

# Data Mining Workflow Templates for Intelligent Discovery Assistance and Auto-Experimentation

Jörg-Uwe Kietz<sup>1</sup>, Floarea Serban<sup>1</sup>, Abraham Bernstein<sup>1</sup>, and Simon Fischer<sup>2</sup>

<sup>1</sup> University of Zurich, Department of Informatics,  
Dynamic and Distributed Information Systems Group,  
Binzmühlestrasse 14, CH-8050 Zurich, Switzerland  
{kietz|serban|bernstein}@ifi.uzh.ch

<sup>2</sup> Rapid-I GmbH, Stockumer Str. 475, 44227 Dortmund, Germany  
fischer@rapid-i.com

**Abstract.** Knowledge Discovery in Databases (KDD) has grown a lot during the last years. But providing user support for constructing workflows is still problematic. The large numbers of operators available in current KDD systems makes it difficult for a user to successfully solve her task. Also, workflows can easily reach a huge number of operators (hundreds) and parts of the workflows are applied several times. Therefore, it becomes hard for the user to construct them manually. In addition, workflows are not checked for correctness before execution. Hence, it frequently happens that the execution of the workflow stops with an error after several hours runtime.

In this paper<sup>3</sup> we present a solution to these problems. We introduce a knowledge-based representation of KDD workflows as a basis for cooperative-interactive planning. Moreover, we discuss workflow templates, i.e. abstract workflows that can mix executable operators and tasks to be refined later into sub-workflows. This new representation helps users to structure and handle workflows, as it constrains the number of operators that need to be considered. Finally, workflows can be grouped in templates which foster re-use further simplifying KDD workflow construction.

## 1 Introduction

One of the challenges of Knowledge Discovery in Database (KDD) is assisting the users in creating and executing KDD workflows. Existing KDD systems such as the commercial Clementine and Enterprise Miner or the open-source Weka, MiningMart, KNIME and RapidMiner support the user with nice graphical user interfaces, where operators can be dropped as nodes onto the working pane and

---

<sup>3</sup> This paper reports on work in progress. Refer to <http://www.e-lico.eu/eProPlan> to see the current state of the Data Mining ontology for WorkFlow planning (DMWF), the IDA-API, and the eProPlan Protege plug-ins we built to model the DMWF. The RapidMiner IDA-wizard will be part of a future release of RapidMiner check <http://www.rapidminer.com/> for it.

the data-flow be specified by connecting the operator-nodes. This works very well as long as neither the workflow becomes too complicated and large nor the number of operators becomes too large.

The *number of operators* in such systems, however, has been growing fast. All of them contain over 100 operators and RapidMiner, which includes Weka, even over 600. It can be expected that with the incorporation of text-, image-, and multimedia-mining as well as the transition from closed systems with a fixed set of operators to open systems that can also use Web services as operator (which is especially interesting for domain specific data access and transformations) will further accelerate the rate of growth resulting in total confusion for most users.

Not only the number of operators, but also the *size of the workflows* is growing. Today's workflows can easily contain hundreds of operators. Parts of the workflows are applied several times ( e.g. the preprocessing sub-workflow has to be applied on training, testing, and application data) implying that the users either need to copy/paste or even design a new sub-workflow<sup>4</sup> several times. None of the systems maintain this “copy”-relationship, it is left to the user to maintain the relationship in the light of changes.

Another weak point is that workflows are not checked for *correctness* before execution: it frequently happens that the execution of the workflow stops with an error after several hours runtime because of small syntactic incompatibilities between an operator and the data it should be applied on.

To address these problems several authors [1, 12, 4, 13] propose the use of planning techniques to automatically build such workflows. However all these approaches are limited in several ways. First, they only model only a very small set of operations and were only demonstrated to work on very short workflows (less than 10 operators). Second, none of them models operations that work on individual columns of a data set, they only model operations that process all columns of a data set equally together. Lastly, the approaches cannot scale to large amounts of operators and large workflows: their used planning approaches will necessarily get lost in the too large space of “correct” (but nevertheless most often unwanted) solutions. In [6] we reused the idea to use the hierarchical task decomposition (from the manual support system CITRUS [11]) and knowledge available in Data Mining (e.g. CRISP-DM) for hierarchical task network (HTN) planning [9]. This significantly reduces the number of generated unwanted correct workflows. Unfortunately, given its capturing of generic data mining knowledge only, it still does not capture the most important knowledge a

---

<sup>4</sup> Several operators must be exchanged and cannot be just reapplied. Consider for example training data (with labels) and application data (without labels). Label-directed operations like feature selection or discretization by entropy used on the training data cannot work on the application data. But even if there is a label like on separate test data, redoing feature selection/discretization may result in selecting/building different features/bins. But to apply and test the model exactly the same features/bins have to be selected/build.

data mining engineer uses to judge workflows and models useful: understanding of the meaning of the data<sup>5</sup>.

Formalizing the meaning of the data requires a large amount of domain knowledge. Eliciting all the possible needed background information about the data from the user would probably be more demanding for her than designing useful workflows manually. Therefore, the completely automatic planning of useful workflows is not feasible. The approach of enumerating all correct workflows and then let the user choose the useful one(s) will likely fail due to the large number of correct workflows (infinite, without a limit on the number of operations in the workflow). Only *cooperative-interactive planning* of workflows seems to be feasible. In this scenario the planner ensures the correctness of the state of planning and can propose a small number of possible intermediate refinements of the current plan to the user. The user can use her knowledge about the data to choose useful refinements, can make manual additions/corrections, and use the planner again for tasks that can be routinely solved without knowledge about the data. Furthermore, the planner can be used to generate all correct sub-workflows to optimize the workflow by experimentation.

In this paper we present a knowledge-based representation of KDD workflows understandable to both planner and user as the foundation for cooperative-interactive planning. To be able to represent the intermediate states of planning, we generalize this to “workflow templates”, i.e. abstract workflows that can mix executable operators and tasks to be refined later into sub-workflows (or sub-workflow-templates). Our workflows follow the structure of a Data Mining Ontology for Workflows (DMWF). It has a hierarchical structure consisting of a task/method decompositions into tasks, methods or operators. Therefore, workflows can be grouped based on the structure decomposition and can be simplified by using abstract nodes. This new representation helps the users since akin to structured programming as the elements (now both operators, tasks, and methods) of a workflow actively under consideration is reduced significantly. Furthermore, this approach allows to group certain groups of operators as templates to be reused later. All this simplifies and improves the design of a DM workflow, reducing the time needed to construct workflows, and decreases the workflow’s size.

This paper is organized as follows: Section 2 describes workflows and their representation as well as workflow template, Section 3 shows the advantages of workflow templates, Section 4 presents the current state and future steps and finally Section 5 concludes our paper.

---

<sup>5</sup> Consider a binary attribute “address invalid”: just by looking at the data is almost impossible to infer that does not make sense to send advertisement to people with this flag set at the moment. In fact they may have responded to previous advertisements very well.

## 2 Data Mining Workflow

Data Mining (DM) workflows generally represent a set of DM operators, which are executed and applied on data or models. In most of the DM tools users are only working with operators and setting their parameters (values). Data is implicit, hidden in the connectors, the user provides the data and applies the operators, but after each step new data is produced. In our approach we distinguish between all the components of the DM workflow, operators, data, parameters. To enable that system and user cooperatively design the workflows, we developed a formalization of the DM workflows in terms of an ontology.

To be able to define a DM workflow we first need to describe the DMWF ontology since workflows are stored and represented in DMWF format. This ontology encodes rules from the KDD domain on how to solve DM tasks, as for example the CRISP-DM [2] steps in the form of concepts and relations (Tbox – terminology). The DMWF has several classes that contribute in describing the DM world, `IOObjects`, `MetaData`, `Operators`, `Goals`, `Tasks` and `Methods`. The most important ones are shown in Table 1.

Class	Description	Examples
<code>IOObject</code>	Input and output used by operators	Data, Model, Report
<code>MetaData</code>	Characteristics of the <code>IOObjects</code>	<code>Attribute</code> , <code>AttributeType</code> , <code>DataColumn</code> , <code>DataFormat</code>
<code>Operator</code>	DM operators	<code>DataTableProcessing</code> , <code>ModelProcessing</code> , <code>Modeling</code> , <code>MethodEvaluation</code>
<code>Goal</code>	A DM goal that the user could solve	<code>DescriptiveModelling</code> , <code>PatternDiscovery</code> , <code>PredictiveModelling</code> , <code>RetrievalByContent</code>
<code>Task</code>	A task is used to achieve a goal	<code>CleanMV</code> , <code>CategoricalToScalar</code> , <code>DiscretizeAll</code> , <code>PredictTarget</code>
<code>Method</code>	A method is used to solve a task	<code>CategoricalToScalarRecursive</code> , <code>CleanMVRecursive</code> , <code>DiscretizeAllRecursive</code> , <code>DoPrediction</code>

Table 1: Main classes from the DMWF ontology

Properties <sup>6</sup>	Domain	Range	Description
<code>uses</code> – <code>usesData</code> – <code>usesModel</code>	<code>Operator</code>	<code>IOObject</code>	defines input for an operator
<code>produces</code> – <code>producesData</code> – <code>producesModel</code>	<code>Operator</code>	<code>IOObject</code>	defines output for an operator
<code>parameter</code>	<code>Operator</code>	<code>MetaData</code>	defines other parameters for operators
<code>simpleParameter</code>	<code>Operator</code>	data type	
<code>solvedBy</code>	<code>Task</code>	<code>Method</code>	A task is solved by a method
<code>worksOn</code> – <code>inputData</code> – <code>outputData</code>	<code>TaskMethod</code>	<code>IOObject</code>	The <code>IOObject</code> elements the <code>Task</code> or <code>Method</code> works on
<code>worksWith</code>	<code>TaskMethod</code>	<code>MetaData</code>	The <code>MetaData</code> elements the <code>Task</code> or <code>Method</code> worksWith
<code>decomposedTo</code>	<code>Method</code>	<code>Operator/Task</code>	A <code>Method</code> is decomposed into a set of steps

Table 2: Main roles from the DMWF ontology

The concepts from the DMWF ontology are connected through roles or properties as shown in Table 2. The parameters of operators as well as some basic characteristics of data are values (integer, double, string, etc.) in terms of data

<sup>6</sup> Later on we use `usesProp`, `producesProp`, `simpleParamProp`, etc. to denote the subproperties of `uses`, `produces`, `simpleParameter`, etc. .

properties, e.g. number of records for each data table, number of missing values for each column, mean value and standard deviation for each scalar column, number of different values for nominal columns, etc. Having them modeled in the ontology enables the planner to use them in planning.

## 2.1 What is a workflow?

In our approach a workflow constitutes an instantiation of the DM concepts; more precisely is a set of ontological individuals (Abox - assertions). It is mainly composed from several basic operators, which can be executed or applied with the given parameters. The workflow follows the structure illustrated in Fig. 1. A workflow consists of several operator applications, instances of operators as well as their inputs and outputs – instances of `IObject`, simple parameters (values which can have different data types like integer, string, etc.), or parameters – instances of `MetaData`. The flow itself is rather implicit, it's represented by shared `IObjects` used and produced by `Operators`. The reasoner can ensure, that every `IObject` has only 1 producer and that every `IObject` is either given as input to the workflow or produced before it can be used.

```
Operator[usesProp1 {1,1}⇒ IObject, ..., usesPropn {1,1}⇒IObject,
producesProp1 {1,1}⇒ IObject, ..., producesPropn {1,1}⇒IObject,
parameterProp1 {1,1}⇒ MetaData, ..., parameterPropn {1,1}⇒ MetaData,
simpleParamProp1 {1,1}⇒ dataType, ..., simpleParamPropn {1,1}⇒ dataType].
```

Fig. 1: Tbox for operator applications and workflows

Fig. 2 illustrates an example of a real workflow. It is not a linear sequence since models are shared between subprocesses, so the workflow produced is a DAG (Direct Acyclic Graph). The workflow consists of two subprocesses: the training and the testing which share the models. We have a set of basic operator individuals (`FillMissingValues1`, `DiscretizeAll1`, etc.) which use individuals of `IObject` (`TrainingData`, `TestData`, `DataTable1`, etc.) as input and produce individuals of `IObject` (`PreprocessingModel1`, `Model1`, etc.) as output. The example does not display the parameters and simple parameters of operators but each operators could have several such parameters.

## 2.2 Workflow templates

Very often the DM workflows have a large number of operators (hundreds), even more some sequences of operators may repeat and be executed several times in the same workflow. This becomes a real problem since the users need to construct and maintain the workflows manually. To overcome this problem we introduce the notion of workflow templates.

When the planner generates a workflow it follows a set of task/method decomposition rules encoded in the DMWF ontology. Every task has a set of methods able to solve it. The task solved by a method is called the head of the method. Each method is decomposed into a sequence of steps which can be

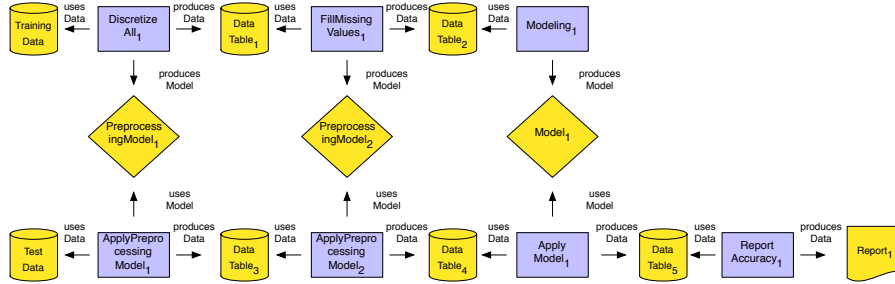


Fig. 2: A basic workflow example

either tasks or operators as shown in the specification in Fig. 3. The matching between the current and the next step is done based on operators' conditions and effects as well as methods' conditions and contributions as described in [6]. Such a set of task/method decompositions works similar to a context-free grammar: tasks are the non-terminal symbols of the grammar, operators are the terminal-symbols (or alphabet) and methods for a task are the grammar-rule that specify how a non-terminal can be replaced by a sequence of (simpler) tasks and operators. In this analogy the workflows are words of the language specified by the task/method decomposition grammar. To be able to generate not only operator sequences, but also operator DAGs<sup>8</sup>, it additionally contains a specification for passing parameter constraints between methods, tasks and operators<sup>9</sup>. In the decomposition process the properties of the method's (the task) or one of the steps can be bound to the same variable as the properties of other steps.

```

TaskMethod[worksOnProp1 ⇒ IOObject, ..., worksOnPropn ⇒ IOObject,
           worksWithProp1 ⇒ MetaData, ..., worksWithPropn ⇒ MetaData]
{Task, Method} :: TaskMethod.
Task[solvedBy ⇒ Method].
{step1, ..., stepn} :: decomposedTo.
Method[step1 ⇒ {Operator|Task}, ..., stepn ⇒ {Operator|Task}].
Method.{head|stepi}.prop = Method.{head|stepj}.prop
prop := workOnProp | workWithProp | usesProp | producesProp
       | parameterProp | simpleParamProp

```

Fig. 3: TBox for task/method decomposition and parameter passing constraints

A workflow template represents the upper (abstract) nodes from the generated decomposition, which in fact are either tasks, methods or abstract operators. If we look at the example in Fig. 2 none of the nodes are basic operators. Indeed, they are all tasks as place-holders for several possible basic operators.

<sup>8</sup> The planning process is still sequential, but the resulting structure may have a non-linear flow of objects.

<sup>9</sup> Giving it the expressive power of a first-order logic horn-clause grammar.

For example, `DiscretizeAll` has different discretization methods as described in Section 3, therefore `DiscretizeAll` represents a task which can be solved by the `DiscretizeAllAtOnce` method. The method can have several steps, e.g. the first step is an abstract operator `RM.DiscretizeAll`, which subsequently has several basic operators like `RM.Discretize All by Size`, `RM.Discretize All by Frequency`.

The workflows are produced by an HTN (Hierarchical Task Network) planner [9] based on the DMWF ontology as background knowledge (domain) and on the goal and data description (problem). In fact, a workflow is equivalent to a generated plan.

The planner generates only valid workflows since it checks the preconditions of every operator present in the workflow, also operator's effects are the preconditions of the next operator in the workflow. In most of the existing DM tools the user can design a workflow, start executing it, and after some time discover that in fact some operator was applied on data with missing values or on nominals whilst, in fact, it can handle only missing value free data and scalars. This is annoying and time consuming. Our approach can avoid this problems since we can encode conditions and effects of our operators, which check for such problems. An operator is applicable only when its preconditions are satisfied, therefore the generated workflows are semantically correct.

### 3 Workflow Templates for auto-experimentation

To illustrate the usefulness of our approach, consider the following common scenario. Given a data table containing numerical algorithms, a modelling algorithm should be applied that is not capable of processing numerical values, e.g., a simple decision tree induction algorithm. In order to still utilize this algorithm, attributes must first be discretized. To discretize a numerical attribute, its range of possible numerical values is partitioned, and each numerical value is replaced by the generated name of the partition it falls into. The data miner has multiple options to compute this partition. E. g., RapidMiner [8] contains five different algorithms to discretize data:

- **Discretize by Binning.** The numerical values are divided into  $k$  ranges of equal size. The resulting bins can be arbitrarily unbalanced.
- **Discretize by Frequency.** The numerical values are inserted into  $k$  bins divided at thresholds computed such that an equal number of examples is assigned to each bin. The ranges of the resulting bins may be arbitrarily unbalanced.
- **Discretize by Entropy.** Bin boundaries are chosen as to minimize the entropy in the induced partitions. The entropy is computed with respect to the label attribute.
- **Discretize by Size.** Here, the user specifies the number of examples that should be assigned to each bin. Consequently, the number of bins will vary.
- **Discretize by User Specification.** Here, the user can manually specify the boundaries of the partition. This is typically only useful if meaningful boundaries are implied by the application domain.

Each of these operators has its advantages and disadvantages. However, there is no universal rule of thumb as to which of the options should be used depending on the characteristics or domain of the data. Still, some of the options can be excluded in some cases. E.g., the entropy can only be computed if a nominal label exists. There are also soft rules. E.g., it is not advisable to choose any discretization algorithm with fixed partition boundaries if the attribute values are skewed. Then, one might end up with bins that contain only very few examples.

Though no such rule of thumb exists, it is also evident that the choice of discretization operator can have a huge impact on the result of the data mining process. To support this statement, we have performed experiments on some standard data sets. We have executed all combinations of the five discretization operators Discretize by Binning with two and four bins, Discretize by Frequency with two and four bins, and Discretize by Entropy on the four numerical attributes of the well-known UCI data set Iris. Following the discretization, a decision tree was generated and evaluated using a ten-fold cross validation<sup>10</sup>. We can observe that the resulting accuracy varies significantly, between 64.0% and 94.7% (see Table 3). Notably, the best performance is not achieved by selecting a single method for all attributes, but by choosing a particular combination. This shows that finding the right combination can actually be worth the effort.

Dataset	#numerical attr.	# total attr.	min. accuracy	max. accuracy
Iris	4	4	64.0%	94.7%
Adult	14	5	82.6%	86.3%

Table 3: The table shows that optimizing the discretization method can be a huge gain for some tables, whereas it is negligible for others.

Consider the number of different combinations possible for  $k$  discretization operators and  $m$  numeric attributes. This makes up for a total of  $k^m$  combinations. If we want to try  $i$  different values for the number of bins, we even have  $(k \cdot i)^m$  different combinations. In the case of our above example, this makes for a total of 1 296 combinations. Although knowing that the choice of discretization operator can make a huge difference, most data miners will not be willing to perform such a huge amount of experiments.

In principle, it is possible to execute all combinations in an automated fashion using standard RapidMiner operators. However, such a process must be custom-made for the data set at hand. Furthermore, discretization is only one out of numerous typical preprocessing steps. If we take into consideration other steps like the replacement of missing values, normalization, etc., the complexity of such a task grows beyond any reasonable border.

This is, where workflow templates come into play. In a workflow template, it is merely specified that at some point in the workflow all attributes must be discretized, missing values be replaced or imputed, or a similar goal be achieved. The planner can then create a collection of plans satisfying these constraints. Clearly, simply enumerating all plans only helps if computational power exists,

<sup>10</sup> The process used to generate these results is available on the myExperiment platform [3]: <http://www.myexperiment.org/workflows/1344>

to try all possible combinations. Where this is not possible, the number of plans must be reduced. Several options exist:

- Where fixed rules of thumb like the two rules mentioned above exist, this is expressed in the ontological description of the operators. Thus, the search space can be reduced, and less promising plans can be excluded from the resulting collection of plans.
- The search space can be restricted by allowing only a subset of possible combinations. E.g., we can force the planner to apply the same discretization operator to all attributes (but still allow any combination with other preprocessing steps).
- The ontology is enriched by resource consumption annotations describing the projected execution time and memory consumption of the individual operators. This can be used to rank the retrieved plans.
- Where none of the above rules exist, meta mining from systematic experimentation can help to rank plans and test their execution in a sensible order. This work is ongoing work within the e-Lico project.
- Optimizing the discretization step does not necessarily yield such a huge gain as presented above for all data sets. E.g., we have executed a similar optimization as the one presented above for the numerical attributes of the Adult data set. Here, the accuracy only varies between 82.6% and 86.3% (see Table 3). In hindsight, the reason for this is clear: Whereas all of the attributes of the Iris data set are numerical, only six out of 14 attributes of the Adult dataset are. Hence, the expected gain for Iris is much larger. A clever planner can spot this fact, removing possible plans where no large gain can be expected. Findings like these can also be supported by meta mining.

All these approaches help the data miner to optimize steps where this is promising and generating and executing the necessary processes to be evaluated.

## 4 Current state

The current state and some of the future development plans of our project are shown in Fig.4. The system consist of a modeling environment called eProPlan (e-Lico Protege-based Planner) in which the ontology that defines the behavior of the Intelligent Discovery Assistant (IDA) is modeled. eProPlan consist of a set of Protégé4-plugins [7], that adds the modeling of the operators with their conditions and effects and the task-method decomposition to the base-ontology modeling. It allows to analyze workflow inputs and to set up the goals to be reached in the workflow. It also adds a reasoner-interface to our reasoner/planner, such that the applicability of operators to IO-Objects can be tested (i.e. the correct modeling of the condition of an operator), a single operator can be applied with applicable parameter setting (i.e. the correct modeling of the effect of an operator can be tested), and also the planner can be asked to

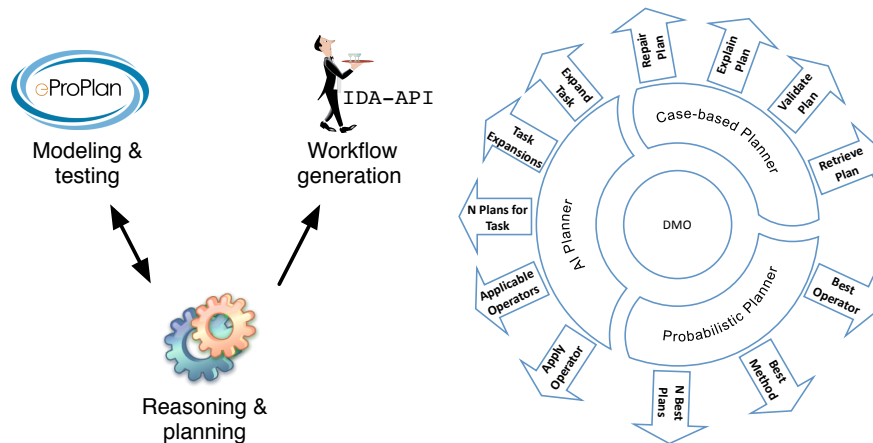


Fig. 4: (a) eProPlan architecture (b) The services of the planner

generate a whole plan for a specified task (i.e. the task-method decomposition can be tested).

Using eProPlan we modeled the DMWF ontology which currently consists of 64 Modeling (Data Mining) Operators, including supervised learning, clustering, and association rules generation of which 53 are leaves i.e. executable Rapid-Miner Operators. We have also 78 executable preprocessing Operators from RapidMiner and 30 abstract Groups categorizing them. We also have 5 Reporting (e.g. a data audit, ROC-curve), 5 Model evaluation (e.g. cross-validation) and Model application operators from RapidMiner. The domain model which describes the IO-Objects of operators (i.e. data tables, models, reports, text collections, image collections) consist of 43 classes. With that the DMWF is by far the largest collection of real operators modeled for any planner-IDA in the related work. A main innovation of our domain model over all previous planner-based IDAs is that we did not stop with the IO-Objects, but modeled their parts as well, i.e. we modeled the attributes and the relevant properties a data table consists of as well. With this model we are able to capture the conditions and effects of all these operators not only on the table-level but on the column-level as well. This important improvement was illustrated on the example of discretization in the last section. On the Task-method decomposition side we modeled a CRISP-DM top-level HTN and its behavior can be modified by currently 15 (sub-) Goals that are used as further hints for the HTN planner. We also have several bottom-level tasks as the DiscretizeAll described in the last section, e.g. for Missing Value imputation and Normalization.

To access our planner IDA in data mining environment we are currently developing an IDA-API (Intelligent Data Assistant - Application Programming Interface). The first version of the API will offer the "AI-Planner" services in Fig.4(b), but we are also working to extend our planner with the case-based planner services shown there and our partner is working to integrate the probabilistic planner services [5]. The integration of the API into RapidMiner as a wizard is displayed in Fig. 5 and it will be integrated into Taverna [10] as well.

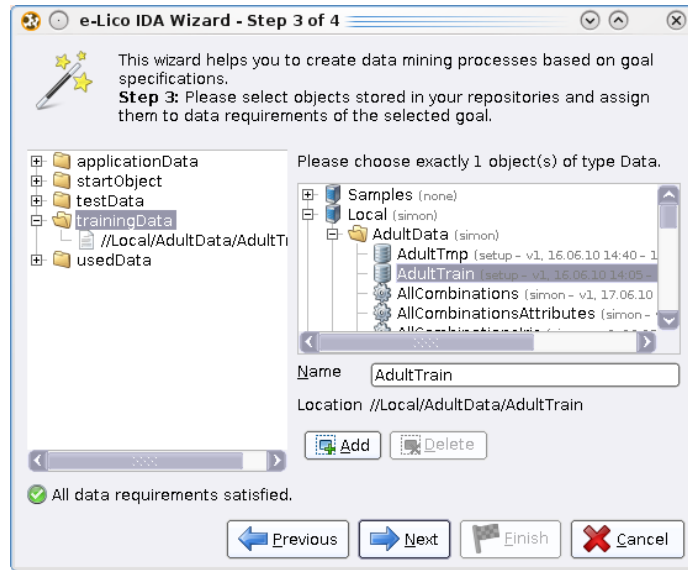


Fig. 5: A screenshot of the IDA planner integrated as a Wizard into RapidMiner.

## 5 Conclusion and future work

In this paper we introduced a knowledge-based representation of KDD workflows as a basis for cooperative-interactive workflow planning. Based on that we presented the main contribution of this paper: the definition of workflow templates, i.e. abstract workflows that can mix executable operators and tasks to be refined later into sub-workflows. We argued that these workflow templates serve very well as the common workspace for user and system to cooperatively design workflows. Due to their hierarchical Task structure they can help to make large workflows neat. We experimentally showed on the example of discretization that they can help to optimize the performance of workflows by auto-experimentation. Future work will try to meta-learn from these workflow-optimization experiments, such that a probabilistic extension of the planner can rank the plans based on their expected success. We argued that knowledge about the content of the data (which cannot be extracted from the data) has a strong influence on the design of useful workflows. Therefore, previously designed workflows for similar data and goals likely contain an implicit encoding of this knowledge. This means an extension to case-based planning is a promising direction for future work as well. We expect that the workflow templates will help us in case adaptation as well, because they tell us what a sub-workflow wants to achieve on the data.

**Acknowledgements:** This work is supported by the European Community 7<sup>th</sup> framework ICT-2007.4.4 (No 231519) “e-Lico: An e-Laboratory for Interdisciplinary Collaborative Research in Data Mining and Data-Intensive Science”.

## References

1. A. Bernstein, F. Provost, and S. Hill. Towards Intelligent Assistance for a Data Mining Process: An Ontology-based Approach for Cost-sensitive Classification. *IEEE Transactions on Knowledge and Data Engineering*, 17(4):503–518, April 2005.
2. P. Chapman, J. Clinton, R. Kerber, T. Khabaza, T. Reinartz, C. Shearer, and R. Wirth. Crisp-dm 1.0: Step-by-step data mining guide. Technical report, The CRISP-DM Consortium, 2000.
3. D. De Roure, C. Goble, and R. Stevens. The design and realisation of the myexperiment virtual research environment for social sharing of workflows. In *Future Generation Computer Systems 25*, pages 561–567, 2009.
4. C. Diamantini, D. Potena, and E. Storti. KDDONTO: An Ontology for Discovery and Composition of KDD Algorithms. In *Service-oriented Knowledge Discovery (SoKD-09) Workshop at ECML/PKDD09*, 2009.
5. M. Hilario, A. Kalousis, P. Nguyen, and A. Woznica. A data mining ontology for algorithm selection and meta-learning. In *Service-oriented Knowledge Discovery (SoKD-09) Workshop at ECML/PKDD09*, 2009.
6. J.-U. Kietz, F. Serban, A. Bernstein, and S. Fischer. Towards cooperative planning of data mining workflows. In *Service-oriented Knowledge Discovery (SoKD-09) Workshop at ECML/PKDD09*, 2009.
7. H. Knublauch, R. Fergerson, N. Noy, and M. Musen. The Protégé OWL plugin: An open development environment for semantic web applications. *Lecture notes in computer science*, pages 229–243, 2004.
8. I. Mierswa, M. Wurst, R. Klinkenberg, M. Scholz, and T. Euler. Yale: Rapid prototyping for complex data mining tasks. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 935–940. ACM, 2006.
9. D. Nau, T.-C. Au, O. Ilghami, U. Kuter, W. Murdock, D. Wu, and F. Yaman. SHOP2: An HTN planning system. *JAIR*, 20:379–404, 2003.
10. T. Oim, M. Addis, J. Ferris, D. Marvin, M. Greenwood, T. Carver, M. Pocock, A. Wipat, and P. Li. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 2004.
11. R. Wirth, C. Shearer, U. Grimmer, T. P. Reinartz, J. Schlösser, C. Breitner, R. Engels, and G. Lindner. Towards process-oriented tool support for knowledge discovery in databases. In *PKDD '97: Proceedings of the First European Symposium on Principles of Data Mining and Knowledge Discovery*, pages 243–253, London, UK, 1997. Springer-Verlag.
12. M. Žáková, P. Křemen, F. Železný, and N. Lavrač. Planning to learn with a knowledge discovery ontology. In *Planning to Learn Workshop (PlanLearn 2008) at ICML 2008*, 2008.
13. M. Žáková, V. Podpečan, F. Železný, and N. Lavrač. Advancing data mining workflow construction: A framework and cases using the orange toolkit. In *Service-oriented Knowledge Discovery (SoKD-09) Workshop at ECML/PKDD09*, 2009.