Planning tasks for Knowledge Discovery in Databases; Performing Task-Oriented User-Guidance.

Robert Engels*
University of Karlsruhe, Institute AIFB,
D-76128, Karlsruhe, Germany
Email: engels@aifb.uni-karlsruhe.de

Abstract

Performing the complex task of Knowledge Discovery in Databases (KDD) requires a break-down of the task-complexity to enable the possibility of performing the KDD-task. Since even more techniques will appear in the future that can solve a variety of KDD-problems, a domain expert that wants to analyse his domain should have the means to work with tools that integrate several of these techniques as well as the techniques themselves. In this paper a framework is proposed for a strategy component that is to be used for a KDD-system that can guide users in breaking down the complexity of a typical KDD-task and supports him in selecting and using several ML-techniques. The goals of such a guidance component are *reuse* of (predefined) taskcomponents in order to decrease development time and to simplify the process of decomposing a KDD-task, task-oriented *planning* in order to break down complexity of a typical KDD-task and *supporting post-processing* (evaluation) of KDD-processes.

Keywords : *User-support, Knowledge Discovery in Databases, Task-decomposition, Reusable Process Components*

Providing User-Guidance for KDD-processes

The question whether there is a need for user-guidance or not is already answered by several studies mentioned in the literature (see i.e. (Consortium 1993), (Mladeniĉ 1995), (Ram & Hunter 1992), (Hunter 1995)). The answer to the "how" question is not so easily answered. There are several approaches that provide a kind of user-support, but a real guidance of a user that helps planning KDD-processes by building up task-decompositions according to the users intentions and helping him to refine and alter his problem statement if necessary is not common.

The framework we propose partly stems from ideas of the field of knowledge acquisition where task-decomposition and reusable task-descriptions are used to describe and specify complex tasks. The user that we see as a typical user of our system is a domain expert that does not have extensive knowledge about KDD and wants an additional tool for the analysis of his database. The expert also has some understanding of the KDD-task in the sense that he knows a little about the several different core tasks that underlie a KDD-process (i.e. visualisation, classification, dependencies, prediction).

User-guidance in a KDD-session is started (as mentioned) with a description of the problem by the expert. This description is to be refined until a clear goal state is defined. This is to be done iteratively and interactively (we do not expect the process to be totally automatical) with a knowledge engineer because it will seldom be the case that the initial problem description is clearly enough specified to form a beginning of the planning process. Once the goal state is described it can be mapped upon a high-level task, where we assume a set of predefined high-level tasks that are defined by the system. Such a high-level task should then be decomposed in a task-structure that breaks down the complexity of the initial task in a sequence of subtasks that together concretise the task and provide a solution for the initial (planning) problem. The task decomposition process consists of a process that has two dimensions, namely the identification of a sequence of subtasks that can perform a certain task, and the identification of several alternative sequences for these subtasks. Pre- and postconditions are defined over tasksequences, as well as a control-flow that is defined over them. The solution (as we will call it, see also section) then consists of a set of algorithms that can be executed in order to get the required "valid, novel, potentially useful, and ultimately understandable patterns in data" as defined in the definition of KDD (Fayyad *et al.* 1996).

*This work has been partially funded by the Daimler Benz AG, Research & Technology, project no. 096 4 965047 1 E B. Official version published by the American Association for Artificial Intelligence (http://www.aaai.org). Copyright 1996 American Association for Artificial Intelligence. All rights reserved.

Terminology

In this section we will clarify the terminology that is used in our approach to KDD-process guidance. We see this step as necessary because of the diversity in terminology that can be found when analyzing the literature and in order to clear the context we want to use.

The terminology we want to clarify is the terminology that relates to the so-called KDD- guidance module. In the following we clarify the concept of a problem, problem description, tasks, task decomposition, techniques, solution, input data, discovered knowledge and mapping.

Problem: A problem is defined in general as an artifact that contains three elements, an initial state, a goal state and a discrepancy between those.

A well known and usefull distinction at this point can be made between ill-defined problems and well-defined problems, where a problem is called ill-defined when[1]: the goal state is not clearly described, the initial state is not clear (some characteristics relevant to the initial state might be unknown) or the information needed to solve the problem is not available, i.e. a problem is also ill-defined if the characteristics of the initial state are not relatable to the characteristics of the goal state, because then there is no possibility to compare the two. A well-defined problem is a problem were the initial and the goal state are known, and where there is only one pair containing a unique initial state and a goal state, and a discrepancy between the two for which no existing solution exists.
A problem can be stated in a natural language form. It forms the basis upon which a problem description is defined.

Problem Description: A problem description is a pair that consists of a single goal state for the system, and a description of the initial state in which the system is.

The description of the initial state can contain more knowledge than is required to define the discrepancy between the initial and the goal state.

A problem description results from the process of transforming a situation in which only an ill-defined problem exists into a situation in which a well-defined problem exists. The domain expert iteratively refines his/her initial problem statement with regard to indistinctnesses that a knowledge engineer recovers.

Tasks: A task consists of a certain set of inferences that, when executed, transforms a certain state into another state. A task consists of an action that, when performed, bridges the gap between an initial and a goal state.

A task has certain pre- and postconditions, where the preconditions define the possible deployment domain of a task (task resources), and the postconditions the characteristics of the knowledge that results when the task is performed (task effects).

Task Decomposition: A task decomposition is a refinement of a task into a task/subtask hierarchy.

The top-level task is identical to the task to be composed, and the leaves of the tree form a set of subtasks to be performed that together can perform the task at the top-node level. A (sub-) task that can not be further decomposed is called a generic or primitive task ((Chandrasekaran, Johnson, & Smith 1992)).

Problem Solving Methods: A Problem Solving Method[2] typically describes how to solve a task by decomposing and defines an order on the subtasks in the decomposition through a controlflow.

One speaks of a task-subtask decomposition if one can reach the functionality of a task by performing a (partial) order of subtasks. This kind of refinement of a task with a subtask structure is a *conjunctive* refinement. The functionality of a certain task is defined in terms of pre- and postconditions of the task and also depends on the (partial) ordering posed upon the subtasks by the controlflow. Typically, the task decomposition can cause a refinement of the tasks' functionality since a subtask might introduce additional or refined pre- and postconditions. The reason for this is that the subtasks of a task would be more specific than their higher-level parents.

Techniques: Techniques refer to algorithms that,when executed, can reach the defined functionality of a (generic) task. Such an algorithm does not necessarily need to be a data mining tool in our context, but can also be a (set of) manual steps that should be performed.

Techniques in our framework thus refer to the (ultimately) implemented algorithms that are available in the KDD-tool. Every technique describes an algorithm and therefore exists of an abstract artifact as well as a description of the parameters that are connected to it. Techniques typically come with parameters that are used for tuning the techniques for several different situations. The abstract artifact is the part that will be used for the mapping on the task-decomposition, where the parameter descriptions can be seen as a definition of the range of application of the artifact.

Solution: With a solution is meant the result of an iterative and interactive (planning-)process, where a domain expert, eventually in cooperation with a

[1] See also (Simon 1985) for a theorie on well- and ill-defined problems in an information-processing theory.

[2] For more about PSM's and their justification see (Schreiber, Wielinga, & Breuker 1993), (Breuker & van de Velde 1994), for an analysis of a typical PSM see (Fensel 1995).

knowledge engineer, transforms an "ill- defined" problem into a "well-defined" problem, and where this "well-defined" problem description (together with eventual additionally known domain characteristics) forms the starting point for a planning process in a (semi-) closed world. The result of this planning process is called a *solution* and consists of a task decomposition of a KDD-process that, when followed, provides a way to bridge the gap between initial and goal state in a problem description.

The pair of states that form the problem description forms the basis for a search-process is performed in a "semi-closed" world of possible task decompositions that will ultimately match a specific task-decomposition upon this pair of states. The search space is in principle closed, since one can expect to find a solution when one has a well-defined problem-description that fits on a certain top-level task, although this can not be garantueed in every case due to the uncertainty that is caused by the refinement operator that may introduce (unsatisfiable) new preconditions at every refinement step.

A solution then refers to a (controlflow-determined) sequence of techniques that, when executed, should reach the functionality of the top-node task that is mapped upon the problem-description.

Input Data: Input data refers to the data that is input to the KDD-process, i.e. the data that finally is to be analysed.

Input data can exist of the intensional and extensional descriptions of domain specific knowledge, and forms the data that is subject of analysis.

Discovered Knowledge: With discovered knowledge we refer to the results of the execution of the planned solution for the KDD-problem that is described in the problem description.

We explicitly make a distinction between a *solution* and *discovered knowledge*, where a solution refers to the *result of the planning* that is the main task of the guidance module (i.e. the order of algorithms and their parameter initialisations), and discovered knowledge, that is the result of the *execution* of our solution. The domain expert who provided the problem will be mainly interested in this "discovered knowledge".

Mapping: The mapping process means to map the task decomposition and the resulting controlflow that is defined over it, onto an ordered sequence of techniques that together can bridge the gap between the initial and goal state in the problem description.

This step connects the resulting task-decomposition to techniques. The process of decomposing and mapping is an iterative process that should also provide support for the parameter initialization of the several techniques.

The main terminology as we want to use has been described in this section. The next section will shortly describe the dimensions of which the guidance module consists of.

Dimensions of User-Guidance

User-guidance and support is the main task for our guidance module and is built up around two different dimensions. The first dimension deals with the planning process and supports a user with defining the "what" and "how" of his problem. With the "what" is meant the clarification of his problem (i.e. transforming an ill-defined in a well-defined problem), the "how" refers to the match of the resulting pair of states onto a high-level task that can dissolve the discrepance between initial and goalstate, and decomposing it into an order of subtasks.

The second dimension deals with the execution of the results of the planned task.

Planning the KDD-process

As mentioned, the first step when dealing with a planning task for KDD is to define what the problem really is. Experience with performing KDD-tasks learned us that most initial user-requests for data-analysis are ill-defined, that is, a request is most often vaguely described, lacks clarity, and it is impossible to decide whether there is a solution to the problem or not. If one does not carefully analyse the request it can happen that one analyses another problem as originally stated by the expert. At this point a lot of human interaction is needed of a knowledge engineer with the domain expert that provided the problem.

When a problem description is found, according to the definition in section , then a process of planning is started. The pair of states, that form the well-defined problem, as well as additional knowledge that might be known about the domain (data), form the starting point of a search for a task-decomposition that can bridge the gap between initial and goalstate. This task-decomposition can be seen as a kind of "augmented" hierarchy, since it is not only a hierarchy that provides subtasks for a task, but also poses an order upon those subtasks (as defined in controlflow of PSM-components).

So, summarized, we see the highlevel KDD-process definition in a schema-based way, where a task model for a typical KDD-task is taken to be the schema that should be filled out with building blocks (PSM's) that are partly pre-defined (but can be user-defined as well) according to the goals that are described in the problem-description.

The task-decomposition describes an order of tasks that should be performed in order to solve the problem. A mapping should then take place in order to select appropriate techniques that can perform this order of tasks. Once such an order is found one can go to the executional phase, where the tasks should be actually performed.

Execution of a KDD-solution

A logical step after the planning of the process is, of course, the execution of the set of techniques in the appropriate order and with the appropriate parameter settings.

That this is not a straightforward problem can be seen in the field of planning, where making plans was always treated as the main problem, but lately a growing interest is reported in the execution of such plans, that seem to cause additional problems when executed. One of these additional problems we have to tackle in our KDD-framework are the parameter settings of the several techniques. When executing, the functionality of the implemented techniques can be changed (sometimes a lot) by just changing a single parameter. Supporting the initialisation of those parameters is a prerequisite for a succesful execution of the solution found in the planning process. It might be that parameter settings are initialisable according to the functionality description that is provided by the task-decomposition. In our framework the parameter setting initialisation should also be supported by the mapping process that precedes the execution of the technique sequence. In case of failure there is a possibility to fall back on a set of defaults.

A KDD-process example

In this section we will provide an example of a KDD-process that is performed using a multi-strategy approach as defined in this paper. The example is performed in the field of prediction of warranty costs for cars, and uses data from a rather extensive database that contains all the registrations of new cars with their construction-characteristics and warranty costs.

From the field of KA, several problem solving methods can be taken that form a profound basis for reusable task-components.

In the following we will integrate the assessment PSM (Valente & Löckenhoff 1994) in the task decomposition that we define for our example problem. We propose a division of the KDD-solution for our problem in three subtasks that map on our initial taskmodel for KDD-processes as proposed in (Reinartz & Wirth 1995):

1. A "recognition"-task that has the goal to recognise and extract the relevant early warning cars from the dataset. In terms of our taskmodel this can be seen as a pre-processing stage where focussing on relevant data takes place, and a first learning step, where (given this relevant data) a grouping for data at a specific time t in several classes is performed.
2. A "classification"-task that produces classification rules for the distinguished classes.
3. A "deployment"-task that matches new data-tuples to the set of rules and thus assesses the class-membership of new tuples and that matches the set of EW-tuples (Early-Warning cars) that the "prediction"-task provides in order to match them to a certain (user-defined) treshold.

Actually, a fourth step should be integrated before the other three, namely a focussing step. It is not the case that the data as present in the database can realistically be used, since there are simply to many data-tuples, and too many of them are irrelevant as well. An example is the occurence of trucks and cars in the database, where the problem description already provide a focus on the subset of the tuples in the database that are concerned with cars.
Every task in the KDD-solution of figure 1 has its assumptions about the (meta-) knowledge it needs and about the output it can deliver. Being too specific does not help us much w.r.t. reuse, and being too general neither (a well- known trade-off in Software Engineering (Krüger 1992)).

The overall task of our example approach is decomposed in three steps at the first level of decomposition. First we will shortly repeat what the problem in our example was[3]:
Give a warning signal at time t if quarantuee costs at time t+x for a (certain type of) car will raise abnormally.
In the next sections we will describe the three subtasks with their decompositions: recognition, classification and deployment.
The "recognition"-task is itself further decomposable in three sub-tasks: a cluster subtask, that clusters a set of database tuples, an "incrementally add" subtask, and an assessment task that estimates the class allocation of the datatuples from the set of datatuples with time t-x. The next subtask on the first level of decomposition is concerned with the production of a set of classification rules for the classes that are found in the "assess class-allocation" subtask. This subtask is concerned with the deployment of these results. In the first subtask an assessment is made of the class-allocation of new datatuples at time t+x. The results of this assessment are gathered and form the input for the second subtask that provides the actual warning signal. Altogether the subdivision in subtasks made it possible to perform the rather complex task that was decomposed in three subtasks (each of which is further decomposed). These tasks where coupled to eachother through a set of pre- and postconditions that reassured the compatibility of the tasks among eachother. Doing so we have a way of defining the compatibility of the subtasks, as well as a way for guiding the planning of technique selection as well as an order in which to execute the selected techniques. Reuse of process components is shown by the multiple usage of a specific PSM, although the example (due to space-restrictions)

[3]Where the timeline covers a range from $t - x$ (x days/months/years in the past) over t (present) to $t + x$ (x days/months/years in the future).

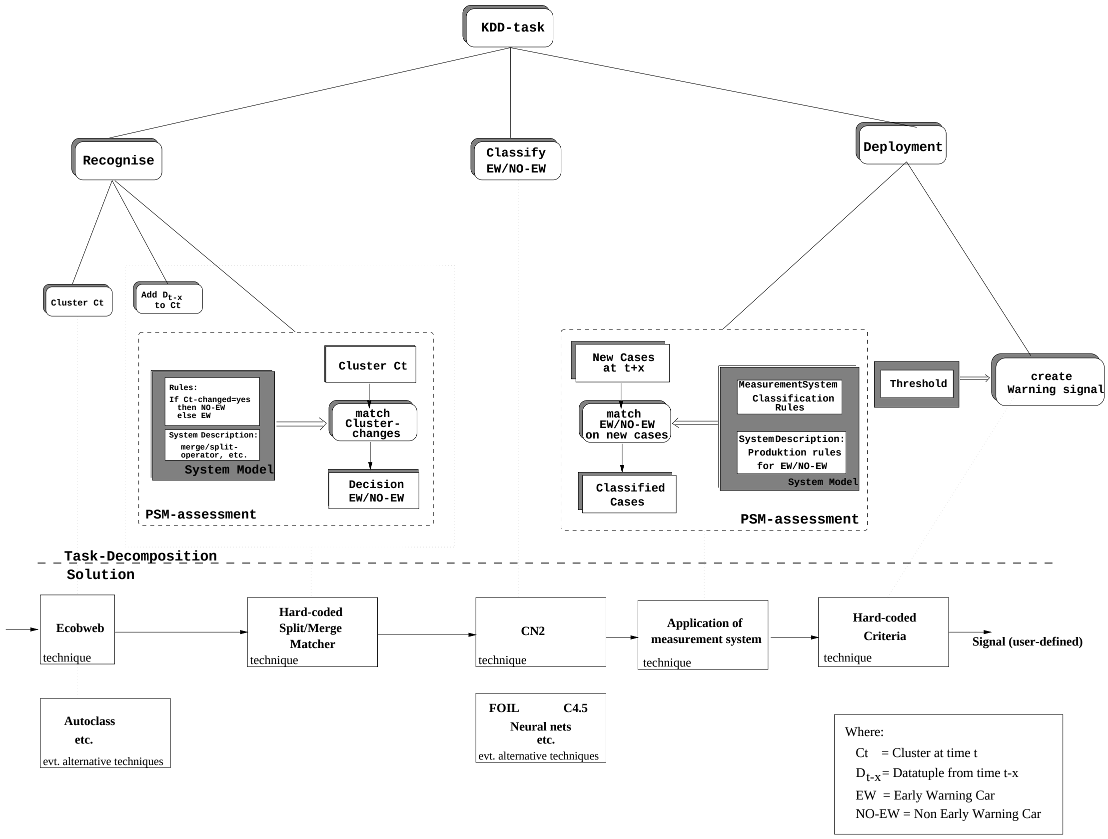

Figure 1: Taskdecomposition of our KDD-solution for the application problem showing integration of the PSM assessment in a (KDD) task decomposition.

had to be kept abstract.

Conclusions and related work

For our purposes we see an opportunity to integrate ideas of the several involved disciplines to support breaking down task-complexity of KDD-processes by using task-decompositions. The framework we show here provides a means of flexible planning of a KDD-process according to goals that are made explicit. The user gets support in the sense of pre-defined task-decomposition components and is provided with a means to define those themselve as well. The need for a kind of task-related support is already noticed in (Consortium 1993) and (Thomas, Ganascia, & Laublet 1993).

Therefore we adapted ideas of the KA-community (for example, we did have a look at the ideas on knowledge level modeling as mentioned in KADS (Wielinga, Schreiber, & Breuker 1992), MIKE (Angele, Fensel, & Studer 1996), Components of Expertise (Steels 1990), General Directive Models (van Heijst 1995) and Generic Tasks, (Chandrasekaran, Johnson, & Smith 1992)) and extended the definition of KDD as proposed in (Fayyad *et al.* 1996) in order to introduce the idea of goal-driven KDD. We feel the definition as provided by (Fayyad *et al.* 1996) needed to be extended by agents, because the 'ultimately understandable patterns', mentioned in the definition, require the notion of an agent in order to be able to define the term 'understandable'. On the other hand we see the need for an extension of this definition with goals since there is a problem defining potentially usefull results when no goals are defined that relate results to needs.

From the field of KDD, the need for breaking down task-complexity in order to solve problems is recognized (see i.e. (Reinartz & Wirth 1995)). Certainly as even more tools will become available to perform data mining, tools that help users selecting and using the right tools for their problems should become available. However, during the KDD-95 conference ((Fayyad & Uthurusamy 1995)) there were not many approaches to be found that dealt with this task-decomposition and user-support problem. Earlier work on the ESPRIT project MLT ((Consortium 1993)), shows a project part that is concerned with the issue of user-guidance and support CONSULTANT (Craw *et al.* 1992), which is later further developed in MUSKRAT (Graner & Sleeman 1993), but no publications are found that do elaborate upon this idea of user-guidance according to tasks and task-decompositions.

An approach that is concerned with the reuse of software components rather than the reuse software processes is the OCAPI system (van den Elst, van Harmelen, & Thonnat 1995). In this approach the idea of modelling processes according to tasks is presented, but, as mentioned, the authors stress the reuse of components rather than of processes. Reuse of components as in OCAPI is also a topic in our framework, where these components exist of data mining techniques rather than image processing programs as in OCAPI. Both approaches share the modelling of pre- and post-conditions as well as the modelling of component parameters (that influence the components functionality) at an abstract level.

Furthermore we see a chance for reusing already defined KDD-processes in certain cases were a problem description defines a very similar problem to a previously solved problem (think for example of our specific task description that should be repeatedly performed in order to adjust predictions to new data that is gathered, or that instead of two classes more classes are to be analysed). Here we can see a straightforward need for some kind of reuse of whole KDD-processes, since there is no doubt that developing whole applications again for just a slightly changed problem is not necessary.

The example also showed, although at a rather abstract level, how one can integrate reusable components (like the assessment PSM as described in (Valente & Löckenhoff 1994)) in a task-decomposition. We see a possibility of augmenting a KDD-tool with these structures in order to have a kind of predefined structures that support the definition of certain subtasks, and to be able to define these in a flexible way.

User-guidance performed in this way delivers a solution in the sense that the user interactively defines a sequence of techniques that can solve his problem, and also gets feedback on whether he can expect an answer on his problem definition using the tool or that he should alter his problem definition (i.e. relaxing or specifying the requirements he states in his problem description).

Future work

For the future we plan the further evaluation and implementation of this framework and test it on some more applications. Problems we want to deal with are problems concerning the selection and retrieval of the reusable building blocks according to their functional descriptions, the support of the user by selection of appropriate parameter settings according to the task characteristics and the value of this for task iterations. Integration of more techniques into the framework is also planned.

Acknowledgements

Thanks to Rüdiger Wirth, Thomas Reinartz, the Daimler project group and Rudi Studer for fruitfull discussions on the topics presented in this paper.

References

Angele, J.; Fensel, D.; and Studer, R. 1996. Domain and task modelling in mike. In *Proceedings of the IFIP WG8.1/13.2 Joint Working Conference on Domain Knowledge for Interactive System Design.*

Breuker, J., and van de Velde, W. 1994. *CommonKADS Library for Expertise Modelling.* IOS Press.

Chandrasekaran, B.; Johnson, T. R.; and Smith, J. W. 1992. Task-structure analysis for knowledge modeling. *Communications of the ACM* 35(9):124–137.

Consortium, M. 1993. Final public report. Technical report. Esprit II Project 2154.

Craw, S.; Sleeman, D.; Granger, N.; Rissakis, M.; and Sharma, S. 1992. Consultant: Providing advice for the machine learning toolbox. In Bramer, M., and Milne, R., eds., *Research and Development in Expert Systems*, 5–23.

Fayyad, U., and Uthurusamy, R. 1995. *Proceedings of the first International Conference on Knowledge Discovery & Data Mining.* Menlo Park, California: AAAI Press.

Fayyad, U. M.; Piatetsky-Shapiro, G.; Smyth, P.; and Uthurasamy, R. 1996. *Advances in Knowledge Discovery and Data Mining.* Cambridge, London: MIT press.

Fensel, D. 1995. Assumptions and limitations of a problem-solving method: a case-study. *Proceedings of the 9th Banff Knowledge Acquisition for Knowledge-Based Systems Workhop.*

Graner, N., and Sleeman, D. 1993. Muskrat: a multi-strategy knowledge refinement and acquisition toolbox. In Michalski, R., and Tecuci, G., eds., *Proceedings of the Second International Workshop on Multistrategy Learning*, 107–119.

Hunter, L. 1995. Planning to learn. In Ram, A., and Leake, D., eds., *Goal-Driven Learning.* London, England: MIT press.

Krüger, C. W. 1992. Software reuse. *ACM Computing Surveys* 24(2):131–183.

Mladeniĉ, D. 1995. Automated model selection. In *Proceedings of the Knowledge Level Modelling and Machine Learning Workshop, Crete.*

Ram, A., and Hunter, L. 1992. The use of explicit goals for knowledge to guide inference and learning. *Applied Intelligence* 2(1):46–73.

Reinartz, T., and Wirth, R. 1995. Towards a task model for kdd-processes. In Kodratoff, Y.; Nakhaiezadeh, G.; and Taylor, C., eds., *Workshop notes* Statistics, Machine Learning, and Knowledge Discovery in Databases. *MLNet Familiarisation Workshop*, 19–24.

Schreiber, T.; Wielinga, B.; and Breuker, J. 1993. *KADS: A Principled Approach to Knowledge-Based System Development.* London: Academic Press.

Simon, H. 1985. Information-processing theory of human problem-solving. In Aitkenhead, A., and Slack, J., eds., *Issues in Cognitive Modeling.*, 253–278. Lawrence Erlbaum Ass., London.

Steels, L. 1990. Components of expertise. *AI Magazine.*

Thomas, J.; Ganascia, J.-G.; and Laublet, P. 1993. Model-driven knowledge acquisition and knowledge-biased machine learning: an example of a principled association. In *Workshop proceedings of the 13th Int. Joint Conference on Artificial Intelligence*, volume W16, 220–235.

Valente, A., and Löckenhoff, C. 1994. Assessment. In *(Breuker & van de Velde 1994)*, 155 – 174. IOS-Press.

van den Elst, J.; van Harmelen, F.; and Thonnat, M. 1995. Modelling software components for reuse. In *Seventh International Conference on Software Engineering and Knowledge Engineering*, 350–357. Knowledge Systems Institute.

van Heijst, G. 1995. *The Role of Ontologies in Knowledge Engineering.* Ph.D. Dissertation, University of Amsterdam.

Wielinga, B.; Schreiber, A.; and Breuker, J. 1992. Kads: A modelling approach to knowledge engineering. special issue "the kads approach to knowledge engineering". *Knowledge Acquisition* 4(1):5–53.