To Appear in: ECML/PKDD09 Workshop on Third Generation Data Mining: Towards Service-oriented Knowledge Discovery (SoKD-09)

Towards Cooperative Planning of Data Mining Workflows

Jörg-Uwe Kietz[1], Floarea Serban[1], Abraham Bernstein[1], and Simon Fischer[2]

[1] University of Zurich, Department of Informatics,
Dynamic and Distributed Information Systems Group,
Binzmühlestrasse 14, CH-8050 Zurich, Switzerland
{kietz|serban|bernstein}@ifi.uzh.ch
[2] Rapid-I GmbH, Stockumer Str. 475, 44227 Dortmund, Germany
fischer@rapid-i.com

Abstract. A major challenge for third generation data mining and knowledge discovery systems is the integration of different data mining tools and services for data understanding, data integration, data preprocessing, data mining, evaluation and deployment, which are distributed across the network of computer systems. In this paper we outline how an intelligent assistant that is intended to support end-users in the difficult and time consuming task of designing KDD-Workflows out of these distributed services can be built. The assistant should support the user in checking the correctness of workflows, understanding the goals behind given workflows, enumeration of AI planner generated workflow completions, storage, retrieval, adaptation and repair of previous workflows. It should also be an open easy extendable system. This is reached by basing the system on a data mining ontology (DMO) in which all the services (operators) together with their in-/output, pre-/postconditions are described. This description is compatible with OWL-S and new operators can be added importing their OWL-S specification and classifying it into the operator ontology.

1 Introduction

In the early days of data mining the biggest challenge for data miners was finding the right algorithm for their task. Todays typical 2nd generation KDD Support Systems (KDDSS) overcome these issues by providing a plethora of operators. The commercial systems such as SAS Enterprise Miner and SPSS Clementine or the Open-Source Systems RapidMiner and MiningMart have 100+ (RapidMiner which includes WEKA even 500+) different operators to support modeling the KDD process. These KDDSS have eased the problem of providing access to applicable data mining operators. They do, however, give rise to a new problem: *How can data miners navigate the multitude of data mining operators to construct a valid and applicable data mining process?*

Indeed this problem gets even aggravated. When considering the whole KDD process [7] the universe of possible combinations is enormous. Consider CRISP-DM [4]: It consists of 6 phases and 24 tasks. Granted 6 of the tasks are of

an organizational nature, but the remaining 18 technical tasks open a huge design space [16] of possible KDD processes. As a consequence, most users are overwhelmed with the decisions they have to face and only explore a small part of the possible design space.

In this paper we propose that one should use a *cooperative planning* approach for designing the KDD process. A third generation KDDSS should employ a mixed initiative planning [8] approach to user interaction simplifying the following tasks:

- Effective representation of hundreds of operators used in KDD-workflows.
- Checking the correctness of KDD workflows
- Enumeration of (partial) KDD workflows
- Retrieval of previous (partial) KDD workflows
- Understanding and explanation of given KDD workflows
- Adaptation and repair of KDD workflows

In this paper we focus on the base of such a cooperative KDDSS. Specifically, we present a Data Mining Ontology (DMO) able to effectively organize hundreds of operators, which is the base for checking the correctness of KDD workflows and an HTN based planning component able to effectively enumerate useful KDD-workflows[3]. This is the base for our future work concerning a KDD-workflow explanation component and a KDD process adaptation component, which can be seen as a case-based planning component assuming that a case base is available.

In the remainder of the paper we first outline related work, and then discuss DMO and HTN in detail.

2 Previous Work

Several attempts have been made to create KDD support systems but none of them provides full support for generation of KDD workflows.

Žáková et. al [18] tried to automatically generate workflows using a knowledge ontology (DM ontology) and a planning algorithm based on the FastForward (FF) system. They limit themselves only to return the optimal workflow with the smallest number of processing steps, any other alternative workflows are not generated. The system does not involve user interaction during workflow generation.

The IDEA system [2] consists of an Intelligent Discovery Assistant (IDA) which provides users with systematic enumerations of valid DM processes and effective process rankings by different criteria (speed, accuracy, etc.). It is based on an ontology of DM operators that guides the workflow composition and contains heuristics for the ranking of different alternatives. The user is guided in her choices by choosing weights to trade-off the rankings of the alternatives along

[3] This is a report about work in progress. Check http://www.e-lico.eu/eProPlan to see the current state of the ontology as well as to download the Protege plug-ins we released so far.

the different dimensions (e.g., speed, accuracy, comprehensiblity) of her desiderata. The rankings are based on heuristics contained in the ontology as well as auto-experimentation.
The NExT system [1] is an extension of IDEA using Semantic Web technology Specifically it employs an OWL-S [11] ontology to specify the DM operators and relies on XPlan [10] for planning. While IDEA and NExT provide the user with a number of alternatives and guide the choice among them they are limited to proposing simple, non-splitting process flows, unless specific splitting templates are provided.

MiningMart [12] is a KDD preprocessing tool specialized in data mining on large data stored in relational databases. The meta-model M4 is used to store data on two levels: the logic level – describes the database schema and allows consistent access to information and the conceptual (ontology) level – uses concepts with features and relationships to model data [6]. The ontology captures all the data preprocessing therefore gives a better understanding and reuse of the data. However this meta-model is expressed in UML/XML and not in an ontology language. The system lacks automatic workflow creation of DM processes but enables the reuse of preprocessing phases across applications through case-based reasoning.

The CITRUS project [17] consists of an IDA which offers user-guidance through mostly a manual process of building the workflows. It uses planning for plan decomposition and refinement. The system is based on an extension of SPSS Clementine, which provides a visual interface to construct DM processes manually.

3 A Data Mining Ontology (DMO) for Planning

We designed a Data Mining Ontology (DMO) to contain all the information necessary to support a 3rd generation KDDSS. As Figure 1 shows (left to right), the DMO contains information about the *objects manipulated* by the KDDSS (*I/O-Object*), the *Meta Data* needed, the *operators* (i.e., algorithms) used by the tool, and a *goal description* that formalizes the user's desiderata. Here we succinctly describe the *I/O-Object* and the Meta Data before providing a slightly more extensive discussion of the Goals and Operators in the following subsections.

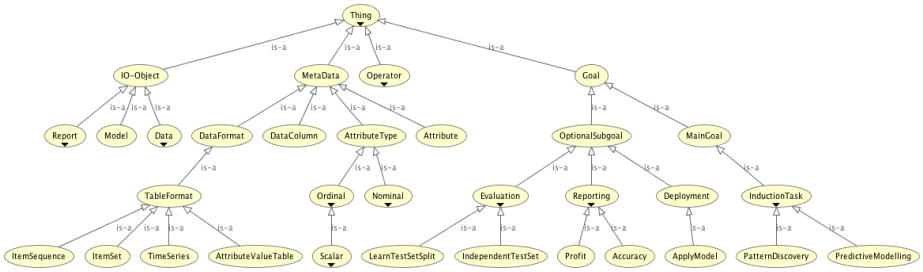

Fig. 1. The Upper-Structure of the Data Mining Ontology (simplified)

*I/O-Object*s are everything that is used or produced by operators. Every *I/O-Object* produced by an operator can be used as an input to any operator occurring at some later stage of the KDD workflow. Examples of *I/O-Object*s are *Data* (used and produced), *Model*s (used and produced) and *Report*s (produced only).

All of the above have several sub-concepts to specify the description of operator inputs and outputs I/O in a more specific fashion and to be used as conditions and effects of operators.

Sub-concepts of *Data* could be *AttributeValueDataTable*, *MissingValueFreeData*, *ScalarDataTable*, *NominalDataTable*, *TimeSeries*, *UnivariateTimeSeries*, *MultivariateTimeSeries*, etc. Prediction models, delivered by data mining operators are, e. g., *DecisionTree*, *Decisionlists*, *RuleSet*, *LinearModel*, ...

MetaData is used to describe *I/O-Object*s in more detail, i.e., the *DataFormat* such as tables, relations, images, etc. *DataColumns* can take specific value types like numerical or categorical values, and have particular roles such as "weight", "id", or "label".

Furthermore, the meta data can contain aggregate information about the values found in this column. Examples are mean, variance, range, modal value, or a flag indicating whether there are missing values.

3.1 Specification of KDD goals and input

A planner requires a goal description consisting of a start state and a final (or goal) state. A KDD-Workflow final state is reached when the workflow solves the "Data Mining Goal" ([4], Sec. II.1.3) and also contains all the evaluation and reporting needed to let the user assess if it fulfills the "Success Criterion". Additionally, all conditions of the operators included in the workflow have to be fulfilled.

In the DMO we model goal descriptions using subclasses of *Goal* (see Figure 1 for a simplified extract.). When specifying the "Data Mining Goal" the user has to choose (or compose) a subclass of *MainGoal*. This main goal can be extended with compatible *OptionalSubGoal*s (which ones are compatible is modeled with object-properties in the DMO). For example, the user can specify "I would like a KDD-Workflow for *PredictiveModeling*, evaluated on an *IndependentTestSet*, where the performance is reported by a *Profit*-Chart."

Several of these Goals require (again modeled as an object-property) that input (data) is available. Namely, *PredictiveModeling* requires *trainingData* and *IndependentTestSet* requires *testData*. So the user is prompted to specify a data file to be analyzed and to extend it according the meta-data in the DMO.

3.2 Operator Ontology

Operators in KDD-workflows differ from operators usually used in planning in two very important aspects. First, they do produce new objects. Consequently, we face a (potentially) infinite planning domain and not a finite one as most previous work about planning assumes. Second, they only produce new things

but never destroy old things. Thus, our world is monotonically growing with additional operator applications, i.e., everything that was valid (or known) in our world before executing an operator is still valid (known) afterwards. An infinite

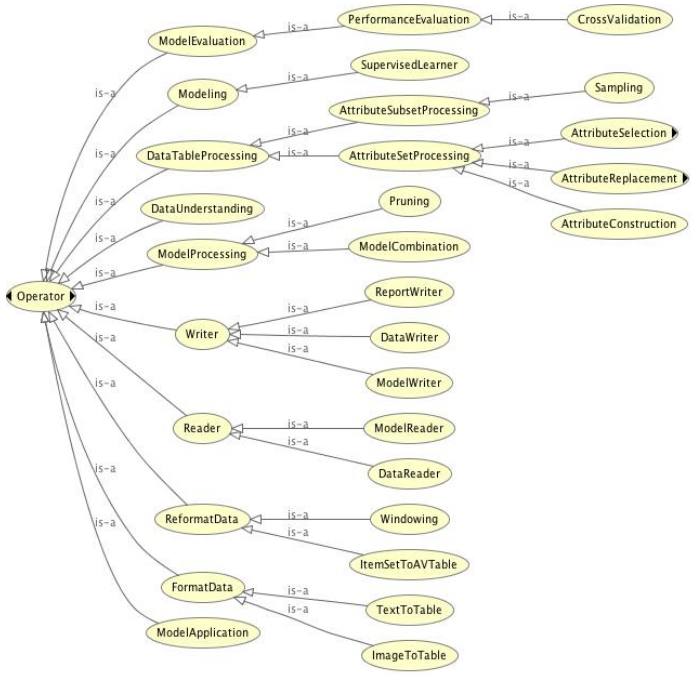

Fig. 2. The Top-Level Abstract Operators in the DMO

planning domain usually means that it is undecidable if a planning problem has a solution. Fortunately, the existence of plans is not really a concern for KDD-workflows. They always have a number of trivial solutions. E. g. for prediction tasks we could just compute the mean value or modal value of the target attribute, and build a model that always predicts that value, ignoring everything else.

Thus, we are not only looking for a plan, but rather for a good plan. The quality of a plan can be guided by the meta data information by which we enrich the objects exchanged by our operators. Note that the quality of a plan can be measured in various ways, but that some of the information is not available at plan-time. Especially not, the quality of a plan measured as expected predictive quality of a prediction model, e.g. its accuracy. This main success criterion of Data Mining is not available before executing the plan. Also, in real-world applications, we have to respect estimated non-functional constraints such as memory consumption and computation time of the workflow during planning.

As an example, consider the problem of cleaning a data set from missing values. Whereas a missing value imputation operation which replaces missing values

by training a model for each column containing missing values, is usually the preferable method, this may be prohibitively expensive in terms of computation time when the number of columns is large. In that case, it may be better to go with the simpler solution of filling in missing values by using the column mean value. In order for a planner to be capable of taking these considerations into account, we annotate operators and I/O-Objects with the respective information.

"RapidMiner.ID3":

Superclass: ClassificationLearning **and**
(*uses* **exactly 1** *AttributeValueDataTable*) **and**
(*produces* **exactly 1** *Model*) **and**
(*simpleParameter1*(name="minimal_size_for_split") **exactly 1** integer) **and**
(*simpleParameter2*(name="minimal_leaf_size") **exactly 1** integer) **and**
(*simpleParameter3*(name="minimal_gain") **exactly 1** real)

Condition: (*AttributeValueDataTable* **and** *MissingValueFreeData* **and**
(*inputAttribute* **only** (*hasAttributeType* **only** *Categorial*)) **and**
(*targetAttribute* **exactly 1** (*hasAttributeType* **only** *Categorial*))
)(?D), noOfRecords(?D,?Size), ?P1 is ?Size / 100
→ *uses*(**this**,?D), *simpleParameter2*(**this**,?P1)

Effect: *uses*(**this**,?D), *hasFormat*(?D,?F), *inputAttribute*(?D,?IA),*targetAttribute*(?D,?TA),
→ new(?M,?D), *DecisionTree*(?M), produces(**this**,?M), hasFormat(?M,?F)
inputAttribute(?M,?IA),*predictedAttribute*(?M,?TA),

Fig. 3. A basic operator: RapidMiner.ID3

Operator in- and output and parameters. The behaviour of each Data Mining operator is controlled by a set of parameters usually specified as key-value-pairs for an operator. Such parameters can be quite simple (e.g. k in a k-fold cross validation), quite complex (such as a reference to a column in a *DataTable* on which the operator is supposed to operate or an SQL statement as a target expression for attribute construction), or sometimes even structured (like a list of parameters to be optimized by an optimization operator).

Such restrictions on in- and output and parameters of operators are specified in OWL-DL. Every input of any operator must be a sub-object-property of *uses*, every output of any operator must be a sub-object-property of *produces*, and every parameter of any operator must be a sub-data-property of *simpleParameter*. *uses* and *produces* have a domain restriction to *Operator* and a range restriction to *I/O-Object*. Figure 3 shows an example.

Operator conditions and effects. Operators may specify restrictions on their input. E. g., most data mining operators will only operate on data sets satisfying particular properties (e.g., having numerical or categorical attributes, or not containing any missing values).

The effects of operators depend on parameter values and on the operator's input. E. g., a discretization operator's output is equal to its input with the exception of a set of columns, which are specified by a parameter. In the output data set, these columns' value types will be changed to categorical.

Operator conditions and effects are described as rules expressed in the Semantic Web Rule Language (SWRL [9]). As an extension to SWRL we have the special object reference **this**, which is the operator instance to which the rule refers. Also, we added three new built-in predicates that allow the creation of new unique object IDs for the generated outputs in the operator effects. They are permitted in the consequent of effect-rules only. These are:

new(NewObject). Exactly one new unique object ID is generated and bound to the argument variable for each such literal (independent of how many solutions satisfy the premisses).

new(NewObject,OldObject). Exactly one new unique object ID is generated and bound to the argument variable for EVERY different OldObject computed by the preconditions (independently of how often this OldObject is part of a solution satisfying the premisses).

copy(NewObject,OldObject,Except). One new unique object ID is generated and bound to the argument variable for EVERY different OldObject computed by the preconditions (independently how often this OldObject is part of a solution satisfying the premisses), additionally all stored property facts involving the OldObject except those in Except are copied to the NewObject. Except is a conjunction of property literals where the OldObject must be one of the arguments.

The condition of an operator is a set of SWRL rules, where the antecedent specifies the condition and the consequent contains all the input- and parameter-properties of the operator. Effects are also sets of SWRL rules, they can contain all three new built-in predicates, must contains all the output-properties of the operator and may also create new meta-data to describe the new *I/O-Objects*. All variables that occur in the conclusion of a condition or effect rule must be either bound by the premisses of the effect rule or be returned as a NewObject by one of the above special literals. If there is more than one rule for conditions and or effects, the resulting condition/effect-rule is the union of all antecedent of the condition/effect-rules imply the union of all consequents of the condition/effect-rule.

Operator nesting. Operators can be nested, i.e., they can contain sub-workflows. E. g., a cross-validation can contain a data mining operator for generating a prediction model from a training set and an operator for applying the model on a test set. But usually, such a nesting is not restricted to single operators, but may involve complex workflows. Therefore we allow such operators to reference nested HTN tasks (see Section 4). Similar to the post-conditions specified for each operator, we specify conditions that the enclosing operator guarantees to and requires from its nested workflow.

Operator Inheritance & Subsumption Any operator not only inherits the IO restrictions from it's super classes (via normal ontological reasoning), it also inherits all conditions and effects from them (to be handled by our operator

extension). If an operator has several rules for conditions or effects, the resulting condition- or effect-rule is the union of all antecedent → the union of all consequent. This leads to the following intuitive definition of an operator subsuming another:

1. the more special operator satisfies all input, output, and parameter restrictions of the more general operator,
2. the more general operator is applicable in all the situations, the more specific operator can be applied to, and
3. the more specific operator has at least all the effects of the more general one.

Formally, this can be captured by combining class subsumption ($\sqsubseteq_{DL}$) for I/O restrictions and instance reasoning with respect to a set of ontological axioms (a tbox) ($\models_{(tbox)}$) together with θ-subsumption for conditions and effects.

Definition 1 (Operator Subsumption). *An operator OP_1 subsumes another operator OP_2 (written $OP_1 \sqsubseteq_{OP} OP_2$), iff $OP_1 \sqsubseteq_{DL} OP_2$, and there exist a substitution θ_C for the condition and θ_E for the effects such that:*

- $antecedent(condition(OP_2)) \models_{(tbox)} \theta_C\, antecedent(condition(OP_1))$,
- $consequent(condition(OP_2)) \models_{(tbox)} \theta_C\, consequent(condition(OP_1))$
- $antecedent(effects(OP_2)) \models_{(tbox)} \theta_E\, antecedent(effects(OP_1))$, *and*
- $consequent(effects(OP_2)) \models_{(tbox)} \theta_E\, consequent(effects(OP_1))$.

3.3 Using OWL-S to import operators

The operators present in the DM ontology are of course limited to the existing operators sources (RapidMiner, Weka, etc.). As new operators will be developed our ontology must be extendible, to allow the users to use them for planning. Thus, we envisaged a way of extending the ontology with new operators based on a description of the operators in a language compatible with the specification from our ontology.

Each new operator has to be available as a Web service, described using a description language (WSDL) and needs to be semantically described in OWL-S [11]. I.e. it describes what it does in the *ServiceProfile*, how it works in the *ServiceModel* and how to access it in the *ServiceGrounding*. An operator can be added in the DM ontology if the given OWL-S description matches our operator description. From the OWL-S description we are mainly interested in the *ServiceProfile* and *ServiceGrounding*, while the *ServiceModel* does not concern our approach for now.

The *ServiceProfile* describes what the service does, it is similar to our *Operator* concept from the DM ontology. Both the ontology and OWL-S have to specify the Inputs, Outputs, Preconditions and Effects. The Preconditions and Effects should be specified in SWRL to be compatible to our representation from the ontology. The Inputs and Outputs need to be described by using the concepts from our ontology.

ServiceGrounding specifies how the service can be accessed and executed. It contains the WSDL location of a given operator that can be stored in our ontology as an annotation of the operator to be passed to an execution engine.

Therefore enhacing the operators with OWL-S facilitates the process of importing new operators thus allowing a flexible and standardized way to enrich the DM ontology.

4 An HTN for Data Mining

Hierarchical Task Network planning (HTN) [14, 13] originates from more than 30 years ago. It provides a powerful planning strategy using hierarchical abstraction and by that is able to handle large and complicated real world domains [13]. Also it is more expressive than classical planning being able to express undecidable problems, and therefore it is also[4] undecidable, if a plan exists in general [13].

Recently, AI planning techniques have been proposed as a way to automate (totally or partially) workflow composition especially Web services composition [15, 10]. Several planning techniques were compared in the context of Web services composition and as a conclusion HTN planning performs best in automation of Web services composition [3].

For us an HTN (similar to [5, 13]) consists of the following:

- A set of available *task*s to achieve the possible *Goal*s.
- Each *task* has an I/O specification (a list of property - ?variable : class) and a set of associated *method*s (that share the I/O specification of the task) that can execute it.
- Each *method* has a
  condition restricting it's possible applications, and a
  contribution specifying which *Goal*s it reaches, and a
  decomposition into a sequence of (sub-)*task*s and/or *operator*s, that - executed in that order - achieve the task.

Therefore an HTN planning problem consists of decomposing the initial task using applicable method that contribute to the current goals into applicable operators and sub-tasks and then recursively decompose each sub-task until we obtain a sequence of applicable operators only.

Fig.4 shows an example HTN to illustrate how one could specify the generation of KDD-workflows.

Even this simple HTN already contains some special built-in *task*s, namely the getPlan/reapplyPlan and the chooseOp/applyOp pairs of *task*s. It also illustrates the use of negation as failure (unknown) available in conditions. getPlan normaly plans the *task* it gets as it's first argument, it only additionally returns the plan it generated for that *task*, such that it can be reused later again. If the training data need a transformation before modeling this transformation needs

[4] The introduction of new objects, leads to potentially infinite - and therefore undecidable - planning problems (Sec.3.2).

Task: DoDatamining
I/O: [*goal* - ?G:*UserGoal*].
Method: "Propositional predictive Data Mining of attribute value data".
condition: *trainingData*(?G,?RTD), *AttributeValueDataTable*(?RTD),
unknown(*subgoal*(?G, ?E), *LearnTestSetSplit*(?E))
contribution: *PredictiveModeling*(?G).
decomposition:
ReadData([produces - ?RTD]),
chooseOp(*SupervisedLearner*([in - ?RTD]),?G,?ModelOp,?Requirements),
getPlan(TransformData([in - ?RTD, out - ?TD, goal -?Requirements]),?DataTransformationPlan),
applyOp(?ModelOp,[uses -?TD, produces - ?M]),
ReadData([produces - ?RED]),
OptionalEvaluation([goal - ?G, *model* - ?M, preprocessing - ?DataTransformationPlan]),
OptionalApplication([goal - ?G, *model* - ?M, preprocessing - ?DataTransformationPlan]).
Method:
...
Task: OptionalEvaluation
I/O: [*goal* - ?G:*UserGoal*, *model* - ?M:*Model*, preprocessing - ?DataTransformationPlan:Plan]
Method: "No evaluation wanted"
condition: unknown(*subgoal*(?G, ?E), *Evaluation*(?E))
contribution: .
decomposition: .
Method: "Evaluation of model performance on independent testdata"
condition: *subgoal*(?G, ?E), *testData*(?E,?RED), *AttributeValueDataTable*(?RED).
contribution: *subgoal*(?G, ?E), *IndependentTestSet*(?E).
decomposition:
reapplyPlan([in- ?RED, out - ?ED],?DataTransformationPlan]),
EvaluateModel([data - ?ED, model - ?M, produces - ?E),
GenerateReport([uses - ?E]).

Fig. 4. A simple extract from our HTN under development

to be reapplied (using reapplyPlan) on evaluation and application data, before the learned model can be applied to them as well. This is not just a simple "do the same again". The data transformation generated by the planner may involve operations like "feature selection", "Discretisation", "random sampling" that would behave differently on different data. Therefore the reapplication of plans must adapt the plans to corresponding dual operations "Select the same features", "Discretize into same Bins" "don't sample", i.e. all data dependent operations have to store their choices into "preprocessing models" and must have a dual re-application operator (specified in the ontology). HTN planning does straight forward planning, i.e. when an operator is applied all its input is already produced and available. However, in KDD-workflows selection of the modeling operator is usually done before preprocessing to set up the goals of preprocessing. This is achieved with the chooseOp/applyOp pair. chooseOp selects[5] an operator from the operator class specified in its first argument just as normal planning does, but opposed to planning it doesn't test applicability nor does it execute the effects, it only returns the conditions and the chosen operator. applyOp is just a meta-call that allows the operator to be a variable.

[5] Currently just by enumeration, maybe later with some heuristic guidance

5 Conclusions

In this paper we presented the basis of an open system for cooperative planning of KDD-Workflows. The system is currently developed within the e-LICO project as a set of Protege 4 plugins. The plugins (goal editor, condition/effect editor, OWL-S im-/export, restricted abox/rule reasoning, operator subsumption, workflow-checking and an HTN-planner) as well as the DMO are to be released to the public within this year.

We expect HTN-planning to be more successful in generating useful KDD-workflows, in the presence of a realistic (high) amount of operators, than forward planning tried in previous work so far, as the HTN grammar rules allow additional expert knowledge to be coded into the plan generation process, which is not possible in simple STRIPS-like forward planning. Nevertheless, we expect the most gain in user-productivity of the 1st version described here lies in the correctness-checking of (user-designed) KDD-workflows at design-time – checking for slight errors in operator applicability that are already detected before execution and will not crash the workflow execution (after some hours).

Planning of KDD-workflows is always limited by the information available at design/plan time. However designing optimal performing KDD-workflows is still a highly creative and interactive process, where most insights and design decisions are based on and revised according to the performance measured in the evaluation-phase of the KDD-workflows, i.e. with information available after execution and not at plan-time.

Data-Mining-Performance optimized KDD-workflows resulting from this expensive iterative design cycle are, therefore, very valuable and should be recorded in a case-base. Especially as the current knowledge on data mining does not allow to distinguish good from bad performing workflows without executing them. For future work we plan to extend the system to case-based planning, i.e. retrieval and adaptation of partial fitting KDD-workflows. This will allow to make this valuable and costly acquired knowledge available to later workflow design.

Acknowledgements: This work is partially supported by the European Community 7^{th} framework program ICT-2007.4.4 under grant number 231519 "e-Lico: An e-Laboratory for Interdisciplinary Collaborative Research in Data Mining and Data-Intensive Science". The DMO described in this paper is the result of ongoing collaborative work within the e-LICO project of Ingo Mierswa, Melanie Hilario, Alexandros Kalousis, Nguyen Phong and the authors. For ease of presentation we made some simplifications with respect to the full e-Lico DMO.

References

1. A. Bernstein and M. Daenzer. The NExT System: Towards True Dynamic Adaptions of Semantic Web Service Compositions (System Description). In *Proceedings of the 4th European Semantic Web Conference (ESWC '07)*. Springer, March 2007.
2. 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.
3. K. S. M. Chan, J. Bishop, and L. Baresi. Survey and comparison of planning techniques for web services composition. Technical report, Univ. of Pretoria, 2007.
4. 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.
5. K. Erol, J. Hendler, and D. Nau. HTN planning: Complexity and expressivity. In *Proceedings of the National Conference on Artificial Intelligence*, pages 1123–1123. JOHN WILEY & SONS LTD, 1995.
6. T. Euler and M. Scholz. Using Ontologies in a KDD Workbench. In P. Buitelaar, J. Franke, M. Grobelnik, G. Paa?, and V. Svatek, editors, *Workshop on Knowledge Discovery and Ontologies at ECML/PKDD '04*, pages 103–108, 2004.
7. U. M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. The kdd process for extracting useful knowledge from volumes of data. *Commun. ACM*, 39(11):27–34, 1996.
8. G. Ferguson, J. Allen, and B. Miller. Trains-95: Towards a mixed-initiative planning assistant. In *Proceedings of the Third Conference on Artificial Intelligence Planning Systems (AIPS-96)*, pages 70–77. AAAI Press, 1996.
9. I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*. http://www.w3.org/Submission/SWRL/, 2004.
10. M. Klusch, A. Gerber, and M. Schmidt. Semantic Web Service Composition Planning with OWLS-XPlan. In *Proceedings of the 1st Intl. AAAI Fall Symposium on Agents and the Semantic Web*. AAAI Press, 2005.
11. D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara. *OWL-S: Semantic Markup for Web Services*. http://www.w3.org/Submission/OWL-S/, 2004.
12. K. Morik and M. Scholz. The MiningMart Approach to Knowledge Discovery in Databases. In N. Zhong and J. Liu, editors, *Intelligent Technologies for Information Analysis*, pages 47–65. Springer, 2004.
13. D. Nau, M. Ghallab, and P. Traverso. *Automated Planning: Theory & Practice*. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 2004.
14. D. Nau, S. Smith, and K. Erol. Control strategies in HTN planning: Theory versus practice. In *Proceedings of the National Conference on Artificial Intelligence*, pages 1127–1133. JOHN WILEY & SONS LTD, 1998.
15. E. Sirin, B. Parsia, D. Wu, J. Hendler, and D. Nau. HTN planning for web service composition using SHOP2. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(4):377–396, 2004.
16. K. T. Ulrich and S. D. Eppinger. *Product Design and Development*. McGraw-Hill, New York, 3rd rev. ed. edition, 2003.
17. 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.
18. 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.