-based ad-hoc solutions.

Within the Knowledge Grid, the UML-activity-diagram formalism is used to represent the conceptual model of the application, while BPEL is used for representing execution plans. The activity diagram represents the high-level flow of service invocations that constitute the application logic, whereas the BPEL expresses how the various services are actually invoked and coordinated.

The UML activity diagrams are built using typical task-graph operators such as *start*, *fork*, *join*, which rule the execution flow. As mentioned earlier, “abstract” and “concrete” execution plans are distinguished. At the abstract level, the execution plan may not refer to specific implementations of services, application components, and input data. All these entities are identified through logical names and, in some cases, by means of a set of constraints about some of their properties, possibly expressing quality of service requirements. For example, requirements on processor speed, amount of main memory or disk space can be used to single out grid nodes, while requirements on grid software may concern input data or target platforms.

Prior of the application execution, all of the resource constraints need to be evaluated and resolved on a set of available grid resources, as to choose the more appropriate ones with respect to the current status of the grid environment. Of course, due to the dynamic nature of the grid, an abstract execution plan can be instantiated into different execution plans at different times. Concrete execution plans include real names and locations of services, data files, etc. According to this approach the workflow definition is decoupled from the underlying grid configuration.

The translation of the conceptual model (represented by the UML formalism) into the abstract execution plan (represented by a BPEL document) is performed by the EPMS. To this end, the EPMS incorporates an algorithm for mapping the UML operators into corresponding BPEL notations. The BPEL notation is explicitly targeted to service invocations and thus more rich with respect to the UML one. It includes several constructs for interacting with services according to different patterns, as well as means for manipulating and accessing services input-messages and responses (both of them through explicit variable-manipulation operators and XPath expressions).

5.3 Applications and experiments

The Knowledge Grid framework has been successfully used to implement several applications in different application areas. The applications developed in such contexts have been useful for evaluating the overall system under different aspects, including performance.

The main issues and problems that have been addressed include the use of massive data sets and the distribution and coordination of the computation load among different nodes. In the remaining part of this section two applications are summarized: the clustering of human protein sequences, and the integration of a query-based data mining system within the Knowledge Grid. More detailed descriptions, along with performance evaluations, are presented in previous work [4, 5].

5.3.1 Protein folding

The implemented application carries out the clustering of human proteins sequences using the TribeMCL method [11]. TribeMCL is a clustering method through which is possible to cluster correlated proteins into groups termed "protein families." The clustering is achieved by analyzing similarity patterns between proteins in a given data set, and using these patterns to assign proteins to related groups. TribeMCL uses the Markov Clustering algorithm.

The application comprises four phases: (*i*) data selection, (*ii*) data pre-processing, (*iii*) clustering, and (*iv*) results visualization. The measurement of application execution-times has been done in two different cases: a) using only 30 human proteins, and b) using all the human proteins classified in the Swiss-Prot database[6] (a well known and widely used source of protein sequences with a high level of annotation).

Comparing the execution times, it is possible to note that the execution of the clustering phase is a computationally intensive operation, consequently it takes much more time when all the proteins have to be analyzed. The grid execution of these phases using 3 nodes has been performed in a time reduced by a factor of 2.5 with respect to the sequential execution.

5.3.2 KDD Markup Language and the Knowledge Grid

KDDML-MQL is an environment for the execution of complex KDD processes expressed as high-level queries [1]. The environment is composed of two layers, the bottom layer called KDDML and the top one called MQL. KDDML (KDD Markup Language) is an XML-based query language for performing knowledge extraction from databases. It combines, through a set of operators, data manipulation tasks such as data acquisition, pre-processing, mining and post-processing, employing specific algorithms from several suites of tools. The composition of the algorithms is obtained by appropriate wrappers between algorithm internal representations and the KDDML representation of data and models.

Objectives of this effort have been:

- combining the possibility to express a data mining application as a query with the advantages offered by the Knowledge Grid environment for executing the data mining query on a grid;
- extending the Knowledge Grid environment with a new class of knowledge discovery applications.

KDDML operators can be applied both to a data set, a leaf in the tree representing the overall query, and to the results produced by another operator. KDDML can support the parallel execution of the KDD application thanks to the possibility to split the query representing the overall KDD process into a set of sub-queries to be assigned to different processing nodes.

[6]http://www.ebi.ac.uk/swissprot/

The distributed execution of KDDML has been modeled according to the master-worker paradigm. In particular, the query entering component acts as the master, while each instance of the query executor is a worker. The main objectives of the execution of such queries on the grid are:

- distributing them to a number of grid nodes to exploit parallel execution of sub-queries;
- avoiding an excessive fragmentation of the overall task, i.e., allocating a number of nodes so that the exchanges of intermediate (partial) results gets minimized.

The grid execution of a KDDML application on a 4-node grid has shown that the benefits coming from the Knowledge Grid environment become profitable as soon as the data set size increases. For instance, the extraction of a classification model from a data set of 320,000 tuples, according to a two-step process (that is, first applying a clustering algorithm and next a classification one), resulted in an average speedup of about 3.7, reducing execution time from about 3 hours to about 45 minutes.

6 Conclusions

To support complex data-intensive applications, grid environments must provide adaptive data management and data analysis tools and techniques through the offer of resources, services, and decentralized data access mechanisms. In this report, we argued that according to the service-oriented architecture model, data mining tasks and knowledge discovery processes can be delivered as services in grid-based infrastructures. Through a service-based approach it is possible to design basic and complex services for supporting distributed business intelligence tasks in grids. Those services can address all the aspects that must be considered in data mining tasks and in knowledge discovery processes from data selection and transport, to data analysis, knowledge models representation and visualization. We discussed this approach and illustrated how it is implemented in the Knowledge Grid framework starting from basic services to data analysis services and KDD applications expressed as composition of lower level services.

References

[1] Piero Alcamo, Francesco Domenichini, and Franco Turini. An XML Based Environment in Support of the Overall KDD Process. In *Int. Conf. on Flexible Query Answering Systems (FQAS'00)*, pages 413–424, Warsaw, Poland, 2000. Physica-Verlag.

[2] Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Ivana Trickovic, and Sanjiva Weerawarana. BPEL for Web Services version 1.1, 2003. http://www-128.ibm.com/developerworks/library/specification/ws-bpel.

[3] Assaf Arkin, Sid Askary, Scott Fordin, Wolfgang Jekeli, Kohsuke Kawaguchi, David Orchard, Stefano Pogliani, Karsten Riemer, Susan Struble, Pal Takacsi-Nagy, Ivana Trickovic, and Sinisa Zimek. Web Service Choreography Interface (WSCI) 1.0, 2002. http://www.w3.org/TR/wsci.

[4] Giuseppe Bueti, Antonio Congiusta, and Domenico Talia. Developing Distributed Data Mining Applications in the Knowledge Grid Framework. In *Int. Conf. on High Performance Computing for Computational Science (VECPAR'04)*, volume 3402 of *LNCS*, pages 156–169, Valencia, Spain, 2004. Springer.

[5] Mario Cannataro, Carmela Comito, Antonio Congiusta, and Pierangelo Veltri. PROTEUS: a Bioinformatics Problem Solving Environment on Grids. *Parallel Processing Letters*, 14(2):217–237, June 2004.

[6] Mario Cannataro, Antonio Congiusta, Andrea Pugliese, Domenico Talia, and Paolo Trunfio. Distributed Data Mining on Grids: Services, Tools, and Applications. *IEEE Transactions on Systems, Man, and Cybernetics: Part B*, 34(6):2451–2465, December 2004.

[7] Mario Cannataro and Domenico Talia. The Knowledge Grid. *Communications of the ACM*, 46(1):89–93, January 2003.

[8] Mario Cannataro and Domenico Talia. Semantics and Knowledge Grids: Building the Next-Generation Grid. *IEEE Intelligent Systems and Their Applications*, 19(1):56–63, January 2004.

[9] Antonio Congiusta, Domenico Talia, and Paolo Trunfio. Distributed Data Mining Services Leveraging WSRF. *Future Generation Computer Systems*, 23(1):34–41, January 2007.

[10] Karl Czajkowski, Donald F. Ferguson, Ian Foster, Jeffrey Frey, Steve Graham, Igor Sedukhin, David Snelling, Steve Tuecke, and William Vambenepe. The WS-Resource Framework Version 1.0, 2004. http://www.ibm.com/developerworks/library/ws-resource/ws-wsrf.pdf.

[11] Anton J Enright, Stijn Van Dongen, and Christos A Ouzounis. An Efficient Algorithm for Large-scale Detection of Protein Families. *Nucleic Acids Research*, 30(7):1575–1584, 2002.

[12] Dietmar W. Erwin and David F. Snelling. UNICORE: A Grid Computing Environment. In *Int. Conf. on Parallel and Distributed Computing (Euro-Par'01)*, volume 2150 of *LNCS*, pages 825–834, Manchester, UK, 2001. Springer.

[13] Dietmar W. Erwin and David F. Snelling. Middleware for the Next Generation Grid Infrastructure. In *Int. Conf. on Computing in High Energy and Nuclear Physics (CHEP'04)*, Interlaken, Switzerland, 2004.

[14] Ian Foster. Globus Toolkit Version 4: Software for Service-Oriented Systems. In *Int. Conf. on Network and Parallel Computing (NPC'05)*, volume 3779 of *LNCS*, pages 2–13, Beijing, China, 2005. Springer.

[15] Ian Foster, Carl Kesselman, Jeffrey M. Nick, and Steven Tuecke. The Physiology of the Grid. In Fran Berman, Geoffrey Fox, and Anthony Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*. Wiley, 2003.

[16] Next Generation Grids Expert Group. Report 3: Future for European Grids: Grids and Service-Oriented Knowledge Utilities, January 2006.

[17] Hillol Kargupta and Philip Chan, editors. *Advances in Distributed and Parallel Knowledge Discovery*. AAAI Press, 2000.

[18] Hillol Kargupta, Byung Hoon Park, Daryl Hershbereger, and Erik Johnson. Collective Data Mining: A New Perspective Toward Distributed Data Mining. In Hillol Kargupta and Philip Chan, editors, *Advances in Distributed and Parallel Knowledge Discovery*. AAAI Press, 2000.

[19] Carlo Mastroianni, Domenico Talia, and Paolo Trunfio. Metadata for Managing Grid Resources in Data Mining Applications. *Journal of Grid Computing*, 2(1):85–102, March 2004.

[20] Andreas Prodromidis and Philip Chan. Meta-learning in Distributed Data Mining Systems: Issues and Approaches. In Hillol Kargupta and Philip Chan, editors, *Advances in Distributed and Parallel Knowledge Discovery*. AAAI press, 2000.

[21] Andrea Pugliese and Domenico Talia. Application-Oriented Scheduling in the Knowledge Grid: A Model and Architecture. In *Int. Conf. on Computational Science and its Applications (ICCSA'04)*, volume 3044 of *LNCS*, pages 55–65, Assisi, Italy, 2004. Springer.