Distributed Data Mining Services Leveraging WSRF

Antonio Congiusta [a] Domenico Talia [a,b] Paolo Trunfio [a]

[a] *DEIS, University of Calabria*
Via P. Bucci 41C, 87036 Rende, Italy

[b] *Exeura S.r.l.*
Edificio Polifunzionale, 87036 Rende, Italy

{acongiusta, talia, trunfio}@deis.unical.it

Abstract

The continuous increase of data volumes available from many sources raises new challenges for their effective understanding. Knowledge discovery in large data repositories involves processes and activities that are computational intensive, collaborative, and distributed in nature. The Grid is a profitable infrastructure that can be effectively exploited for handling distributed data mining and knowledge discovery. To achieve this goal, advanced software tools and services are needed to support the development of KDD applications. The Knowledge Grid is a high-level framework providing Grid-based knowledge discovery tools and services. Such services allow users to create and manage complex knowledge discovery applications that integrate data sources and data mining tools provided as distributed services on a Grid. All of these services are currently being re-designed and re-implemented as WSRF-compliant Grid Services. This paper highlights design aspects and implementation choices involved in such a process.

Key words: Distributed Data Mining, Grid Computing, OGSA, WSRF

1 Introduction

Today we are are much more able to store data than to extract knowledge from it. The Grid can be effectively exploited to analyze distributed data stores for finding interesting and useful knowledge hidden in them. The advent of the Grid has introduced substantial changes in the way data and computations are conceived and developed within industrial and scientific applications. Size limits, administrative boundaries, and data heterogeneity are no longer intractable problems nowadays. As a consequence, ever more huge amounts of data are being produced, stored, and moved within Grid systems as a result of data acquisitions from remote instruments, or scientific experiments, simulations, and so forth. Handling and mining large volumes of semi-structured and unstructured data is still the most critical issue currently affecting scientists and companies attempting to make an intelligent and profitable use of their data.

One of the recent challenges of the Grid is thus making the production and ownership of such data competitive and useful by allowing effective and efficient extraction of valuable knowledge from it. To this end, knowledge discovery and data mining services are needed to help researchers and professionals to analyze the very large amount of data that today is stored in digital formats in file systems, data warehouses, and databases distributed over corporate or worldwide Grids.

The Knowledge Grid [1–3] is a framework for implementing knowledge discovery tasks in a wide range of high-performance distributed applications. The Knowledge Grid offers to users high-level abstractions and a set of services by which is possible to integrate Grid resources to support all the phases of the knowledge discovery process, as well as basic, related tasks such as data management, data mining, and knowledge representation. Therefore, it allows end-users to concentrate on the knowledge discovery process they must develop, without worrying about the Grid infrastructure and its low-level details. The framework supports data mining on the Grid by providing mechanisms and higher level services for

- searching resources,
- representing, creating, and managing knowledge discovery processes, and
- composing existing data services and data mining services as structured, compound services,

to allow users to design, store, document, verify, share, and re-execute their applications, as well as manage their output results.

Previous research activities on the Knowledge Grid have been focused on the development of a system prototype by using early Grid middleware, as well as the design and evaluation of distributed KDD applications. Currently, the Knowledge Grid mechanisms are being designed and implemented following the *Service Oriented Architecture* (*SOA*) model. In particular, the so-called *Open Grid Services Architecture* (*OGSA*) paradigm and the emerging *Web Services Resource Framework* (*WSRF*) family of standards are being adopted for re-implementing the Knowledge Grid services. These services will permit the design and orchestration of distributed data mining applications running on large-scale, OGSA-based Grids.

This paper describes the development of the Knowledge Grid services by using OGSA and WSRF, discusses design aspects, execution mechanisms, and performance evaluations. The remainder of the paper is organized as follows. Section 2 presents a background about the Knowledge Grid architecture and its previous implementation. Section 3 discusses the SOA approach and its relationships with Grid computing. Section 4 presents a WSRF-based implementation of the Knowledge Grid services. Section 5 discusses preliminary considerations regarding system implementation and performance. Section 6 briefly compares system features and approach with related work. Section 7 concludes the paper.

2 Background

The Knowledge Grid architecture uses basic Grid mechanisms to build specific knowledge discovery services. These services can be implemented in different ways using available Grid environments such as the Globus Toolkit, UNICORE, and Legion. This layered approach benefits from "standard" Grid services that are more and more utilized and offers an open distributed knowledge discovery architecture that can be configured on top of Grid middleware in a simple way.

The Knowledge Grid services are organized in two hierarchical levels: the *Core K-Grid layer* and the *High-level K-Grid layer*. The High-level K-Grid layer includes services to compose, validate, and execute a distributed knowledge discovery computation. The main services of the High-level K-Grid layer are:

- The *Data Access Service* (*DAS*), responsible for the publication and search of data to be mined (data sources), as well as the search of inferred models (mining results).
- The *Tools and Algorithms Access Service* (*TAAS*), responsible for publishing and searching extraction tools, data mining tools, and visualization tools.
- The *Execution Plan Management Service* (*EPMS*). An execution plan is represented by a graph describing interactions and data flows between data sources, extraction tools, data mining tools, and visualization tools. The EPMS allows for defining the structure of an application by building the corresponding execution graph and adding a set of constraints about resources. The execution plan generated by this service is referred to as *abstract execution plan*, because it may include both well identified resources and *abstract resources*, i.e., resources that are defined through constraints about their features, but are not known a priori.
- The *Results Presentation Service* (*RPS*), offers facilities for presenting and visualizing the extracted knowledge models (e.g., association rules, clustering models, classifications).

The Core K-Grid layer offers basic services for the management of metadata describing features of hosts, data sources, data mining tools, and visualization tools. This layer also coordinates the application execution by attempting to fulfill the application requirements with respect to available Grid resources. The Core K-Grid layer comprises two main services:

- The *Knowledge Directory Service* (*KDS*), responsible for handling metadata describing Knowledge Grid resources. Such resources include hosts, data repositories, tools and algorithms used to extract, analyze, and manipulate data, distributed knowledge discovery execution plans, and knowledge models obtained as result of mining processes. The metadata information is represented by XML documents stored in a *Knowledge Metadata Repository* (*KMR*).
- The *Resource Allocation and Execution Management Service* (*RAEMS*), used to find a suitable mapping between an abstract execution plan and available resources, with the goal of satisfying the constraints (e.g., CPU, storage, memory, database, and network bandwidth requirements) imposed by the execution plan. The output of this process is an *instantiated execution plan*, which defines the resource requests for each data mining process. Generated execution plans are stored in the *Knowledge Execution Plan Repository* (*KEPR*). After the execution plan activation, this service manages the application execution and the storing of results in the *Knowledge Base Repository*(*KBR*).

2.1 Previous work

The main components of the Knowledge Grid environment have been implemented and are available through a software toolkit named *VEGA* (*Visual Environment for Grid Applications*), which embodies services and functionalities ranging from information and discovery services to visual design and execution facilities [4]. VEGA offers the users a simple way to design and execute complex Grid applications by exploiting the advantages coming from a Grid environment. In particular, it offers a set of visual facilities to design applications starting from a view of the present Grid status (i.e., available nodes and resources), and composing the different steps in a structured and comprehensive environment (see Figure 1). By using the abstractions offered by VEGA, a user does not need to directly perform the task of coupling the application structure with the underlying Grid infrastructure.

VEGA integrates functionalities of the EPMS and other Knowledge Grid services; in particular it provides for the following main features: *i*) task composition, *ii*) consistency checking, and *iii*) execution plan generation.

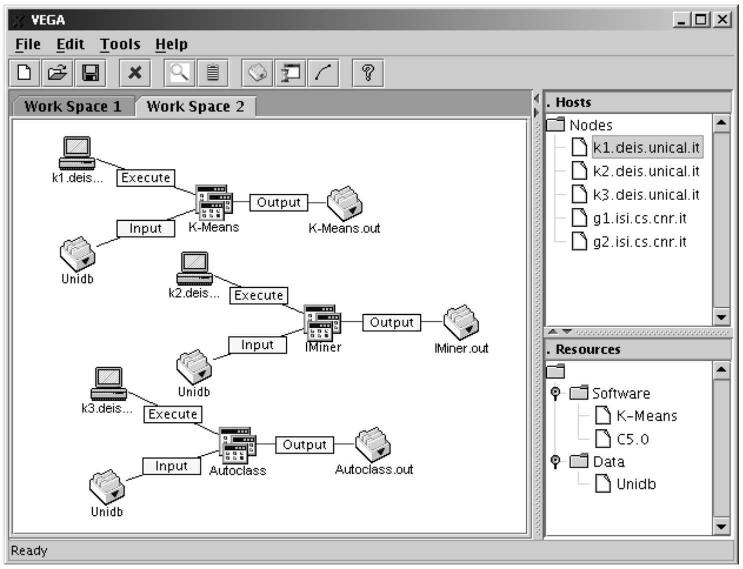

Fig. 1. The VEGA visual interface.

The *task composition* consists in the definition of the entities involved in the computation, and the specification of relationships holding among them. Key concepts in the VEGA approach to the design of a Grid application are the *visual language*, used to describe in a component-like manner the jobs constituting an application, and the possibility to group these jobs to form specific interdependent stages. In the *execution plan generation* phase the computation model is translated into an *execution plan* represented through an XML formalism.

VEGA has been used to implement several distributed data mining applications within the Knowledge Grid framework. Two application areas in which significant results have been achieved are intrusion detection [3] and bioinformatics [6]. The applications developed in such contexts have been useful for evaluating the overall system under different aspects, including its performance.

Main characteristics and problems that have been addressed include the use of massive datasets and the distribution of the computation load among different nodes. In the intrusion detection application a master-worker approach has been enforced, obtaining good results in terms of computation-time speed-up, as detailed in [3]. A similar approach has been applied to the clustering of human protein sequences, the main difference being the presence of different phases to be assigned to several nodes and the more complex execution coordination. Application details and performance results are presented in [6].

Another application developed using VEGA has been concerned with the integration of a query-based data mining system within a Grid environment [5]. The overall aim, in this case, has been enabling the execution of complex mining tasks expressed as high-level queries. To this purpose an existing system (able to combine KDD operators to classical database queries such as selection, join, etc.), has been transformed into a distributed Grid application by slightly modifying its structure and defining a proper allocation policy for the sub-queries generated by the system.

3 SOA and the Grid

The *Service Oriented Architecture* (*SOA*) is a programming model for building flexible, modular, and interoperable software applications. Concepts behind SOA are derived from component-based software, the object-oriented programming, and some other models. SOA enables the assembly of applications through parts regardless of their implementation details, deployment location, and initial objective of their development.

A *service* is a software building block capable of fulfilling a given task or business function. It does so by adhering to a well defined interface, defining required parameters and the nature of the result. Once defined and deployed, services operates independently of the state of any other service defined within the system, that is they are like “black boxes.” External components are not aware of how they perform their function, they care merely that they return the expected result. Nonetheless, services independence does not prohibit to have services cooperating each other to achieve a common goal. In fact, the final objective of SOA is to provide for an application architecture within which all functions are defined as independent services with well-defined interfaces, which can be called in defined sequences to form business processes [7].

The most important implementation of SOA is represented by *Web Services*. The popularity of Web Services is mainly due to the adoption of universally accepted technologies such as XML, SOAP, and HTTP. The Web is not the only area that has been attracted by the SOA paradigm. Also the Grid can provide 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 trend of the latest years proved that not only the Grid is a fruitful environment for developing SOA-based applications, but also that the challenges and requirement posed by the Grid environment can contribute to further developments and improvements of the SOA model.

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 [8]. Recently the *WS-Resource Framework* (*WSRF*) has been adopted as an evolution of early OGSA implementations [9]. 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. Besides, the *WS-Notification* specification defines a *publish-subscribe* notification model for Web Services, which is exploited to notify interested clients and/or services about changes that occur to the status of a WS-Resource. The combination of stateful resources and the notification pattern can be exploited to build distributed, long-lived Grid applications in which the status of the computation is managed across multiple nodes, and services cooperate in a highly-decentralized way.

4 Knowledge Grid WSRF services

This section describes the design and implementation of the Knowledge Grid in terms of the OGSA and WSRF models.

Figure 2 proposes a view of the Knowledge Grid architecture in which each Knowledge Grid service (*K-Grid service*) is exposed as a Web Service that exports one or more operations, by using the WSRF conventions and mechanisms.

The operations exported by High-level K-Grid services (DAS, TAAS, EPMS, and RPS) are designed to be invoked by user-level applications, whereas operations provided by Core K-Grid services (KDS and RAEMS) are thought to be invoked both by High-level and Core K-Grid services.

As shown in Figure 2, users can access the Knowledge Grid functionalities by using a *client interface* located on their machine. The client interface can be an integrated visual environment that allows for performing basic tasks (e.g., searching of data and software, data transfers, simple job executions), as well as defining distributed data mining applications described by arbitrarily complex execution plans (see Section 4.3). The client interface performs its tasks by invoking the appropriate operations provided by the different High-level K-Grid services. Those services may be generally executed on a different Grid node; therefore the interactions between the client interface and High-

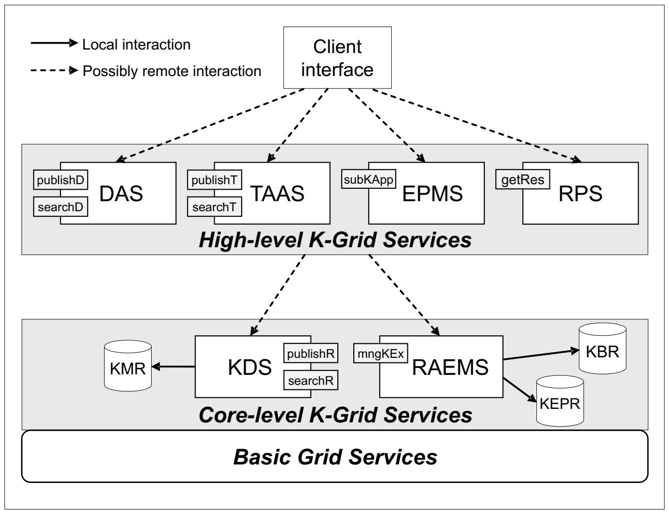

Fig. 2. Interactions between a client and the Knowledge Grid environment.

level K-Grid services are possibly remote.

All K-Grid services export three mandatory operations - `createResource`, `subscribe` and `destroy` - and one or more service-specific operations. The `createResource` operation is used to create a stateful resource, which is then used to maintain the state (e.g., results) of the computations performed by the service-specific operations. The `subscribe` operation is used to subscribe for notifications about computation results. The `destroy` operation removes a resource.

The implementation of a K-Grid service follows the *WS-Resource factory pattern* (see Figure 3). In this pattern, a *factory service* is in charge of creating the resources and an *instance service* is used to operate on them. Thus the `createResource` mandatory operation introduced above is provided by the factory service, while the other operations are exported by the instance service. To create a resource the client contacts the factory service, which creates a new resource and assigns to it a unique key. The factory service will return an *endpoint reference* that includes the *resource id* and is used to directly access the resource through the instance service.

Table 1 lists and describes the main operations associated to K-Grid services. In the following subsections a more detailed presentation of the main services and operations features is provided. Moreover, considerations about the client interface are reported.

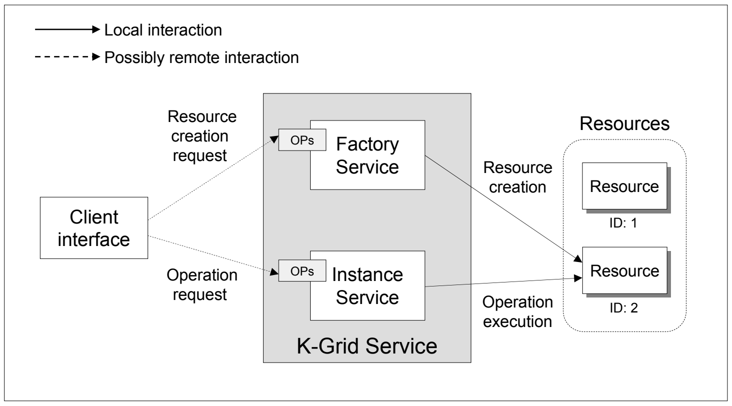

Fig. 3. K-Grid service design

Table 1
Description of main K-Grid service operations.

Service	Operation	Description
DAS	`publishData`	This operation is invoked by a client for publishing a newly available dataset. The publishing requires a set of information that will be stored as metadata in the local KMR.
	`searchData`	Data to be used in a KDD computation is located during the application design by invoking this operation. The searching is performed on the basis of appropriate parameters.
TAAS	`publishTools`	This operation is used to publish metadata about a data mining tool in the local KMR. As a result of the publishing, a new DM service is made available for utilization in KDD computations.
	`searchTools`	It is similar to the searchData operation except that it is targeted to data mining tools.
EPMS	`submitKApplication`	This operation receives a conceptual model of the application to be executed. The EPMS generates a corresponding abstract execution plan and submits it to the RAEMS for its execution.
RPS	`getResults`	Retrieves results of a performed KDD computation and presents them to the user.
KDS	`publishResource`	This is the basic, core-level operation for publishing data or tools. It is thus invoked by the DAS or TAAS services for performing their own specific operations.
	`searchResource`	The core-level operation for searching data or tools.
RAEMS	`manageKExecution`	This operation receives an abstract execution plan of the application. The RAEMS generates an instantiated execution plan and manages its execution.

4.1 Execution management

Figure 4 describes the interactions that occur when an invocation of the EPMS service is performed. In particular, the figure outlines the sequence of invocations to others services, and the interchanges with them when a KDD application is submitted for allocation and execution. To this purpose, the EPMS exposes the `submitKApplication` operation, through which it receives a conceptual model of the application to be executed (step 1). The conceptual model is a high-level description of the KDD application more targeted to distributed knowledge discovery aspects rather than to Grid-related issues.

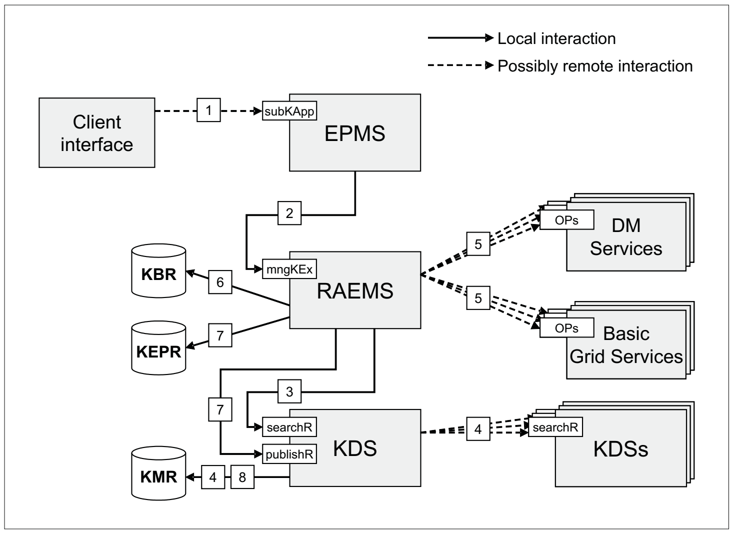

Fig. 4. EPMS interactions.

The basic role of the EPMS is to transform the conceptual model into an abstract execution plan for subsequent processing by the RAEMS. It is worth recalling here that 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 to be used, but rather constraints and other criteria about them.

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). Note that the KDS performs the searching both accessing the local KMR and querying remote KDSs (step 4).

After the instantiated 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 instantiated execution plan (step 5). The results of the computation are stored by the RAEMS into the KBR (step 6), while the execution plan is stored into the 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).

DAS and TAAS services are concerned with the publishing and searching of datasets and tools to be used in a KDD application. They possess the same basic structure and perform their main tasks by interacting with a local instance of the KDS that in turn may invoke one or more other remote KDS instances.

Figure 5 describes the interactions that occur when the DAS service is invoked; similar interactions apply also to TAAS invocations. The `publishData` operation is invoked to publish information about a dataset (step 1). The DAS passes the corresponding metadata to the local KDS, by invoking the `publishResource` operation (step 2). The KDS, in turn, stores that metadata into the local KMR (step 3).

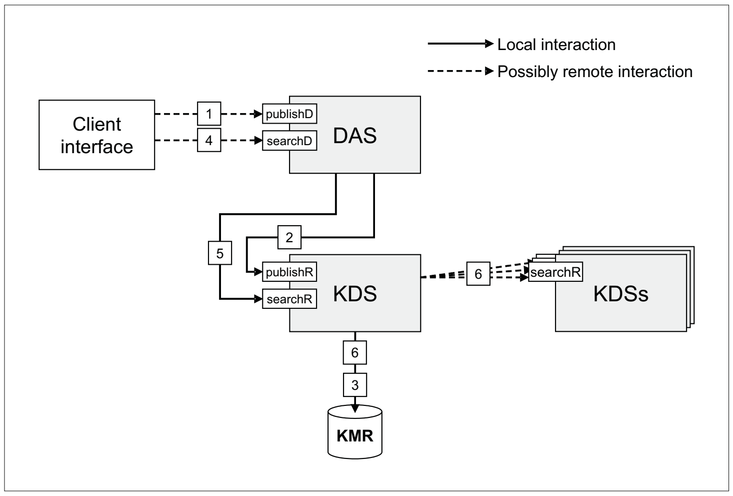

Fig. 5. DAS interactions.

The `searchData` operation is invoked by a client interface that needs to locate a dataset on the basis of a given set of criteria (step 4). The DAS submits its request to the local KDS, by invoking the corresponding `searchResource` operation (step 5). As mentioned before, the KDS performs the searching both accessing the local KMR, and querying remote KDSs (step 6). This is a general rule enforced in all the interactions between a high-level service and the KDS when a searching is requested. The local KDS is thus responsible for dispatching the query to remote KDSs and for generating the final answer.

The search for a dataset is performed through the `searchData` operation starting from a search string passed by the client. It contains the searching criteria expressed as attribute-value pairs regarding key properties through which datasets are categorized within the system by using metadata. The outcome of the searching is a set of URLs (stored as an array of strings) pointing to the metadata of the datasets corresponding to that searching criteria. These kinds of URLs are specifically targeted at the KDS service: it implements, in fact, a custom protocol for locating metadata descriptions of Grid resources.

4.3 Client interface

As mentioned before, the *client interface* is a component through which a user can publish and search resources and services, design and submit KDD applications, and visualize results.

An application can be a single data mining task (e.g., data clustering or classification), as well as a distributed data mining application described by a given formalism, such as the visual language used in VEGA (see Section 2.1). In other words, the client interface provides the user with an environment to design data mining applications and submit requests to the EPMS service. During the execution, the client interface will receive notifications from the EPMS about the execution progress, including failure notices.

Our experience demonstrated that providing a set of abstractions, a programming model, and high-level facilities results in an improved and simplified way of building KDD applications able to exploit the Grid environment. An archetype of a client interface for the Knowledge Grid interacts with the user as to guide her/him in the process of building distributed KDD applications using high-level services and features provided by the Knowledge Grid environment (e.g., access services), to better achieve the design goals. The process of designing a data mining application through a client interface adhering to this approach should thus comprise the search for resources and services to be included in the computation, and the specification in a given high-level formalism of the structure of the application. This must reflect how the selected services interact each other and must be coordinated. In addition, the client interface should allow the definition of specific requirements on the execution and results visualization.

5 Discussion and evaluation

Designing the WSRF-based version of the Knowledge Grid benefitted of the service-oriented approach used in the original design of the system [1]. That design approach conceived the Knowledge Grid architecture and functionality as a set of basic and high-level services that did not pose any constraints on the implementation strategy. This choice facilitated re-designing the system and implementing the new WSRF-version by maintaining the same architecture and exposing the same functionalities as Web Services. In terms of service composition and integration, the WSRF standards simplify and homogenize the service-to-service interaction. In the pre-WS version of the Knowledge Grid each service adopted a specific protocol. This resulted in a more complex implementation and in a limited interoperability with external services and components. On the contrary, the WSRF standard allows for a better integration of the Knowledge Grid services and their interaction with third-party Web and Grid services.

Given the design and implementation benefits discussed above, another key aspect in evaluating the appropriateness of using WSRF is related to its performance in supporting data mining service execution. To evaluate the performance of the WSRF mechanisms to implement distributed data mining services as discussed throughout the previous sections, we performed some experiments to measure execution times of the different steps for invoking a WSRF-compliant K-Grid service. The K-Grid service used in our tests exports two service-specific operations `clustering` and `classification`, as well as the mandatory operations `createResource`, `subscribe` and `destroy`.

The `clustering` operation implements the *Expectation Maximization* (*EM*) clustering algorithm, while the `classification` operation implements the *J48* classification algorithm. The K-Grid service and the client program have been developed by using the WSRF Java library provided by Globus Toolkit 4 [10]. Client and service have been deployed on Grid nodes connected through a wide area network, with an average bandwidth of 213 Kbps and an average RTT of 19 ms.

In the clustering experiments we used the *census* dataset available at the UCI repository [11]. Through random sampling we extracted from it ten datasets, containing a number of instances ranging from 1700 to 17000, with a size ranging from 0.5 to 5 MB.

Table 2 reports the times needed to complete the different `clustering` operation phases, defined as follows. *Resource creation*: the client invokes the `createResource` operation and receives a reference to the created resource. *Notification subscription*: the client invokes the `subscribe` operation. *Task submission*: the client submits the execution of the clustering task by invoking the `clustering` operation. *File transfer*: the dataset to be mined is transferred to the node hosting the service. *Data mining*: the clustering analysis is performed by the service. *Results notification*: the clustering model is delivered to the client through a notification message. *Resource destruction*: the client invokes the `destroy` operation for destroying the resource.

Values reported in the table refer to the execution times obtained for different dataset sizes. The table shows that the *data mining* phase takes from 84.6% to 88.3% of the total execution time, while the *file transfer* phase fluctuate

Table 2
Execution times (in ms) of the invocation phases of a single K-Grid Service.

Dataset size	**Resource creation**	**Notification subscription**	**Task submission**	**File transfer**	**Data mining**	**Results notification**	**Resource destruction**
0.5 MB	1960	345	293	14638	110415	2578	258
2.0 MB	1999	300	333	54582	425238	2991	204
3.5 MB	1929	321	272	93596	749369	2823	239
5.0 MB	1904	287	242	132463	1044533	3057	237

around 11.2%. The overhead due to the specific WSRF mechanisms - *resource creation*, *notification subscription*, *task submission*, *results notification*, and *resource destruction* - is very low with respect to the overall execution time, decreasing from 4.2% to 0.5% with the growing of the dataset size.

In the classification experiments the *ad* dataset from the UCI repository has been employed. The sizes of the extracted datasets were ranging from 5 to 10 MB. The phases involved in the `classification` operation are the same described in the clustering experiments (and listed in Table 2). As expected, the execution times of the WSRF mechanisms resulted substantially equal to those measured in the clustering experiments, while the *file transfer* and *data mining* execution times changed because of the different dataset sizes and algorithm complexity. In particular, the *file transfer* execution time ranged from 132655 ms for the dataset of 5 MB to 256021 ms for the dataset of 10 MB, while the *data mining* execution time ranged from 257116 ms for 5 MB to 1271721 ms for 10 MB. In terms of percentages, the *file transfer* took from 33.6% to 16.7%, while the *data mining* phase required from 65.2% to 83.0% of the total execution time. The overall WSRF overhead ranged from 1.2% to 0.3% of the total execution time.

In general, it can be observed that the overhead introduced by the WSRF mechanisms is not critical with respect to the duration of the service-specific operations. This is particularly true in typical knowledge discovery applications, in which the data mining algorithms working on large datasets are expected to take a long processing time. On the basis of our preliminary experimental results, we conclude that WSRF mechanisms can be effectively exploited to develop high-level services for distributed KDD applications on Grids.

6 Related work

Since computational Grids proved effective as platforms for data-intensive computing, some Grid-based data mining systems have been proposed (see [3] for a quick survey). Among those, two systems that exploit a service-oriented approach for providing Grid-based KDD services are *Discovery Net* [12] and

Grid Miner [13].

Discovery Net 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 an OGSA-compliant Grid service. However, Discovery Net currently uses an early implementation of OGSA - namely, the *Open Grid Services Infrastructure* (*OGSI*) - which has been replaced by WSRF for lack of compatibility with standard Web Services technologies.

GridMiner aims at covering the main aspects of knowledge discovery on Grids. Key components in GridMiner are *Mediation Service*, *Information Service*, *Resource Broker*, and *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 services themselves do not communicate with each other. No service is aware of any other existing service. Hence each of them is able to run completely independently. To support the individual steps of KDD processes, the output of each service can be used as input for the subsequent service. Like Discovery Net, also Grid Miner has been implemented on OGSI.

As a matter of fact, the Discovery Net approach is similar in many aspects to the approach we followed in the Knowledge Grid in providing a service-based middleware for data mining on Grids. On the contrary, the Grid Miner system provides single services implementing the main steps of a KDD process and a service composition engine to execute a multi-step data mining application.

To the best of our knowledge, none of the existing systems makes use of WSRF as basic technology. Therefore, the Knowledge Grid is the first system leveraging WSRF for building a comprehensive high-level framework for distributed knowledge discovery in Grid environments, supporting also the integration of data mining algorithms exposed through a Web Service interface.

7 Conclusions

Data-mining Grid services are key elements for practitioners who need to develop knowledge discovery applications that use large and remotely dispersed datasets and/or high-performance computers to get results in reasonable times and improve their competitiveness. In this paper we addressed the definition and composition of Grid services for implementing distributed knowledge discovery processed and applications on WSRF-based Grids. We presented Grid services for searching Grid resources, composing software and data elements, and managing the execution of data mining applications on Grids.

The paper discussed the definition of data mining Grid services in the context of the Knowledge Grid architecture. The services and operations presented in this paper allow for data and tools publishing and searching, execution submission and resource management, and retrieving of the produced results. We observed that the original design of the Knowledge Grid system as a service oriented architecture simplified the porting process towards the OGSA-compliant implementation.

The availability of basic services for knowledge discovery in Grid and Web environments will allow users to implement:

- single data mining tasks that run on remote machines
- distributed data mining models such as collective mining, meta-learning and peer-to-peer mining, and
- complex long-lived distributed knowledge discovery applications involving a large number of world-wide Grid nodes providing datasets and/or mining algorithms.

Acknowledgements

This research work is carried out under the FP6 Network of Excellence Core-GRID funded by the European Commission (Contract IST-2002-004265). This work has been also supported by the Italian MIUR FIRB Grid.it project RBNE01KNFP on High Performance Grid Platforms and Tools.

References

[1] M. Cannataro, D. Talia and P. Trunfio. Distributed data mining on the grid. *Future Generation Computer Systems*, **18**(8):1101-1112, 2002.

[2] M. Cannataro and D. Talia. The Knowledge Grid. *Communitations of the ACM.* **46**(1):89-93, 2003.

[3] M. Cannataro, A. Congiusta, A. Pugliese, D. Talia and P. Trunfio. Distributed Data Mining on Grids: Services, Tools, and Applications. *IEEE Transactions on Systems, Man, and Cybernetics, Part B.* **34**(6):2451-2465, 2004.

[4] M. Cannataro, A. Congiusta, D. Talia and P. Trunfio. A Data Mining Toolset for Distributed High-Performance Platforms. Int. Conference Data Mining 2002, WIT Press, pp. 41-50, 2002.

[5] G. Bueti, A. Congiusta and D. Talia. Developing Distributed Data Mining Applications in the KNOWLEDGE GRID Framework. Int. Meeting on High Performance Computing for Computational Science (VECPAR'04), LNCS 3402, pp. 156-169, 2005.

[6] M. Cannataro, C. Comito, A. Congiusta and P. Veltri. PROTEUS: a Bioinformatics Problem Solving Environment on Grids. *Parallel Processing Letters.* **14**(2):217-237, 2004.

[7] K. Channabasavaiah, K. Holley and E. M. Tuggle. Migrating to a service-oriented architecture. 2003. http://www-106.ibm.com/developerworks/library/ws-migratesoa.

[8] I. Foster, C. Kesselman, J. Nick and S. Tuecke. The Physiology of the Grid. In: F. Berman, G. Fox and A. Hey (eds.), Grid Computing: Making the Global Infrastructure a Reality, Wiley, pp. 217-249, 2003.

[9] K. Czajkowski et al. The WS-Resource Framework Version 1.0. 2004. http://www-106.ibm.com/developerworks/library/ws-resource/ws-wsrf.pdf.

[10] I. Foster. Globus Toolkit Version 4: Software for Service-Oriented Systems. Proc. Conference on Network and Parallel Computing (NPC 2005), LNCS 3779, pp. 2-13, 2005.

[11] The UCI Machine Learning Repository. http://www.ics.uci.edu/∼mlearn/MLRepository.html.

[12] S. Al Sairafi et al. The Design of Discovery Net: Towards Open Grid Services for Knowledge Discovery. *Int. Journal of High Performance Computing Applications.* **17**(3):297-315, 2003.

[13] P. Brezany, J. Hofer, A. M. Tjoa and A. Woehrer. GridMiner: An Infrastructure for Data Mining on Computational Grids. APAC Conference and Exhibition on Advanced Computing, Grid Applications and eResearch (APAC'03), 2003.