Doctoral School on Engineering Sciences
Università Politecnica delle Marche

Extended summary

KDD Process Design in Collaborative and Distributed Environments

Curriculum in Ingegneria Informatica, Gestionale e dell'Automazione

Author

Emanuele Storti

Tutor

Prof. Claudia Diamantini

Date: 31-01-2012

Abstract. Knowledge Discovery in Databases (KDD), as well as scientific experimentation in e-Science, is a complex and computationally intensive process aimed at gaining knowledge from a huge set of data. Often performed in distributed settings, KDD projects usually involve a deep interaction among heterogeneous tools and several users with specific expertise. Given the high complexity of the process, such users need effective support to achieve their goal of knowledge extraction. This work presents the Knowledge Discovery in Databases Virtual Mart (KDDVM), a user- and knowledge-centric framework aimed at supporting the design of KDD processes in a highly distributed and collaborative scenario, in which computational resources and actors dynamically interoperate to share and elaborate knowledge. The contribution of the work is two-fold: firstly, a conceptual systematization of the relevant knowledge is provided, with the aim to formalize, through semantic technologies, each element taking part in the design and execution of a KDD process, including computational resources, data and actors; secondly, we propose an implementation of the framework as an open, modular and extendable Service-Oriented platform, in which several services are available both to perform basic operations of data manipulations and to support more advanced functionalities. Among them, the management of deployment/activation of computational resources, service discovery and their composition to build

KDD processes. Since the cooperative design and execution of a distributed KDD process typically require several skills, both technical and managerial, collaboration can easily become a source of complexity if not supported by any kind of coordination. For such reasons, a set of functionalities of the platform is specifically addressed to support collaboration within a distributed team, by providing an environment in which users can work on the same project and share processes, results and ideas.

Keywords. Collaborative technologies, KDD Process, Knowledge Discovery in Databases, Process Design

1 Problem statement and objectives

The rapid growth of databases in last years asks organizations to deal with issues related to the management of large amounts of data, which represent a valuable resource for decision making. Although technologies for data management/storage are widely available, much effort is still needed to provide users with systems for effectively analyzing and understanding data. We use the term Knowledge Discovery in Databases (KDD) to refer to the non-trivial process of extracting interesting, valid and useful patterns from data [1]. As a process, it often involves several steps, which may include: selection of a subset of data from the whole dataset, data cleaning and transformation, feature extraction, choice of the appropriate Data Mining technique for extracting patterns, its configuration and execution, evaluation and interpretation of results and deployment of new knowledge to users [2]. Especially for non-experts, definition and management of a KDD process are themselves demanding activities, because they require user to know how to choose the proper tools among the plethora of available ones, how to setup them, how to interpret their output. In order to manage a KDD process, a team of different experts is usually needed, each of which is able to configure only a part of the whole process. Such a team either can belong to the same organization or can be a geographically distributed virtual team of experts. Hence, integration of distributed users and tools, along with heterogeneity of these latter are issues to take into account in order to define effective solutions for the problem at hand. Real scenarios for KDD may include not only network organizations, for which distributed cooperative KDD projects may represent a significant added value, but also the support to e-Science processes (e.g. particle physics, earth sciences, and bioinformatics). In such a highly distributed environment, in fact, scientists need technologies for a collaborative analysis of data produced by scientific experimentations [3].

Various systems have been proposed for supporting users in the design and management of KDD processes. However, so far, to the best of our knowledge no solution can be found both in commercial software and in the Literature to properly face all above mentioned challenging issues. About the former, commercial KDD platforms typically provide tools for building a KDD process out of a set of algorithms, but users have to manually compose the workflow. Often, tools can be executed only locally with little or no actual chance for extension/integration with complex support functionalities. Especially in last years several work, available in the Literature, faced more advanced topics like effective retrieval of processes, specialized support to collaborative process design, specification and reuse of common practices for the usage of tools, semi-automatic goal-oriented process composition [4, 5], and integrated management of versioning. However, most of such proposals are mostly concerned with large-scale and high-performance issues [6, 7], or focusing mainly on Data Mining phase without considering the KDD process as a whole. Then, although many supporting environments have been designed for cooperative work [8], only recent ones consider collaboration in KDD, and very few of them with a knowledge representation perspective.

For the above reasons, there is growing need of effective systems for supporting users, especially non-experts, in the design of a KDD process, in particular in the localization of interesting tools (possibly remotely available and produced by some external organizations or research groups), in the choice of the most suitable tools for a given problem, in their composition and execution.

The aim of this work is a systematization of the knowledge involved in the design of KDD processes, and the following exploitation in order to build an effective CKDD sys-

tem, i.e. a KDD support system for Collaborative and distributed environments. This work represents a substantial revision from a theoretical perspective of a previous work by Diamantini et al. [9], and an extension from an applicative and functional point of view.

2 Research planning and activities

Given that a KDD process involves a deep interaction among users and tools in order to perform data analysis operations, we recognize four main typologies of resources that must be managed by a collaborative KDD platform: *computational units*, *computational processes*, *data* and *actors*. The main contributions of the work are defined as follows.

Knowledge about each resource is fully described from different levels of abstractions by a Knowledge Layer that includes specific languages and descriptors for each of such levels, which are interrelated in order to provide machine-readable mappings among them. The exploitation of semantic information is one of the key elements of our *knowledge-centric* approach, as they provide, at a conceptual level, the needed terminology which the concrete level descriptors can refer to. Semantic technologies are widely recognized as the state-of-the-art solution to explain and formally explicit the meaning of the objects in the domain, towards a new generation of systems with advanced intelligent functionalities for collaboration in large heterogeneous, distributed environments. In particular, we refer in this work to domain ontologies, which have been designed following a formal methodology and quality criteria. Formalism in concepts definition allows to support inferential mechanisms, to find non-explicit relations among the ontological concepts.

The Knowledge Layer provides the data model on which an open and modular Service-Oriented Architecture is built, which includes both services for KDD and Data Mining tasks and support services providing all the needed high-level functionalities. The platform is based on Web Service standards, which guarantee interoperability and the possibility to integrate several simple services in order to provide more complex functionalities. Among them: easy service publishing and sharing, service and process discovery, automatic process matchmaking and composition, project team building, collaborative process design and versioning. The adoption of service-oriented principles lets users focus on KDD experiments without having to deal with interfaces and message exchange. This enables a high flexibility of the system, by allowing to dynamically reconfigure a computational process using each time only the services that are actually needed.

Our focus on knowledge is also to be considered from a *user-centric* perspective, as one of the most important aims of the platform is towards a collaborative dimension. Collaboration among users is tracked and supported at the utmost, by technological solutions that help to enable effective communication and coordination.

3 Analysis and discussion of main results

3.1 Knowledge Layer

Purpose of the Knowledge Layer is to provide a systematization of the knowledge involved in a KDD process from a generic and theoretical point of view, as well as a detailed description of the relevant information needed for a concrete Collaborative KDD (CKDD) platform. Such formalization regards each kind of resource involved in a KDD process, and is based on several points of view: level of abstraction of the resource, role played by the resource in a generic KDD process, purpose for which a resource is actually used in the

KDDVM platform. An overview of the conceptual and concrete levels of the Knowledge Layer is shown in Figure 1.

3.1.1 Computational units

About computational units, according to different degrees of abstraction we recognize *algorithms*, *tools* and *services*. An algorithm is an abstract prototype of a tool, whereas a tool is an implementation of an algorithm in a concrete programming language. A service is a tool running on a server, offering its interface through standard Web Service protocols. In this way, many services can refer to the same tool, whereas several tools can implement the same algorithm. From an informative perspective, this means that all the characteristics defined at one abstraction level are inherited by the lower level(s). Such a loose-coupled approach enhances modularity and reusability, and supports advanced functionalities for discovery and composition, as will be shown in the following.

Algorithms are described into KDDONTO, a domain ontology, which also includes information about the task achieved by the algorithm, the KDD phase in which it is commonly used, the search method it implements for achieving its task, performance indices like complexity and scalability. Moreover, the I/O interface is described: the data required as input and yielded as output, together with the preconditions that such data must satisfy in order to be actually used by the tool.

Since there are many tools available for the same task and they typically have heterogeneous interfaces, we manage integration and reuse of such tools by describing them with open formats (i.e., XML-based). In such a way the user is relieved of the need to understand the service interface and to prepare data accordingly.

For KDD tools we adopt *KDTML*, an XML-based open descriptor, which is aimed at annotating a tool through a set of metadata, in order to describe its details in a structured fashion, including its interface, the algorithm it implements, its performances. Legacy tools produced by third-parties and written in any programming language can be described through KDTML in order to allow higher support for sharing and integration.

In our platform each KDD service is described by an extended-SAWSDL (*eSAWSDL*), i.e. a fully-compatible SAWSDL [10] descriptor with some additional details, namely specific syntax of I/O data or performance values (e.g., QoS, which depends on the server on which it is running, and on the network status), useful to choose the most high-performing service for a specific task, or to find a service with a certain interface, in order to support process matchmaking or composition. Although each service may be remotely executable, a public UDDI registry holds its relevant details: information about I/O interfaces, the algorithm it implements, and the URL of the eSAWSDL descriptor needed for its execution.

Our infrastructure relies on mappings among levels, such that the descriptor for a tool (or a service) refers to KDDONTO to explicit the semantic meaning of its interfaces. In the same way, there is a mapping between each descriptor and the specific ontological concept that represents the implemented algorithm.

3.1.2 Computational process

Prototype processes are abstract processes composed of algorithms defined in KDDONTO. Given that the platform is aimed at supporting process composition and management mainly at the concrete level, there is no need to explicitly represent them through a specific language.

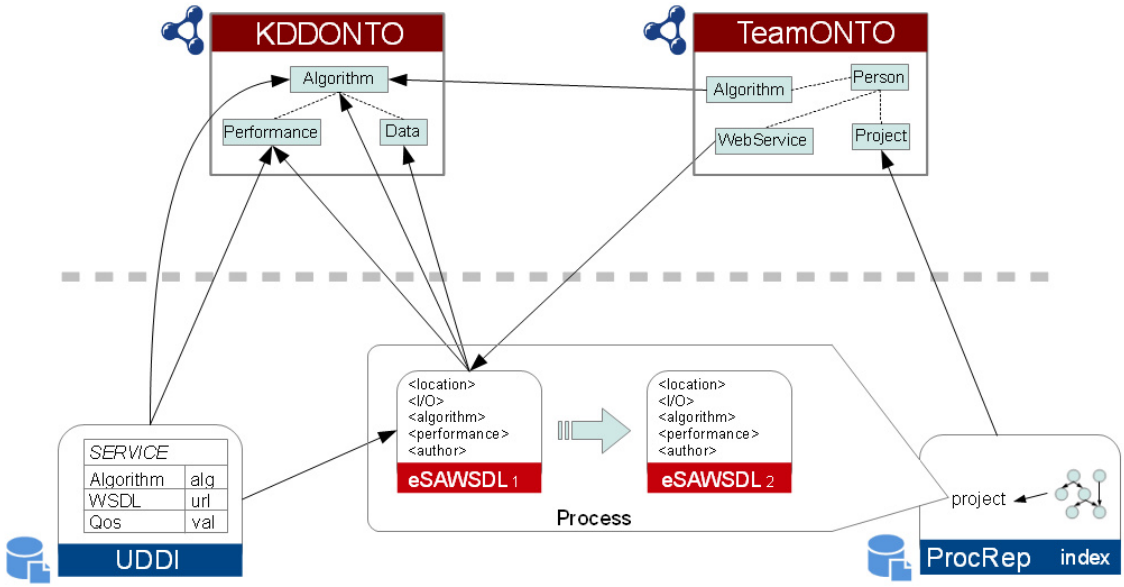

Figure 1. Overall view of the Knowledge Layer.

As regards concrete level, *process schemas* are processes composed of tools or services. In this work, however, we take into account only process schemas made of web services: in fact, the platform is aimed at distributed computation which requires a Service-Oriented approach. Process schemas are internally serialized in an XML-based language, which contains information about web services' name, eSAWSDLs of the services in the process, connections among their input/output interfaces, and the users that are in charge of their management. Moreover, the descriptor includes metadata about the whole process, such as the creation date/time, the author's name, textual comments, and the process state, to track whether it is still under development or is definitive (i.e., if it can be released).

Process schemas are then stored into a *Process Repository* (ProcRep), that arranges them in projects. With the aim to support the discovery of process schemas inside the Process Repository, as described in Section 3.2.2, an indexing system has been defined. Such indexing model is a lattice, in which each node is a sub-process structure made of web services. The link among a node and its parent node means that the sub-process identified by the first node extends the one of its parent. Each node, then, can be considered as a cluster of all those process schemas in ProcRep that include its structure: in this way an indexing strategy can be defined, that links each node with the corresponding process schemas.
Such a lattice is generated through a hierarchical clustering technique that, given a repository of process schemas represented as graphs, extracts frequent and non-trivial substructures and relations among them, putting at the top of the lattice the most frequent sub-structures [11].

3.1.3 Data

As regards data and models we refer to KDDONTO, which besides describing algorithms offers also a description of data at the conceptual level, defining data typologies in the context of KDD domain. The *Data* concepts is further specified in *Dataset*, *Model* and *Parameter*, which respectively represent the initial corpus of observation from which to begin a KDD process, the schema generated as output by some algorithms and simple parameters used for input interface setup. Moreover, a peculiar characteristic of KDDONTO is the intro-

duction of the "part-of" relation, motivated by the need to manage structured data, and to express the relation between a complex, compound datum and its subcomponents.

Such ontological concepts are useful, as already explained, to make explicit the meaning of I/O interfaces in KDTML and eSAWSDL, as an effective solution for integration and interoperability.

3.1.4 Actors

For what concerns actors, we define them in *TeamONTO*, an ontology aimed at representing the characteristics of participants in the team, such as their affiliations, their skills about business domains or about KDD algorithms and services, their publications, and the previous projects in which they participated.

TeamONTO can be considered as an extension of the previous defined knowledge bases, as its elements can be put in connection with corresponding concepts in KDDONTO (algorithms), UDDI registry (services) and Process Repository (projects), allowing more expressive queries to be performed. For instance, it is possible to search for persons from a given organization, which are experts in a "DecisionTreeAlgorithm", and which have already worked in projects about the "e-Health" domain.

3.2 A collaborative platform for KDD process design

The main goal of the KDDVM platform is to support users in the complex task of collaboratively designing and executing a KDD process. To this end, KDDVM provides a set of support functionalities in the form of web services, which can be accessed both through standard SOAP requests and through some clients. Among the services offered by the platform we recognize basic services, which provide single Data Mining and KDD functionalities allowing to analyze and transform data and to extract knowledge. Moreover, a set of support services is provided for giving both low-level and KDD-specific functionalities to different types of users, which are grouped in four categories on the basis of their aims, namely *deployment*, *discovery*, *composition* and *execution*. Besides these KDD-specific support services, the platform offers also functionalities enabling cooperative work. Figure 2 shows support services together with their mutual interactions.

Oriented to final users, the platform includes *KDDDesigner*, a web-based visual tool that serves as a whiteboard, where the user designs a process accessing to platform's support functionalities in an integrated fashion. Through the KDDDesigner (Fig. 3), users are enabled to collaboratively build a process, by choosing the tools to use, linking them together, discussing about design issues, and executing the process.

3.2.1 Service deployment and publication

Before using the platform, services have to be published into the UDDI repository, from which final users can retrieve the ones useful to solve their problem. For this reason, a set of services and consumer-side applications are available for (1) helping users in writing a KDTML for the description of a tool, (2) transforming a KDD tool, written by a developer in any programming language, into a web service with a standard interface, and generating the corresponding eSAWSDL at the same time, for (3) deploying such a service in an application server, and (4) publishing it into the common registry.

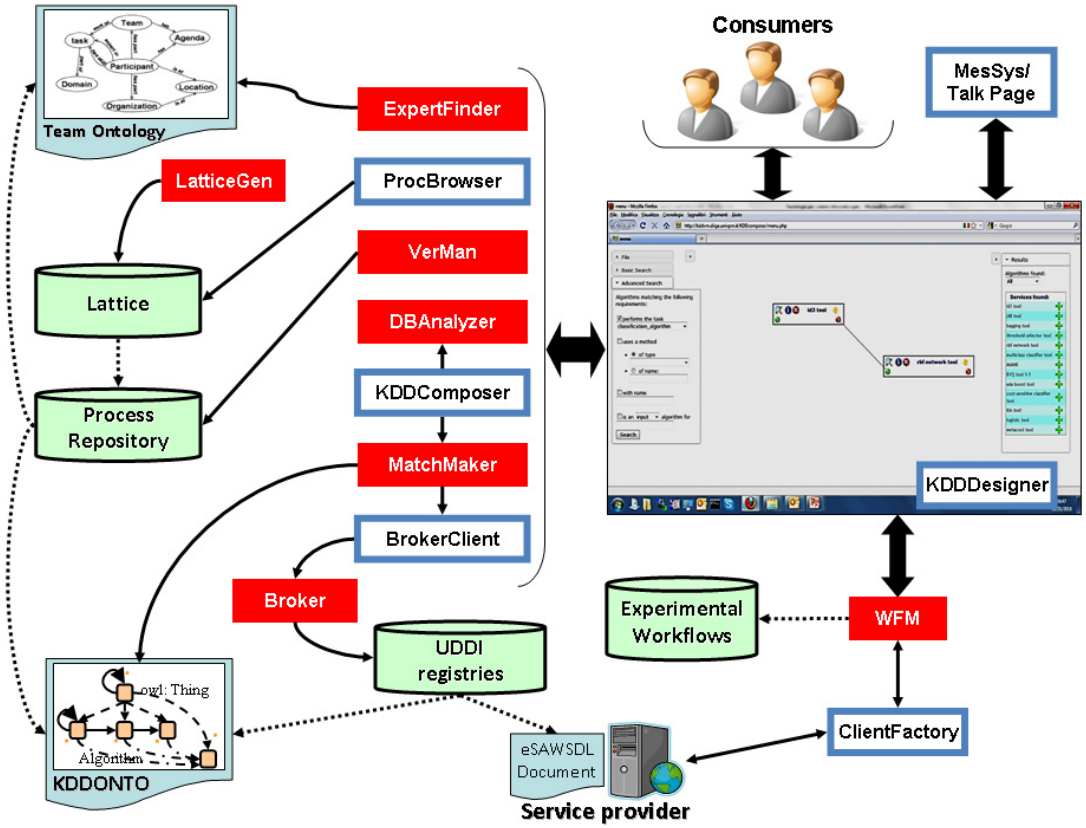

Figure 2. Interactions among services and clients of the platform.

Some other services include back-end functionalities for the management of data transfer among servers and authentication.

3.2.2 Service and process discovery

A fundamental functionality for a distributed platform is the capability of querying repositories for discovering suitable resources, both services and processes.

In order to retrieve a certain service among the others, KDDDesigner interacts with the *Broker* service, which acts as a search engine inside the UDDI registry. Through the Broker, a user can look for specific services which satisfy not only syntactic requirements, like having a certain name or being published by a certain provider, but also semantic requirements by exploiting information available in KDDONTO and in services descriptors.

Given the conceptual separation in distinct logical layers, it is possible to find services implementing specific algorithms: for instance, algorithms useful for a task, or using a certain method, or executable before/after a given algorithm (see also Sec. 3.2.3). Once some algorithms are found in the ontology, the Broker looks inside UDDI registry for services implementing them, thus providing the user with accurate and semantically consistent results. By selecting a service from the result list (right panel in Fig.3), the user can drag it in the design board and begin to compose a new process.

As concerns the discovery of processes, we refer to the Process Repository already described in Section 3.1.2. The service devoted to search inside the process repository is the *ProcBrowser*, which is aimed at two main retrieval tasks:

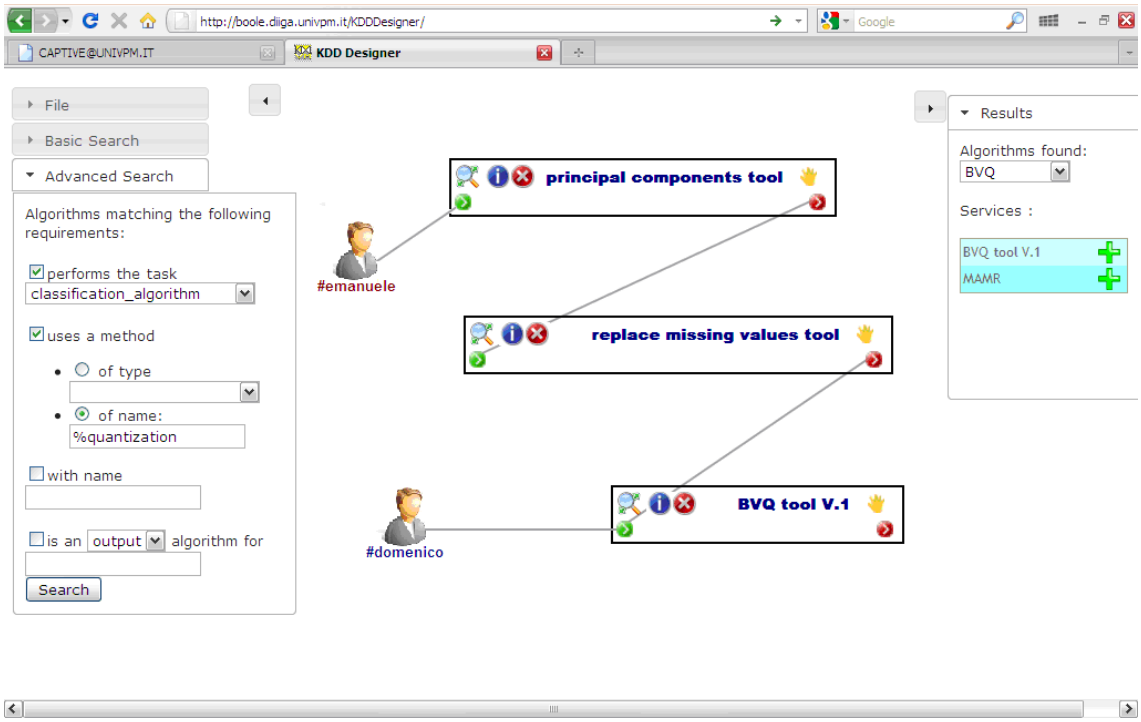

Figure 3. KDDDesigner.

• *process browsing*: the activity of browsing the lattice in order to provide the user with a high-level and expansible representation of all the sub-structures that are defined in the lattice itself. The user can either select a sub-structure in order to visualize all the corresponding processes stored in the ProcRep, or decide to expand the sub-structure in order to reach those at a lower level, which are an extension of the sub-structure at hand;
• *process discovery*: the activity of searching in ProcRep those processes which satisfy a set of requirements. Discovery can be performed according to both standard search (e.g., processes having a given service, involving a specific user or category of users) and structure-based search (e.g., processes including a given sub-process specified by the user). Then, the ProcBrowser performs a sub-graph match between the query and the sub-structures at the top level in the lattice, checking if the smaller of the two contains the other. If the query contains the sub-structure, for each positive match the ProcBrowser evaluates the query against children of the related cluster (sub-structure), and so forth. At the end, the query is evaluated against the union of all processes belonging to clusters at lowest levels that have reported a positive match. The lattice is built by the *LatticeGen* service, which is also aimed to rebuild it when the number of modifications (insert, update and delete) to the repository is such that query performances get worse.

3.2.3 Process composition

A process can be built connecting two processes together in KDDDesigner, linking an output of the first to an input of the second. Nevertheless, some expertise is needed to check the correctness of such connection: in fact, the user is supposed to know that such data are compatible each other, and that such algorithms can be meaningfully connected. For these reasons, when the link between two data is established by the user, the *MatchMaker* service is called, to verify whether the data can be actually connected, and if so it

yields as output the estimated semantic *cost* of their connection. The check is performed by comparing the data being connected, not only syntactically but also semantically, with the aim to understand if they are conceptually equivalent, or if the output is a subtype/subcomponent of the input: such operation is realized through looking for a path, inside KDDONTO, between the input and the output, made of the ontological relations "sameAs", "subClassOf" and "partOf", and by properly weighting each relation. For a match, the evaluation of the cost takes into account also the existence of preconditions on the input datum, which can be completely or partially satisfied by the output, and the computational complexity of algorithms. By selecting a connection between two data, the user is informed about the cost, in order to provide the user with information about the validity of her choice, suggesting which connections are correct and which aren't. Moreover, the estimated total cost for a process, or a sub-process, can be evaluated by selecting the whole process or a part of it, defining in this case the starting/ending services. The MatchMaker is also useful during the service discovery. In fact, by selecting a service (and so its corresponding algorithm), it is possible to search for algorithms that are executable before/after the algorithm at hand, and then to look in the UDDI registry for services implementing them. Then, by exploiting functions for the evaluation of costs, the Broker is able to return a ranked list of useful services. Such functionalities offer an invaluable help during composition, by supporting naïve users to reduce the search space, and thus helping them to find only those services that are compatible with the one at hand.

To bring support to non-expert users, incapable of choosing proper solutions for their goals, the *KDDComposer* is available in the platform, aimed at generating prototype processes in a semi-automatic fashion. Through such an application, a user is only asked to provide a dataset and to specify the task to achieve; by using Artificial Intelligence and planning techniques and by interfacing with the MatchMaker service, KDDComposer yields a list of possible KDD processes. The generated processes are not directly executable because they are formed of algorithms; their aim is to provide suggestions about possible sequences of algorithms that may be exploited to solve the user problem. The composition procedure starts from the user task, then the KDDComposer looks inside KDDONTO for a set of algorithms which are able to achieve it: for each of them a candidate process is created. Iteratively, for each candidate process, the procedure goes on backwards by adding, in the process head, algorithms that can be executed before it, i.e. algorithms with an output interface which is compatible with the input interface of the process head. The iterative step is executed until a valid process is found, that has an input interface capable to elaborate the user dataset.

This work includes also a preliminary study about the exploitation of a formal coordination language Reo [12] for representation of processes and their formal verification through model checking. In the context of KDD processes, this approach is aimed at the individuation, formalization and formal verification of complex coordination patterns that typically appear in distributed experimental processes for the extraction of potentially useful knowledge from data, in order to support their successive reuse during composition.

3.2.4 Process execution

A process schema is a workflow of services, which can be executed by means of an engine that is able to interpret standard workflow languages (e.g. XPDL, XScufl, BPEL). Although processes in KDDDesigner are currently represented in an internal XML format, users can export them into standard workflow languages and execute them by means of external engines. Alternatively, their execution can be managed by the *WorkFlow Manager* (WFM). This service extends existing engines in order to interpret semantic annotation and specific

KDDVM information. At present we have developed a prototype WFM that interprets processes written in XPDL. Since the KDDDesigner lets users create a process without completely wiring all the services' interfaces, when the execution of a service needs the human intervention (e.g., for parameters setting), the WFM calls the *ClientFactory* tool. Such a tool reads the eSAWSDL and on-the-fly generates a graphical interface showing to the user, which has been entrusted with the execution of the service, every service parameter plus other information taken from the descriptor. The use of ClientFactory during the process execution allows users to design processes at different detail levels. As a matter of fact, a team of users can either specify all the details needed to execute the process, or fix just the structure of the process, outsourcing the work of parameter tuning to other users with more specific skills and expertise.

3.2.5 Collaborative functionalities

KDDVM platform provides tools to support team building, versioning and communication. By exploiting the knowledge coded into TeamONTO it is possible to plan the process assigning the management of a specific web service to a certain user, taking into account its competencies. To this aim, an *Expert Finder* service helps users in searching for experts satisfying a set of requirements. For instance, by exploiting inference on both KDDONTO and the team ontology, it is possible to look for users which are expert in a specific service, in a given class of algorithms, in algorithms useful for a certain task, or which have participated in similar projects and are available in certain periods.

Collaboration within a project takes place only if the team members are able to interact each other, in order to align their actions towards the shared goal: for this reason, the system should allow to discuss design choices and to jointly edit the process. Given that experts usually cooperate in KDD projects according to their work schedules, a common type of communication is asynchronous. Following this approach, many users can collaboratively work on a same KDD project by opening the same process in edit mode, saving the process at the end. In order to keep track of how the process is evolving, the platform provides *ProcVer*, a service for process versioning management. KDD projects are fully described, inside the Process Repository, through a version tree, in which each node is linked to an item in the Repository, i.e. a different version of the same process. In such a way the history of versions is available, thus allowing to be aware of possible alternative branches, as well as to roll-back to previous versions.

ProcVer performs the serialization of a process developed by the KDDDesigner into the XML format, its storage into the repository and the update of the version tree of the KDD project at hand. Moreover, it also supports retrieval functionalities, such as: *browsing* the versions of a given project, traversing the tree and providing the user with metadata about each version, and *discovery*, w.r.t. a given project, of those versions that satisfy some user requirements.

Besides the comments that can be attached to a process version, a more direct form of communication among authors is achieved through a Talk Page, aimed at providing space for editors to discuss changes for a specific process.

4 Conclusion

In this work we propose a novel framework and platform for supporting the design of KDD processes in distributed and collaborative environments. The fundamental features

of our approach are: (a) the capability to manage various kinds of heterogeneities, like different implementations with different characteristics of the same algorithm, (b) the possibility to easily add functionalities to the system: for instance, the integration of new data analysis tools is not limited to applications complying with a certain standard; legacy tools are made talk together by wrapping every tool with a standard web service interface, and describing the tool's capabilities and input/output interface by means of ontological concepts; (c) a community-centered attitude, with functionalities for both resource production and consumption, facilitating end-users with different skills as well as resource providers with different technical and domain specific capabilities.

What makes KDDVM original with respect to other proposals is the systematic use of semantic information, a loosely coupled and layered architecture, a cooperative approach and its flexibility. While most solutions focus only on Data Mining or on local KDD support systems, our proposal is more general and natively conceived for an open, distributed and collaborative environment.

References

[1] U.M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, *From data mining to knowledge discovery: an overview*, pp. 1–34, American Association for Artificial Intelligence, Menlo Park, CA, USA, 1996.

[2] C. Shearer, *The crisp-dm model: The new blueprint for data mining*, Journal of Data Warehousing, vol. 5, no. 4, 2000.

[3] T. Hey S. Tansley, and K. Tolle, Eds., *The Fourth Paradigm: Data-Intensive Scientific Discovery*, Microsoft Research, Redmond, Washington, 2009.

[4] A. Bernstein and M. Dänzer, *The next system: Towards true dynamic adaptations of semantic web service compositions*, in Proceedings of the 4th European conference on The Semantic Web: Research and Applications, Berlin, Heidelberg, 2007, ESWC '07, pp. 739–748, Springer-Verlag.

[5] M. Žáková, P. Kremen, F. Železný, and N. Lavrac, *Automating Knowledge Discovery Workflow Composition Through Ontology-Based Planning*, IEEE T. Automation Science and Engineering, vol. 8, no. 2, pp. 253–264, 2011.

[6] M. Cannataro and D. Talia, *The knowledge grid*, Commun. ACM, vol.46, pp. 89–93, January 2003.

[7] G. Kickinger, J. Hofer, P. Brezany, and A.M. Tjoa, *Grid knowledge discovery processes and an architecture for their composition*, in Proc. of IASTED Conference 2004, Innsbruck, Austria, February 17-19 2004.

[8] D. De Roure, C. Goble, and R. Stevens, *The Design and Realisation of the Virtual Research Environment for Social Sharing of Workflows*, Future Generation Computer Systems, vol. 25, no. 5, pp. 561–567, 2009.

[9] C. Diamantini, D. Potena, and M. Panti, *Developing an Open Knowledge Discovery Support System for a Network Environment*, in Proc. of the 2005 Int. Symp. on Collaborative Technologies and Systems, Saint Louis, Missouri, USA, pp. 274–281, 2005.

[10] *Semantic annotations for wsdl and xml schema*, W3C Recommendation 28 August 2007, http://www.w3.org/TR/sawsdl/.

[11] I. Jonyer, D.J. Cook, and L.B. Holder, *Graph-based hierarchical conceptual clustering*, J. Mach. Learn. Res., vol. 2, pp. 19–43, March 2002.

[12] F. Arbab, *Reo: a channel-based coordination model for component composition*, Mathematical. Structures in Comp. Sci., vol. 14, no. 3, pp. 329–366, 2004.