KDDBroker: Description and Discovery of KDD Services

Jessica Cellini, Claudia Diamantini, Domenico Potena

Dipartimento di Ingegneria Informatica, Gestionale e dell’Automazione, Università Politecnica delle Marche - via Breccia Bianche, 60131 Ancona, Italy
{cellini,diamantini,potena}@diiga.univpm.it

Abstract. Service Oriented Architectures (SOA) can be profitably used in the domain of distributed Knowledge Discovery in Databases (KDD), since they provide a way to effectively share and re-use information, tools, models and domain expertise, and to compose different tools to design a KDD process. A key SOA component is the service broker, which provides service publishing and discovering facilities, by exploiting information stored in the UDDI registry. In this paper we present the results of an ongoing project for the design of a KDD service broker, which extends the existing standards in order to introduce semantic information on the KDD domain. In particular, the paper focuses on the semantic description and discovery of Data Mining tools. To this end, we introduce an ontology that, at this stage, represents a categorization of Data Mining domain in terms of tasks, methods and algorithms, and gives linkability relations among them. The ontology is then exploited for tool annotation. We discuss a possible architecture for the inclusion of the ontology in the UDDI, and present the implementation of the service broker functionalities.

Keywords: Knowledge Discovery in Databases, Service Broker, UDDI Extension, Ontology.

1 Introduction

The scope of a Knowledge Discovery in Databases (KDD) process is to mine database in order to identify valid, novel, potentially useful, and ultimately understandable knowledge nuggets [12]. To this end, it implies the the accomplishment of different tasks, like domain understand, data preprocessing, data mining, and knowledge interpretation/evaluation. Each of these can be realized by different and various tools. Such tools come from different research areas, like databases, machine learning, artificial intelligence, statistics and so on. It implies that an user should have various skills and expertise in order to manage all of them. As a matter of fact, in a KDD process the user has to choose, to set-up, to compose and to execute the tools more suitable to her/his problem. In a distributed and collaborative scenario, where tools are put at disposal by different actors distributed over the Net, the design of a KDD process becomes more complex. As a matter of fact, in this scenario, there is a lack of knowledge about both the functionalities of tools and their technical details. Even if the tool is provided
with a description, it is not given neither in a standard nor machine-readable form. Moreover, due to the continuous research activities in KDD and Data Mining fields, the amount of tools that an user can use for discovering knowledge is constantly growing. In [10, 11] we extensively discussed and motivated the use of a Service Oriented Architecture (SOA) in order to build an open, collaborative and distributed support environment, where resources can be easily annotated, introduced, accessed, described, composed and activated. There, the general architecture of a platform called Knowledge Discovery in Databases Virtual Mart (KDDVM) is also provided [16]. A core element of our platform as well as any other SOA [9, 23] is the service broker, that is a registry and a set of APIs to interact with it. According to W3C standards [28], we implement the registry by the Universal Description, Discovery and Integration (UDDI). At present, a service broker is able to provide a narrow set of discovery functionalities: the search for a service is only based on its name or, at the most, on its provider. These functionalities limit the support that can be offered to an user in the KDD domain.

To give some example, let us consider a KDD consulting company, dealing with a classification problem. Consultants, on the basis of their experience and analyzing the problem, decide that a non-linear Support Vector Machine can be profitably exploited to achieve their goal. In this case, a service broker should be able to search for service implementing the specific algorithm, and the broker besides listing services implementing “non-linear Support Vector Machine”, should replying with services where the algorithm’s name is “SVM”, “SVM-light”, “RBF Support Vector Machine”, and so on. At this point, the service is activated and a model is built, but consultants decide to improve the result trying a feature extraction tool to reduce the problem dimensionality. So, a useful discovery functionalities could be search for feature extraction algorithm that can be used in combination with SVM, like SVMDBA [29]. Besides these examples, others specific KDD discovery functionalities should be provided, like search services accomplishing a given task or method, search services on the basis of their input/output, as well as a search based on authors/providers reliability. Thus, in this paper we present the results of an ongoing project for the design of a KDD service broker (KDDBroker), which extends the existing standards in order to introduce semantic information on the KDD domain. To this end, we introduce an ontology that, at this stage, represents a categorization of KDD domain in terms of tasks, methods and algorithms, and gives linkability relations among them.

In the next section, we present the data schema and APIs of the UDDI standard. Section 3 gives an overview of related works, discussing the solutions proposed for extending the UDDI registry and for supply a broker with semantic functionalities. In Section 4, we show the information needed to describe a generic KDD tool and to introduce a language to represent it in the KDDVM. Section 5 proposes KDDONTO, an ontology for KDD domain. In Section 5, we first present the service broker for the KDDVM platform, then an implementation is proposed. Finally some conclusions are given.
2 UDDI Broker

The Universal Description, Discovery and Integration (UDDI) [25] specification is an initiative to create a global, platform-independent, open framework for publishing and discovering Web Services. The UDDI standard allows businesses to describe their business services, how services can be accessed, what information they want to make public, and what information they choose to keep private. On the other hand, UDDI supplies customers with a set of standard functionalities for service discovery. In order to support the discovery process, the UDDI provides three types of information about Web Services:

- white pages: name and contact details;
- yellow pages: categorization based on business and service types;
- green pages: technical data about services;

The UDDI specifications take advantage of World Wide Web Consortium (W3C) [28] and Internet Engineering Task Force (IETF) [14] standards such as Extensible Markup Language (XML), HTTP, Domain Name System (DNS) protocols, Web Service Description Language (WSDL) and Simple Object Access Protocol (SOAP) messaging specifications. UDDI data model is defined by four basic entities: `businessEntity`, `businessService`, `tModel`, and `bindingTemplate`. Each instance of the basic entities is uniquely identified by the Universally Unique Identifier (UUID). These entities are briefly described in the following.

`businessEntity`:

The `businessEntity` is the principal entity of UDDI. It contains the core details of the business: its name and description, details of personal contacts within the business, identifiers used to refer to the business, categorical classifications of the business. Name and contact details define the white pages information of the UDDI registry. Categorical classifications of the business is part of the yellow pages. A `businessEntity` contains a list of services provided by the business, whose description is contained in the `businessService` entity.

`businessService`:

Each `businessService` instance describes a single service provided by a business. A service is characterized by a name, a description, and a classification of the service that completes the yellow pages registry. It contains also a link to one or more `bindingTemplate` providing technical information.

`bindingTemplate`:

`bindingTemplate` mainly represent the technical information about the access point (URL) of the service. `bindingTemplate` information is typically retrieved by an organization interested in using the related `businessService`, and exploited for dynamic invocation at runtime. `bindingTemplate` allows a publisher to suddenly change the entry point without notifying all its business partners of this change. Each `bindingTemplate` references to one or more instances of `tModel`, describing the technical information about how to access the service.

`tModel`:

A `tModel`, or technical model, represents two concepts within UDDI: a technical
specification for a given service type, or a model for a particular identifier or taxonomy. For examples tModels can refer to WSDL files, XML DTDs, or classification schemes. A unique tModel can be addressed by different services. A tModel gives information about its name, the name of the organization that publishes it, a list of categories that identify the service type, and pointers to technical specifications (e.g. interface definitions, message formats, and message protocols). BindingTemplate and tModel together define the green pages.

Classification of services, provided by the yellow pages, are maintained in a specific structure called categoryBag. The UDDI public registries provide validated support for a number of taxonomies that cover industry codes (NAICS), product and service classifications (UNSPSC), and geographical classification (ISO 3166). New domain classifications can be specified and included in the categoryBag. Inclusion of classifications is optional but greatly enhances search functionalities.

The UDDI specifications also include definitions for web services interfaces in order to allow programmatic access to the registry information (APIs). These are divided into two logical parts that are the inquiry and the publishing APIs.

At present, an UDDI broker supplies user with only syntactic discovery functionalities, limiting the support that can be offered to an user in the KDD domain. As a matter of fact, user may search services only by its name (white pages) or, at the most, by a plain taxonomy (yellow pages). Furthermore, although UDDI entities are defined by XML, they lack explicit semantics. Thus, it is needed extending UDDI standard in order to introduce new discovery functionalities based on services capabilities.

3 Related works

In the literature, it is often discussed the need of both representing and searching for other kinds of information not provided in the standard. UDDI specification [25] defines two ways to add new information into registry. The former is to define a tModel in order to address the WSDL description of the service. As a matter of fact, a WSDL document defines how to invoke a service. It provides information on the data being exchanged, the sequence of messages for an operation, the location of the service and the description of bindings (e.g. SOAP or HTTP). The other way is to use a categoryBag to classify services based on their functionalities.

In [24] researchers discuss the introduction of new properties like service leasing and quality of service. To this end they develop another type of information, called blue pages. Properties are added into blue page registry by defining <name,value> pairs, like categoryBag. This approach is similar to one of the approaches discussed in [19], which proposes a domain oriented UDDI registry architecture and addresses new concepts such as service property schema, service relationship and service constraint. The authors introduce three possible solutions to UDDI extension, discussing advantages and disadvantages for each of
them. The first solution provides a method to publish new information without modifying UDDI data model. Such solution uses tModel to refer to service properties. The main problem related to this solution is that any provider should use the same property definition schema and should extend the discovery interface (i.e. the APIs). Moreover, it guarantees a low discovery efficiency, because the external files are distributed. Second solution considers extending UDDI data model in order to store non-functional information like Quality of Service. Such UDDI extension allows for browsing and discovering only Quality of Service properties. The last solution is to use an external database. This solution does not modify UDDI data model. In [19] authors also prove that this approach allows to store additional information related to service relationship and constraints with a high discovery efficiency. Our scenario is quite different and more complex compared with previous works. As a matter of fact, information we have to manage is structured and related each other. So, these approaches are inadequate to our aims.

An important research direction is to discover web services on the basis of the capabilities they provide. Note that this implies the use of some form of semantic information, that can be represented by means of ontologies. In [20], authors propose a semantic broker for InfoSleuth, an agent-based platform for information discovery and retrieval in a dynamic, open environment. This broker searches for agents that provide services to agents requesting services. To this end, the broker implements reasoning capabilities that exploit a common and shared ontology concerning both agents description and functionalities they provide. A more general approach to discovery information is that described in [8], where the authors analyze a model for the semantic interoperability of an open network system. This paper focuses on the information resource discovery problem, where autonomous businesses require a coordinated access to heterogeneous and distributed information resources. The search function is based on a semantic affinity evaluations on ontology, representing the semantics of the information resources required and in use by a given business. In [4] and [21], the problem of compiling semantic information into UDDI is discussed. In [4], is proposed an approach to ontology-based service discovery, in order to enhance the UDDI broker functionalities. In this work a service is selected through semantic-based matching algorithms according to a scored mechanism taking into account semantic relationships among services. [21] shows how services capabilities described in DAML-S can be mapped into UDDI registry, and how they can be used to perform a semantic service discovery. However, these works propose general methodologies for the introduction of semantic information into UDDI, while our scope is design an UDDI broker for the management of specific KDD information.

4 KDD service

The Knowledge Discovery in Databases process includes several steps, that implies the interaction of various tools derived from different domains. The het-
ergonomy of tools as well as the goal-driven and domain dependent nature of any KDD problem introduce complexity in the design of a KDD process. Such a complexity is mainly due both to the huge amount of tools the user can choose and to the expertise needed facing various KDD tasks. So, it is also necessary to define an environment supporting users in specific KDD activities, like:

- building a repository of KDD tools;
- easily introducing in the repository new algorithms or releases;
- browsing the tool repository and obtaining information about tools;
- managing and understanding the input dataset;
- choosing the more suitable tools on the basis of a number of characteristics: tools performances (complexity, scalability, accuracy), the kind of data tools can be used for (textual/symbolic data, numeric, structured data, sequences, ...), the kind of goal tools are written for (data cleaning, data transformation, data mining, visualization, ...) the kind of data mining task (classification, rule induction, ...);
- setting the tool parameters and activating it;
- designing the KDD process by tools composition;
- managing the KDD whole process and its versions;
- representing the models in a understandable form.

Open support environments proposed in the recent literature are based on open architecture like Grid [6, 7, 27, 5] and SOA [18, 17, 13, 1]. Such proposals are mainly focused on supporting the data mining phase. In order to support users in the design of a whole KDD project, we are working on the development of an open environment based on the service-oriented paradigm, called Knowledge Discovery in Databases Virtual Mart [10] (see Fig. 1).

Such system is formed by three kinds of services: basic, support and advanced services. The former includes tools implementing data mining algorithms as well as any other data manipulation algorithm. Advanced services provide an "intelligent" support to user. In particular, they are mainly devoted to proactively help user in the choice of tools most suitable with her/his goals. Finally, the core layer of KDDVM contains support services, that mainly give support to browsing, composition and activation of tools. We codified the information needed
to accomplish these tasks in the Knowledge Discovery Tool Markup Language (KDTML)[22]. KDTML tool descriptions are XML documents where XML tags describe the characteristics of tools and their interfaces. Due to lack of space, we refer to the KDDVM project site [16] for both the general structure of a KDTML document and some examples. The language is divided into four main sections: The initial KDTML fragment contains information in order to locate and to execute the tool, the second section describes the tool I/O interface, the third part lists the KDD software modules that can be linked up with the described tool, and the final section is a categorization of the KDD tool with respect to a KDD taxonomy. In particular, the last two parts play a leading role in the KDDVM. As a matter of fact, in order to give support to users with different degree of expertise and knowledge, it is necessary to give a description of functionalities of any KDD tools. Such information are represented according to both a common vocabulary and a predefined taxonomy. Such a structure is an extension of the Data Mining taxonomy DAMON [6], which covers the other KDD tasks, like data cleaning, data transformation, data selection, visualization and so on. A taxonomical structure is a natural way to characterize the KDD tools domain in terms of the implemented task. The goal of each KDD task can be achieved by various and different methods, e.g. for the classification task an user can use a Decision Tree, a Neural Network, a Fuzzy Set Approach as well as a Genetic Algorithm. For each method are available a lot of algorithms, e.g. C4.5, ID3 and CART are Decision Tree algorithms. Finally any KDD tool is a specific implementation of a given algorithm. Furthermore, in order to achieve a KDD goal, the exploitation of a single tool is often not sufficient, rather a KDD process is typically composed of many different tools, such that the output of a tool represents the input to another one. Then, in order to support the user in such a tool composition activity, the description of a tool should be enriched with information about KDD tools that can be linked up with the described one, we named such kind of information linkable modules.

5 KDDONTO: a KDD ontology

This section is devoted to present the design of KDDONTO an ontology for the KDD domain. In the following, we first present an abstract specification of the ontology, then an OWL implementation is proposed.

Analyzing the information needs to describe a KDD tool, it turns out that the main concepts to represent are: algorithm, method and task. More formally, for the scope of our ontology:

- algorithm is a procedure through which a KDD tool is accomplished;
- method is a KDD methodology or technique used to achieve a specific task;
- task is a goal of a KDD problem.

These concepts are represented by disjoint classes.

Categorical relationships among these concepts exist. In particular, each algorithm implements a specific method, while a method can be performed by
different algorithms. As regards task and method, a task can be achieved by multiple methods, and vice versa. Moreover, each algorithm can be linked to other algorithms, according to the linkable modules section of KDTML. Finally, in the KDD literature a same concept could be indicated with different names or acronyms (e.g. BVQ and MAMR LVQ represent the same algorithm, SVM is the acronym of Support Vector Machine). Thus, an alias relationship between different concepts is also needed. So, in KDDONTO the following properties are defined:

- uses between an instance of algorithm and the implemented instance of method;
- specifiesTask between an instance of method and its instance of task;
- in_module between instances of algorithm, the range is formed by the algorithms that are often used before the described one;
- out_module between instances of algorithm, it is the inverse of in_module;
- sameAs between instances belonging to any concept of the ontology, its range is the set of instances having the same mean.

Figure 2 shows an overview of concepts and properties of the KDDONTO. Figure 3 shows the piece of KDDONTO related to Bayes Vector Quantizer (BVQ) algorithm, a Data Mining algorithm used in classification tasks. In particular, the Figure lists the BVQ properties: “BVQ” is the same of “MAMR_LVQ” and “Bayes Veto Quantizer”, the output of BVQ can be used in input to a feature extraction algorithm based on LVQ model (BVQFE) as well as to an algorithm extracting the symbolic form of the Voronoi diagram, and so on. Finally, KDDONTO is useful both for service discovery and for giving a proactive support to the user, for instance in the choice of the most suitable tools w.r.t. her/his KDD problem, as well as in tools composition. So, besides being used by service broker, the ontology is exploited by some advanced services of the KDDVM platform.
6 KDDBroker: a broker for KDD

This section is devoted to present the KDDBroker, a broker for the publication and discovery of KDD services. To this end, we first illustrate the approach we use to extend UDDI registry, and then we propose an implementation of the broker.

First of all, let us remember that the main goal of a service broker is to help the user in obtaining information about the tools. This information can be found in the UDDI registry, while more detailed technical information is in WSDL. Hence, the first step is to analyze the KDTML main elements, in order to evaluate whether they can be described by service oriented standards, in particular UDDI and WSDL. Information contained in the Location and execution section of the KDTML is standard information, so WSDL is sufficient to describe it. On the other hand, although WSDL has some tags to describe I/O, the KDTML I/O interface section introduces additional tags that have not a correspondence in WSDL. In order to represent all the information about the I/O of KDD tools we defined an extended WSDL. At this point, as usual, a tModel is used to address the extend WSDL of the service. Several examples of extended WSDL describing different KDD services are available at [16]. As regards taxonomy and linkable modules sections, the extension of the UDDI registry regards the use of categoryBag structure, included in the businessService entity. This structure describes a new property called “algorithm”, whose value represents the name of the algorithm implemented by the service. Furthermore, the categoryBag points to a tModel addressing the ontology where the specific algorithm is classified. In Figure 4 we show the overview of our solution. The

![Fig. 4. Overview of UDDI architecture.](image)

adopted solution presents the following advantages:

- the standard UDDI structure is not modified, hence more traditional services and APIs can coexist in our broker;
- the choice of representing taxonomy information by an ontology allows us to implement reasoning capabilities;
- adding more categoryBag instance, we can describe the service by different concepts in different ontologies. The use of multiple ontologies is specially relevant in a collaborative environment, since it allows to model different pieces
of domain knowledge by independent experts. However, the use of multiple ontologies leads also to consider integration and reconciliation issues.

At present the KDDVM project uses Apache as web server, Jakarta Tomcat 5.5.9 [2] as the application server, and Axis 1.0 RC1 [3] as the SOAP Engine. Tomcat makes available the KDD services to users, while Axis allows to receive, to translate and to send SOAP messages. So, in order to build the service broker we choose JUDDI 0.9 RC4 [15] and UDDI4J 2.0.5 [26], that in the literature are recommended as the best solution for implementing an UDDI-based broker over an Apache-Tomcat server. JUDDI defines the data schema of the registry, while UDDI4J is a Java class library that provides a set of APIs to interact with the registry.

We developed new APIs in order to allow the discovery and publication of specific information about KDD services. In particular, for the semantic discovery we introduce two main methods:

- `ontoInquiry`, that returns the instances of the ontology satisfying a given RDQL¹ query;
- `semInquiry`, that returns the KDD services implementing a given algorithm described according to a given ontology.

Combining these APIs, we obtain various functionalities, like: search of KDD services on the basis of methods and/or tasks, search of KDD services that implement a given algorithm, search of KDD services that can be linked up to other ones, and publication of KDD services and their properties.

Figure 5 and 6 show two screenshots of the Service Broker Client, that provides an user-friendly interface to KDDBroker. In the example we search for KDD services implementing a Classification task, and in particular a Vector Quantizer method. As result, the broker provides the name of KDD services satisfying the above requirements, together with additional information: the WSDL address, the implemented algorithm, a brief description of service behavior, the Business providing the service, linkable algorithms and information about the QoS of the provider which, at present, is simply stated as estimated availability.

7 Conclusion

In this paper we present results of an ongoing project for the design of the KDDBroker, an UDDI-based service broker for publication and discovery of KDD services. To this end we also introduce KDDONTO an embryonal ontology for representing the KDD domain. Including ontology in the UDDI allows us to enrich the broker with various semantic discovery functionalities. We also discussed some issues related to the current implementation of the service broker and showed an user-friendly interface designed to easily enquiry the broker. This work is part of the Knowledge Discovery in Databases Virtual Mart project, a

¹ RDF Data Query Language: http://www.w3.org/Submission/RDQL/
more general project for the development of an open and extensible environment where users can look for implementations, suggestions, evaluations, examples of use of tools implemented as services. Future work will be devoted to expand the KDD ontology with more relevant concepts about the domain, and to endow the service broker with more advanced searching capabilities. An other interesting direction to investigate is to represent the whole registry by OWL-S, an OWL-based Web Service Ontology.

8 Acknowledgment

The authors would like to thank Emanuele Storti for his contribution to the implementation of the Service Broker.

References