Developing an Open Knowledge Discovery Support System
for a Network Environment

Claudia Diamantini, Domenico Potena, Maurizio Panti
Dipartimento di Ingegneria Informatica, Gestionale e dell’Automazione
Università Politecnica delle Marche
via Brecce Bianche, 60131 Ancona, Italy
{diamanti,d.potena,panti}@diiga.univpm.it

ABSTRACT

Knowledge Discovery in Databases (KDD) is a highly complex process where a lot of data manipulation tools with different characteristics can, and in fact have to, be used together in an interactive and iterative fashion, to reach the goal of previously unknown, potentially useful information extraction. In this paper we analyze the major sources of complexity in the framework of network organizations, pointing out the necessity to give support to the user in many different ways and at very different levels of granularity, from the use of a single tool, to the management of whole, distributed, KDD projects. Unfortunately, currently available systems lack to support the users in at least some of these features. We then propose a solution based on the Service Oriented Computing paradigm, arguing that the advantages of this paradigm, namely openness, modularity, reusability and transparency, as well as ubiquity, can help in the design of an effective support system for Knowledge Discovery in Databases in network environments.

KEYWORDS: Architecture and Design of Collaborative Systems, Human-Centric Systems, Grid-Based Collaborative Environments.

1. INTRODUCTION

The current market and the ICT technologies induce organizations to modify their internal structure towards a cooperative network of autonomous entities, and to augment their information processing capacity. This implies the formation of a distributed decision process, where each decision maker operates in its business unit and coordinates himself with the other decision makers at the same level or at different levels. Different kinds of decisions, that have to be taken, are based on the appropriate form of knowledge which thus has to be at disposal of decisions makers at the different network nodes. In order to support decision makers, decision support systems have been developed based on different techniques like for instance OLAP and reporting systems and Data Warehousing. Knowledge Discovery in Databases (KDD) is the research field studying advanced techniques and methodologies aimed to the extraction of previously unknown, potentially useful knowledge [7] without a predefined search pattern, giving more sophisticated and powerful means to take decisions. A KDD process is a highly complex, iterative and interactive process, with a goal-driven and domain dependent nature. For this reason, it is convenient to envisage the design of a support system for KDD process design, giving the user a set of basic tools and an effective environment for their use. For effectiveness we mean the capability of a KDD system to make, in any sense, the activities of its users productive. In the framework of network organizations the following main issues influences effectiveness:

- network organizations can be formed of small business units with a limited budget for KDD projects and with limited internal skill, then they are typically prevented from the use of KDD tools or, at least, they use KDD techniques on limited, simple and repetitive problems. Also, they can lack sufficient data on which KDD techniques can be applied successfully. In this case, the key to effectiveness is the capability to bridge the gap between the necessary knowledge to achieve a KDD goal and the available one. Also, it is desirable to have a sort of “KDD service on demand”, that is a system which is able to give the minimal set of functionalities required, growing with the needs of the user. For larger business units the knowledge gap can be still a problem, as they manage bigger and more various KDD projects, at very different levels of the business organization.

- In a network organization KDD processes are distributed and collaborative in nature, since data, computational resources, expertise and information are spread all over the network. Then some problems have to be issued, like heterogeneous databases manage-
ment, resource sharing and collaborative computing and working.

- The huge amount of tools for data manipulation and the continuous development of new ones, the different performances shown by different tools on different kind of data, implies an effort for the discovery, retrieval, implementation and experimentation of different algorithms and techniques, in order to find out the one which best matches the specific problem and data at hand. This effort is emphasized when the network is formed by researchers or consultants in the KDD field.

From the above description, it can be deduced that effectiveness is characterized by a number of very different features, basically because of both the complexity and variety of the business goals, the differences in the characteristics and sophistication degree of tools and users, and the characteristics of the KDD process itself. Thus, the design of a KDD system should guarantee a good overall effectiveness. To this end, in the present paper, we analyze the KDD domain in terms of the major sources of complexity an user is faced with, in order to define a comprehensive set of requirements of an ideal KDD support system. From this analysis, it turns out that these requirements can be summarized by the openness, modularity, reusability and transparency keywords. Modularity, flexibility, transparency, and reuse are the motivations of the recent introduction of new loosely coupled, cooperative paradigms of computation, that are named, with slightly different meanings, component based computing, or service oriented computing, and that find in Web Services [4, 16] the emerging reference architecture. Thus, we conclude that this paradigm is naturally suited to design an effective support system for Knowledge Discovery in Databases. Then, we propose a system of services aiming to cover the fundamental features of such a support system.

In the literature, solutions adopting similar paradigms exist, [9, 11, 15, 2], but they are mainly concerned with large-scale and high-performance issues, and they focus mainly on the Data Mining phase (see also [13] for an extensive survey), without considering the KDD process as a whole. As a consequence, they share with the present proposal some basic feature, but they lack to include higher-level supports to the user. In particular, [9, 11, 15] share with the present paper the emphasis on compositionality, reusability, annotation. In [11] the authors describe an agent-based parallel and distributed data mining system. The agents are charged with mining knowledge from distributed databases and are coordinated by the facilitator. As in our proposal, [11] provides a world wide web based user interface for a visual interaction with the system. In [15] it is introduced a system based on the CORBA technology, that supports component composition by a dataflow mechanism driven by a domain specific knowledge base, the wrapping of existing algorithms as Java/CORBA objects and the description of the component structure and interface by XML. We believe that the use of Web Service technology, instead of agent or CORBA, enhances the ubiquity and openness properties of our system. In our proposal, each element of the system is viewed as a service, and thus can be spread and exploited across the network. Furthermore, such an open architecture allows to design user-centric client applications, exploiting only those services which are from time to time needed. The component composition in [15] is based only on syntactical compatibility of the outputs of a components with the inputs of the next one. The service vision, and more generally the recent paradigm shift towards distributed and cooperating global network computing, leverages a richer semantic annotation of components [6, 1, 10]. In this general mainstream, we envisage a new type of services which allows a more intelligence support to the design of KDD processes. In the same mainstream, Knowledge Grids have been introduced, as a means of implementing distributed and parallel Data Mining applications on the top of a grid architecture [2]. The main focus of the approach is on high-performance parallel distributed computing and on generic high-level services for knowledge management and discovery, while the present work is mainly on the identification of specific high level services to support the design of KDD processes. Thus, we believe that the two approaches could integrate and benefit from each other, especially in the light of the OGSA proposal [19, 1]. The idea of data mining models as services on the Internet has been introduced in [17], mainly with the aim of facilitating the use of such techniques among naive users. However, it concerns single, ready to use models, especially classification models, and it does not address design issues. Recently, a service oriented infrastructure for distributed data mining has been presented in [12]. This approach introduces services for registration, discovery and execution of data mining services, without discussing services supporting the design, the composition and the management of the whole KDD process.

The rest of the paper is organized as follows: section 2 discusses in more detail the characteristics of a KDD process and the requirements of an ideal KDD support system. In section 3 the main services for the satisfaction of such requirements are introduced. Section 4 gives some concluding remarks.

2. REQUIREMENTS OF A KNOWLEDGE DISCOVERY IN DATABASE SYSTEM

Methodological standards [7, 18] suggest to divide the design of a KDD process into six principal phases: Business Understanding, Selection, Preprocessing, Transformation,
Figure 1: KDD Process Phases Schema.

Data Mining, Interpretation/Evaluation (see the schema in Figure 1). Each of these phases refers to a set of homogeneous activities:

1. Business Understanding: The initial phase aims to the understanding of the business domain from the point of view of both the project goals and requirements, and the available data, to get familiar with them and to form preliminary hypotheses for hidden information. This knowledge is converted into the formulation of a Data Mining problem, and a preliminary plan of activities;

2. Selection: it can be understood as a sort of database query to select the dataset of interest; however data are not necessarily structured;

3. Preprocessing and Data Cleaning: filtering of the noise, removal of the outliers, treatment of missing data fields, accounting for time sequence information;

4. Transformation: reduction of data dimensionality, either by projection or by some form of combination of data components, analysis of the distribution function, identification of invariant representations of the data;

5. Data Mining: models induction by Classification, Clustering, Associative rules, Regression or other techniques;

6. Interpretation/Evaluation: graphic and tabular representation of the results, symbolic representation of the models, creation of summary reports, evaluation of performances, etc.

The first phase is a kind of a feasibility study involving at a great extent the analyst creativity and experience. It cannot be fully automated and, at present, even limited support tools exist, but for the manipulation and exploration of data. Each of the other listed activities can be practically realized through many different techniques, coming from many different research areas, such as Statistics, Pattern Recognition, Machine Learning, Artificial Intelligence, Databases and Visualization. For instance, data cleaning can be performed by exploiting statistical tools, from the simple calculus of mean values to sophisticated estimation of most probable values, data mining can be performed by algorithms such as Support Vector Machines, C4.5, apriori, eclat, just to name a few. A more detailed description of the Data Mining phase, in terms of activities, algorithms and their characteristics is available in [1]. The existence of a great amount of techniques and tools does not limit for itself the complexity of the task, in fact it presupposes a certain degree of acquaintance with the tools and with the mathematical models underlying them, so to chose the most appropriate one and to use it correctly and effectively.

Analyzing the problems reported by expert as well as naive KDD users, we recognize and discuss three major sources of complexity in the design of a KDD process: (1) Mapping general KDD models into the specific business domain; (2) Retrieving and managing data and algorithms; (3) Managing the whole life cycle of a KDD process;

1) Mapping general KDD models into the specific business domain

In the first phase of the KDD process, one has to find out the best representation of the business goal in terms of data mining goal, and to envisage the best data structures and techniques to achieve that goal. In practice, this means drawing a rough model of the business and of the entities involved in, to form some model-driven hypothesis on the kind of regularities which can be found in data, and to focus the search towards the most appropriate techniques. The complexity of this process is mainly related to the cultural, knowledge and terminological gaps between business and KDD domains. Creativity, mastery of a methodology, and experience are at present the only way to reduce that gaps.

2) Retrieving and managing data and algorithms

The huge amount of tools for data manipulation, exploration, analysis and mining, and the continuous development of new ones stresses the limit of ’closed’ existing commercial systems. Then, a user has few alternatives:

1. he can limit himself to choose among the tools provided by a commercial system. In this case, some management support is given, but he could miss the chance to obtain better results by newer or more complex techniques. Some problems or data could not simply be treated by the system at hand;

2. he can extend the functionalities of a single commercial system by using it in combination with other commercial systems and/or algorithms either (a) implemented by the user or (b) retrieved from the Web. If the user is in a small business unit, that often works with limited, simple and repetitive KDD problems and, consequently, needs a limited set of techniques,
the adoption of commercial tools is a right choice. Facing, instead, with various and frequent KDD problems, the user has to be able to choose among many tools and to combine them in effective and efficient way. So, if an user wants to fully exploit the potential of KDD for his goals, then he has to take charge of retrieval, use, interoperability and integration issues. The user has to know where to find useful algorithms and, in any case, how to assess the relevance of an algorithm or tool for his goals. He has to know how to correctly and effectively use the algorithms. He has to manage the implementation and the activation of algorithms into the different running environments, transforming the generic data structure so that it matches the input format of the chosen implementation. He has also to collect the outputs, linking them to the KDD project and to the algorithms inside it, together with the parameters used in the call.

(3) Managing the whole life cycle of a KDD process

The intrinsic complexity of the design of a KDD process is due to the numerous degrees of freedom the user has to work with and to the goal-driven and domain dependent nature of the problem. As said before, the user should be able to acknowledge the tools more conformed for its task, and this can often be accomplished only by trial-and-error and comparison steps. This activity presupposes some degree of technical skill, however, the user is typically a domain expert, but not an expert of all the implementable algorithms.

Another source of complexity is due to the intrinsic features of any discovery process, namely the lack of knowledge and consequently the difficulty to a priori define the best plan to discover that knowledge. This fact is recognized in all the existing process models (see e.g. [7, 18]) by accounting for the need of repeated backtracking to previous phases and repetition of certain actions: the lessons learned during a phase can help recognizing errors in previous phases or can give knowledge to enhance their outcomes. Backtracking and comparison of different trials have to be managed in terms of intermediate data, results, hypothesis, algorithms and so on.

The whole KDD process is cyclic in nature. A KDD process continues after a solution has been deployed. The results learned during the process can trigger new, often more focused business questions. Subsequent KDD processes will benefit from the experiences of previous ones [18]. Finally, let us notice that, in a collaborative environment, different users participate to the same KDD project, being assigned to different KDD phases. Hence, problems arise about user interoperability, and about the integration of the final results coming from the single phases.

From the previous discussion on the characteristics and difficulties of the KDD domain, we arrive at the definition of the following requirements of an effective KDD process design support environment:

- **Completeness**: the system should have the possible richest variety of tools, in order to extend its applicability to, and find the best techniques for, different problems, domains and sources;

- **Versioning**: with this term we define the necessity of give support to both the management and the comparison of different alternative solutions. This implies to keep track of the experiments and to manage in efficient way the intermediate data. Also, to enable extensive experimentation, it is hoped that the system allows to evaluate the performances after each change of the components or of the parameters in a KDD process. This characteristic evolves towards the intelligent management of KDD processes similarities and the reuse of previous solutions;

- **Use easiness**: the system should give support to different categories of users, ranging from domain experts with little knowledge of methodologies and theoretical principles, to researchers in the KDD field. This implies the introduction of a conceptual level of tool manipulation, allowing to hide implementation and technical details. It should propose an user friendly interface, preferably of graphical type. Besides, a proactive support is hoped for the design of a process, which manages the constraints on the linkability of tools and suggests possible process prototypes;

- **Transparency**: with respect to the implementation, to location and call of the algorithms and with respect to the access to the data to be analyzed, with the possibility to manage one or more heterogeneous and distributed databases;

- **Flexibility**: the system should manage the introduction of new algorithms or releases easily and the deletion of existing ones. It should also have a simple and flexible mechanism for tool composition;

- **Reusability**: the system should support the use of algorithms already implemented without modifying the code. It should also support the use of previously generated pieces of knowledge.

In order to satisfy the requirements of completeness and flexibility the system should be open and modular, and to satisfy transparency and reusability the system should be based on a descriptive language (such as XML). Notice that the use of XML-based descriptive languages is the current reference solution to manage the semantic issues related to interoperability and integration. These characteristics are the motivations of the recent introduction of a new loosely
coupled, cooperative paradigm of computation, which is named Service Oriented Computing and which finds in Web Services the reference architecture. The Web Service descriptive languages are XML-based: Web Service Description Language (WSDL), Universal Description Discovery and Integration (UDDI) and Simple Object Access Protocol (SOAP). WSDL is used to give information about both the description and the binding of the service. The UDDI is the language to build the broker registries, that organize web services description by name, functional categories and technical details. SOAP is an XML-based protocol allowing the messages exchange between service and user or service and service. Web Service technology enhances the openness and modularity properties of a system. Such an open architecture allows to design user-centric client applications, exploiting only those services which are from time to time needed. Services can be spread and exploited across the network, using a simple world wide web based user interface. Differing from the classical client-server architecture, service identification and location is transparent to the user, since this information is centralized to the broker service by the UDDI registries. Furthermore, the service vision leverages a richer semantic annotation of components.

3. SERVICES FOR KDD

In order to guarantee the satisfaction of the above requirements, we introduce three different levels of service: basic services, support services and advanced services (Fig. 2).

3.1. Basic Services

Basic services are all the algorithms possibly involved in a KDD process, from data manipulation, exploration and analysis, to model induction and exploitation, to visualization of results. Also, the management of data needs a family of basic services. In order to interface data sources, either local or distributed, and to manage heterogeneity, replication and any other issue related to the physical distribution of data, our proposal can be built on the top of a Data Grid architecture [3]: as a matter of fact, exploiting the OGSA proposal [19], grid resources can be integrated easily in our platform as basic services. We do not define a priori the number and type of services which can be found at this level, rather we assume an open architecture, which is the only way to ensure completeness and flexibility requirements.

3.2. Support Service

Support services are the core services of the architecture, which substantially ensure the transparency requirement. One of the most important functions is that of registration and binding of basic services. This special function is fundamental to keep track of all available services in an open environment and in fact it is an element in all implemented architectures of this type: it is called the facilitator in multi-agent systems, or the broker in the Web Service as well as in the CORBA Architecture. Another support function is that of service localization, that is the autonomous discovery of new services and the automatical feed of the relevant information into the broker registries.

In the previous section, we said that reusability is one of the principal qualifications of an ideal KDD system. This means the capability to use implementations developed in any programming language and put at disposal by others without modifying or rewriting the code. A wrapping service is then introduced, to create a wrapper encapsulating any software module. To this end, the wrapping service uses information from the software developer about the description, location and execution of the KDD module. It translates this information in WSDL and stores it in the module wrapper for future requests. The same WSDL is used to update the registries of the Broker Service. An important function of the wrapper is the management of KDD module execution, that is both the translation of data from the RCP-SOAP request to the internal schema and vice versa, and the call of the software module. At this point, the pair formed by the module and the related wrapper really constitutes the basic service. At the web site http://\babbage\diiga\univpm.it:8080\axis\ it is possible to experiment the use of the wrapping service as well as of some Data Mining wrapped software.

One of the advantages of the use of the service oriented paradigm for a KDD support system is to view the formation of a KDD process as a composition of services. This allows to borrow all the theoretical results on generic web service composition from the literature [8, 14]. We define two basic composition service types: those related to generic processes, let call it workflow-like, and those more specifically designed for KDD processes. Workflow-like composition services are considered support services,
Middle DataBase

Data sources or from an internal database storing all data come either from basic services managing the external data sources or from an internal database storing all data. The LAS is responsible for the preparation of input data of a basic service implementing the SOM algorithm, the WMS translates the process built by the user in a workflow and interprets it, verifying its syntactic correctness and activating the necessary services. The WMS interacts with the SLVS to translate the process and with the LAS to prepare the data in input to the basic service called; furthermore, WMS needs information by the broker to call the basic and support services. For instance, let us suppose that a simple classification process is built by an user after interaction with the Broker Service, and let us suppose that the chosen classification algorithm is the Bayes Vector Quantizer (BVQ) [5]. The process steps include (1) the selection of data from the input database and (2) the training of the BVQ net, starting from an adapted SOM net. This process is verified by the SLVS and translated by the WMS in an XML Workflow. To this end, the SLVS reads the input/output information from the web service description (in WSDL) verifying the syntactical linkability of any pair of two consecutive services. In the scope of our system, two services are defined to be syntactically linkable if the input of the consequent is a transformation of a subset of the output of the precedent. The translated XML Workflow is showed in Table 1. Now, the WMS has to activate the workflow. Before any basic service call, the WMS interacts with LAS to prepare the input data of this service. To this end, the LAS uses the web service description of both the service that has generated the output and the service will be called. For instance, the output of the query to the database service will be transformed by the LAS to an ASCII file arranged according to the SOM file input format. To activate the basic service implementing the SOM algorithm, the WMS builds and sends the related Remote Call Procedure SOAP (RCP-SOAP) request (see table 2). When the computation ends, the basic service sends the RCP-SOAP response to the WMS, and so on until the end of the process.

### Table 1: The Workflow.

```
< service >DataBase Service
  < operation name >Select Query </ operation name >
  < input string >SELECT att1 as feature1 ... 
    , attN as class FROM ... </ input string >
  < output >
    < output_name >datain </ output_name >
  </ service >

< service >SOM
  < operation name >SOM </ operation name >
  < input >
    < neurons_num xsi:type="xsd:integer">4</neurons_num>
    < iteration_num xsi:type="xsd:integer">1000</iteration_num>
    < train_param xsi:type="xsd:real">0.5</train_param>
    < data xsi:type="xsd:file">datain</data>
  </ input >
  < output >
    < output_name >SOM_model </ output_name >
  </ output >
</ service >
```

while the others are more sophisticated services at the level of advanced services and will be described later. Workflow-like composition services manage basic logical controls, like sequence, conditional branching and cycles, as well as those related to the execution of concurrent processes. The workflow-like composition services we identified are: the Syntactical Link Verification service (SLVS), the Link Adapter service (LAS) and the Workflow Manager service (WMS). The SLVS verifies the syntactical linkability of a pair of services. Two services are defined to be linkable if the input of the consequent is a transformation of a subset of the output of the precedent. The LAS is responsible for the preparation of input data of a basic service, that involves the translation of data structures. These data come either from basic services managing the external data sources or from an internal database storing all the outcomes of previous elaborations. This database is named Middle DataBase. We will give a description of this database later. The design of a KDD project is accomplished by a set of actions which can be described as a workflow. In fact, a KDD project can be defined as a set of related KDD (sub)processes which, in turn, are formed by sequences and cycles of actions. The relation among (sub)processes implies comparisons and conditional choices, as well as the management of concurrency. For instance, an user could design a project where two different classification algorithms are run in parallel on the same data, continuing to use the model with the best performance, or he could combine the results of both. Hence, it is natural to introduce a workflow manager service. The WMS translates the process built by the user in a workflow and interpret it, verifying its syntactic correctness and activating the necessary services. The WMS interacts with the SLVS to translate the process and with the LAS to prepare the data in input to the basic service called; furthermore, WMS needs information by the broker to call the basic and support services. For instance, let us suppose that a simple classification process is built by an user after interaction with the Broker Service, and let us suppose that the chosen classification algorithm is the Bayes Vector Quantizer (BVQ) [5]. The process steps include (1) the selection of data from the input database and (2) the training of the BVQ net, starting from an adapted SOM net. This process is verified by the SLVS and translated by the WMS in an XML Workflow. To this end, the SLVS reads the input/output information from the web service description (in WSDL) verifying the syntactical linkability of any pair of two consecutive services. In the scope of our system, two services are defined to be syntactically linkable if the input of the consequent is a transformation of a subset of the output of the precedent. The translated XML Workflow is showed in Table 1. Now, the WMS has to activate the workflow. Before any basic service call, the WMS interacts with LAS to prepare the input data of this service. To this end, the LAS uses the web service description of both the service that has generated the output and the service will be called. For instance, the output of the query to the database service will be transformed by the LAS to an ASCII file arranged according to the SOM file input format. To activate the basic service implementing the SOM algorithm, the WMS builds and sends the related Remote Call Procedure SOAP (RCP-SOAP) request (see table 2). When the computation ends, the basic service sends the RCP-SOAP response to the WMS, and so on until the end of the process.

### Table 2: An Example of RCP-SOAP.

```
< SOAP : body >
  < m : SOM 
    .... >
  < input >
    < neurons_num xsi:type="xsd:integer">4</neurons_num>
    < iteration_num xsi:type="xsd:integer">1000</iteration_num>
    < train_param xsi:type="xsd:real">0.5</train_param>
    < data xsi:type="xsd:file">datain</data>
  </ input >
</ SOAP : body >
```
3.3. Advanced Services

On the top of the set of services defined above, one can define a number of different types of advanced services, that is "intelligent" services supporting the user. In what follows we will introduce only some services allowing to ensure the versioning and ease easiness requirements. These services have two principal tasks: to manage the versions of a process and to reduce the knowledge gap between business and KDD domains.

The versioning management principally involves the management of the Middle DB, a database storing information about the whole process and the intermediate data generated during the process. Information about the process involves: the business domain(s) it is used for, the business and KDD goals and the performance of the process for each domain and goal. Information about intermediate data involves: the initial input data, the basic services call, the parameters used in the call, their outputs and performance parameters. Depending on the size of data, the cost of the computation and other characteristics of the services like the waiting time, inputs and outputs data can be either stored in explicit form or not. A special type of output is given by the models generated by Data Mining algorithms. These typically describe information in synthetic form, and could be reused for subsequent applications, in combination with the appropriate basic service [17]. Furthermore, several models (possibly of different types) can be associated to the same data set, due to the application of different algorithms. Thus, the models have not a standard form and have a big semantic content. For these characteristics, it is worthwhile envisaging an explicit model storage. The problem of model representation and management is an open research problem [10]. We consider the model as data stored in the Middle DB. The management of the Middle DB is entrusted to the Middle DataBase Manager Service. The Middle DB is designed to directly manage XML service description (WSDL) and calls (RCP-SOAP), with a PMML-like model representation [10].

To join a process with its performance indexes, the architecture needs an advanced service named Assessment service. The Assessment service interacts with the Middle DB Manager to extract all the possible performance indexes: the algorithms performance estimators (e.g. error probability, precision and recall) and the computational indexes (e.g. execution time and memory). This information can, in case, be enriched also by some form of feedback and notes by the users.

The information contained in the Middle DB is exploited to help the user in KDD process composition, both by taking advantage of the reuse of previously executed processes and by giving proactive support to reduce the knowledge gap between business and KDD domains. To do this, a service for finding similarities between sub-processes is introduced, called the Versioning Manager service (VMS). In the case an autonomous user has already designed a process, the WMS can interact with the VMS to look for identical subsequences stored in the Middle DB. If they are found, then the user can exploit the work already done. Similar subsequences can also be useful, for instance the user can reuse the subsequence on different data, by exploiting previous parameter setting, or they can be presented as a sort of running example to explain the basic service use and scope. In this direction, similarity search become an important tool also to proactively support naive users. We can go a step further, by considering similarities not only with respect to subsequence composition, but also with respect to the goals and domains they are used for. Definition of similarities among business domains, data structures and types, data mining and business goals, allows us to introduce Meta-Learning services, that are able to define relationships among clusters of similar data, classes of algorithms and business goals [6]. This information is stored in the Middle DB and is exploited by a set of services, named case-based support services, that can suggest to the user tools and/or subsequences possibly suitable to his specific case. For instance, a domain expert can be assisted in mapping his business goal into some data mining goals, by comparing the present business domain with those of previously executed processes. The system can suggest the tools which have been demonstrated the best performance on similar domains or similar kind of data. Finally, it can suggest how to proceed in the process construction, by showing the services or whole sub-processes most frequently used starting from that point.

The advantage of this similarity search function is leveraged in an open cooperative scenario. As matter of fact, the Middle DB can be easily thought of as a distributed database (given by the union of different Middle DBs already populated by various users) and, therefore, the user can search the best tools, for his goals, in a large repository even if he has only few positive trials or he is using the system for the first time. Furthermore, linking the data of the Middle DB with the user has produced them, we can evaluate the best tools, not only according to the indexes produced by the assessment service, but also on the basis of the reliability of the user.

In order to realize the proposed services, it is fundamental to have descriptions of both the application and KDD domains. Such descriptions can be easily related to the notion of ontology. A KDD ontology is basically built up by resorting to the phase structure showed at the beginning of the previous section [1]. A domain ontology is defined starting from the relationship between business goals, previously presented to the system, and data. The detailed study of this problem has not been considered yet, and will be the subject of future work.
4. CONCLUSIONS

The paper gives an overview of the KDD domain and of its criticalities in the framework of network organizations, from which the requirements of an ideal KDD support system are drawn. The main requirements can be summarized by the openness, modularity, reusability and transparency keywords. Hence, it is argued that a Service Oriented Computing paradigm can help to design a system supporting the KDD process design under the above mentioned requirements. The analysis introduces some more requirement specifically focused on improving the effectiveness of the system and support to the user, in particular for the management and the comparison of different alternative solutions, as well for the reuse of previous solutions. Then, a set of categories of services, of data, and their relationships, is introduced to satisfy such requirements. The result is a general architecture which could serve as a reference in the implementation of KDD services.

REFERENCES