# Describing and Integrating Competence Theories for Problem-Solving Components and Machine Learning algorithms

Robert Engels and Rainer Perkuhn
Institute AIFB, University of Karlsruhe,
D-76128 Karlsruhe, Germany.
E-mail: [engels;perkuhn]@aifb.uni-karlsruhe.de
http://www.aifb.uni-karlsruhe.de/WBS/index.engl.html

**Abstract**

A topic in the field of knowledge acquisition is the reuse of components that are described at the knowledge level. Problems concern the description, indexing and retrieval of components. In our case there is the additional feature of integrating so-called *automated building blocks* in a knowledge level description. This paper describes what knowledge level descriptions of components for reuse should look like, and proposes a way to describe *assumptions and requirements* that are to be made explicit. In the paper an extension of the "normal" knowledge acquisition setting is made in the direction of machine learning components and their description and integration *at* the knowledge level.

**Keywords** : *Knowledge Acquisition, Machine Learning Algorithms, Competence Theories, Knowledge Level Descriptions, Multi-Strategy Learning*

## 1 Introduction

In the field of Knowledge Acquisition many approaches deal with a separation of several levels on which to represent knowledge (i.e KADS [WSB92], Components of Expertise [Ste90], Role-limiting methods [McD88], Generalized Directive Models [vH95] and Generic Tasks [Cha86, Cha88]). Most approaches make a distinction between knowledge at the domain layer (domain specific knowledge) and problem-solving knowledge represented at the generic level. One argument that justifies this approach is that systems described on several levels separate different types of knowledge which makes them better maintainable, more understandable and parts of it are better suited for reuse in other systems/domains. In order to make this claim come true, we feel that such a methodology should contain a way of describing the competence of its (knowledge level) components.

Knowledge acquisition approaches define such components explicitly. They are possible candidates for reuse, and form so-called Problem-Solving Methods (PSM's). These PSM's describe *how* they can perform a certain task, i.e. what steps need to be taken in order to reach a certain functionality. We concentrate on the generic parts of such PSM's.

Although the functionality of these PSM's (i.e. describing *what* a certain PSM can do) is an important topic for describing PSM's, few approaches make this competence explicit [WJ95].

The main focus of this paper will be concerned with describing the competence of the building blocks. Describing these re-usable components on the knowledge level has the advantage that one can define PSM's and Machine Learning algorithms at the same level of abstraction. Thereby we enable the integration of learning-algorithm descriptions with PSM-description at the knowledge level. On the one hand, the background for such an approach is provided by the need to support users of a multi-strategy learning system to break down the complexity of their tasks and guide them by the application of machine learning components to solve their problem. On the other hand we offer a uniform framework which defines pre- and postconditions for both PSM's and learning components a basic requirement for integration of those components on a generic level.

For this purpose we want to extend our MIKE[1] approach. In this paper we will therefore first introduce our MIKE-approach and second the notion of knowledge level descriptions and PSM's.

Also a formal way will be proposed in which one can describe the competence (functionality) of these components by their in- and output behaviour. As an example on which we want to clarify our points of view we take the assessment-model as defined in the *CommonKADS*-approach [BvdV94] and relate this to a knowledge level integration with a (classifying) ML-algorithm. The paper concludes with some related work and an evaluation of the approach.

# 2 MIKE

The MIKE-approach is a methodology that supports the whole cycle of building and implementing expert-systems. The approach supports the step of knowledge elicitation, building a semi-formal structural model of problem-solving and transforming this semi-formal model into the so-called Model of Expertise expressed in a formal specification language KARL (cf. [Fen95], [Ang93]). This process is supported by the MIKE-tool (cf. [Neu94], [Neu93]). The final design and implementation step is supported by the design-model (cf. [Lan95], [LS95]) which integrates non-functional requirements in the model.

In the context of this paper it is not relevant to describe the whole methodology, therefore we restrict ourselves to the part that will return in the discussion later on. The Model of Expertise in the MIKE-approach, as in KADS, encloses three layers: the task-layer, the inference-layer and the domain-layer.

The Model of Expertise aims to define the functionality of a system that performs a certain task in a certain domain using certain knowledge. The three layers describe how a task-definition relates to certain goals, how these relate to the actions that have to be performed and how this connects to the knowledge at the domain layer. A generic configuration of a task- and inference-layer at a reusable level of abstraction forms a so-called PSM. Usually a PSM is used to describe the data- and control flow internal to a problem solving process. In many expert systems such an exhaustive description of a problem-solving process is satisfying.

[1]Model-Based and Incremental Knowledge Engineering; for an overview of the MIKE-approach, see: ([AFL+93], [AFS96])

# 3 Integration of Machine-Learning techniques at the knowledge level

However, there are cases in which we want to tie together components of our "conventional" knowledge level descriptions with ML-techniques in order to perform tasks that either are not suited for normal knowledge acquisition (such as the analysation of large databases, etc.), or should be done instead of performing knowledge-acquisition because the same functionality and performance might be obtained automatically. This means we want to integrate these two sorts of components at the knowledge level in our MIKE-approach. However, when integrating ML-techniques at the knowledge level the point arises that on the one hand we have got the ("conventional") PSM-competence descriptions that (as they are meant to) describe problem solving knowledge independent from its later implementation, and on the other hand we have got to describe machine-learning algorithms that do represent certain (non-functional) design decisions as well[2].

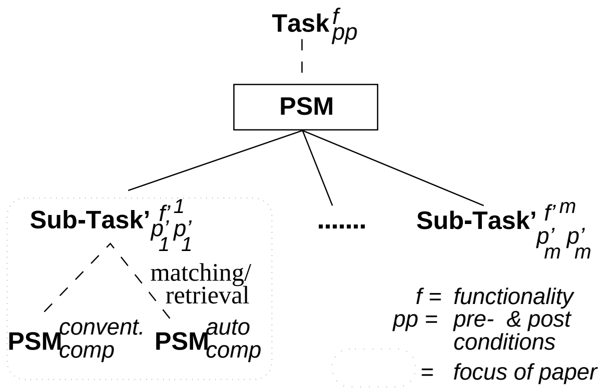


Figure 1: *Task-structures for flexible integration of ML-algorithms.*

So, the goal is rather to integrate ML-algorithms at the knowledge level than to use ML-algorithms for defining parts of an expert system. Sub-tasks (as long as they are generic) could be replaced by "conventional" PSM's or by $\text{PSM}^{auto}$ that describe ML-algorithms. In this case we are interested in the ability to define a task-structure in which ML-algorithms may be combined with "conventional" components. This is interesting for our purposes, since it gives us the possibility to define multi-strategy systems where we can select ML-algorithms using task-features and goal-descriptions (see figure 1).

In this case the task-goal is to be resolved assuming the usage of descriptions of the functionality of ML-algorithms at the knowledge level. For integrating these functionality-descriptions of ML-algorithms with the other PSM-functionality descriptions at the knowledge level we need a way to describe this functionality. In the next section we will propose a way in which such a description can be performed.

---

[2] A simple example of this is the notion of efficiency, something that up till now did not play a big role in knowledge level descriptions of problem-solving processes in knowledge acquisition. Some researchers argue that implicitly such kind of requirements are implemented nevertheless and argue therefore to make them explicit for PSM's as well ([FSvH96], [FS96]). In that case our point concerning functional and non-functional aspects of PSM's becomes less important.

# 4 Describing the functionality of Tasks and Problem Solving Components

## 4.1 Motivation

In the MIKE approach we regard the model of expertise as a "functional specification" of the knowledge based system to be built. This notion is quite different from specification in terms of software engineering. It is argued that we need a knowledge level description as a specification of the system incorporating more knowledge of *how* to achieve the functionality of the system than software engineering specification languages like Z [Spi92] or VDM [BFL+94] allow to formulate [Fen95]. In software engineering one wants to separate clearly *what* one wants to achieve from *how* to achieve this. To our mind this is a favourite starting point to think about reuse. The two kinds of specification are not mutually exclusive or in conflict. They enhance each other: the model of expertise is the specification of the functionality of the knowledge based system. But for describing and retrieving reusable components one needs a more abstract description of which tasks the components can accomplish. In the following we present a description for tasks and problem solving components in a software engineering fashion via pre- and postconditions.

On the one hand we use the notation for describing classes of tasks (or problem instances) schematically which are then instantiated during the requirements elicitation process. On the other hand we capture the functionality of the components and by this we enable a way of indexing the library of our components.

| requirements | | functional description |
|---|---|---|
| task | | component |
| $\text{PRE}_{task}$ | $(\Longrightarrow)$ | $\text{PRE}_{component}$ |
| $\text{POST}_{task}$ | $\Longleftarrow$ | $\text{POST}_{component}$ |

An overview of our framework is given in the figure above. The first arrow is bracketed because some of the preconditions of the task are hidden assumptions and no expert can or is willing to explain all the details we need. In the ideal case the user/expert explains the postconditions, and perhaps (s)he mentions some preconditions, but one cannot assume that these preconditions are complete. With this (preliminary) description one can start the retrieval process, i.e. to a given task T one has to find a component C with[3]:

$$POST_{C[generic\ term|domain\ specific\ term]} \Rightarrow POST_T$$

In the normal case the above required relation for the preconditions does not hold, instead one obtains a relation like this:

$$PRE_{C[generic\ term|domain\ specific\ term]} \Leftarrow PRE_T \wedge assumptions$$

This deviation could initiate two different processes: First one can try to validate the assumptions against the experts knowledge, i.e. the process of knowledge elicitation is triggered once again to make these assumptions explicit

[3] Actually, there also is a translation needed from the domain specific specification as extracted from the expert into the generic terms of the components. This translation is beyond the scope of this paper.

(illustrated below on the left hand side).

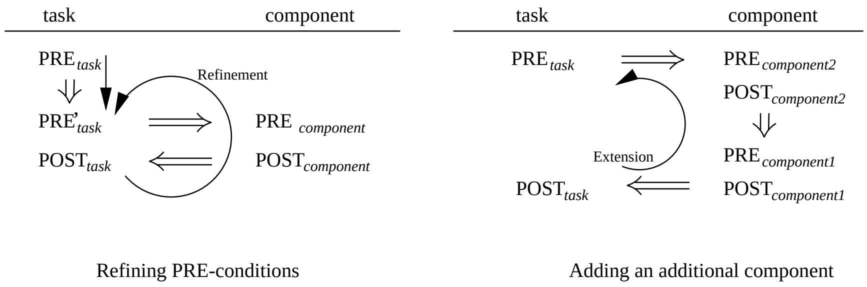


Figure 2: Refinement and Extension of Task-structures

And/or second - if this fails - one can set up the precondition of the component as a new goal to search a library. One can search for another component with a weaker precondition and a postcondition which implies the above mentioned precondition (illustrated above on the right hand side). In this way we can support (through definition of a task's functionality) the extension and refinement of a task structure that resolves a certain task-goal.

## 4.2 Capturing Requirements and Guiding Knowledge Elicitation

To structure the process of requirements elicitation we browse through a taxonomy of tasks (see figure 3).

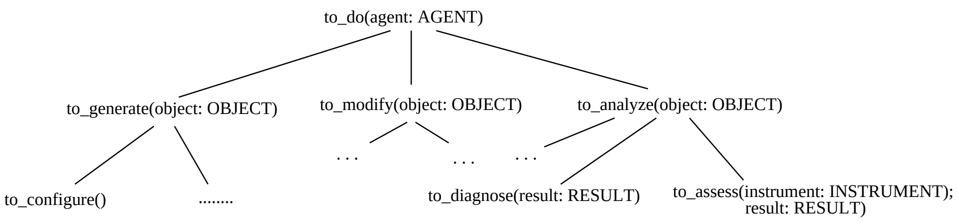


Figure 3: Taxonomy of Tasks (Taken from [BvdV94]).

We assume that tasks can be described best with verbs ([BvdV94], [SG95][4]) and to put emphasis on this aspect we use to-do-forms. We augment the nodes in the taxonomy with the semantic roles which have to (or can be) filled - according to the spirit of Fillmore's Case Grammar ([Fil68], [Fil77]). Fillmore proposes a finite set of semantic roles some of which are obligatory, others are facultative, and which can be used to describe what constituents a verb demands/allows and how the semantics of a sentence can be composed out of the constituents according to the role they play in the whole sentence[5].

[4] In [SG95] a similar framework for describing goals is used, however, the explicit notion of pre- and post-conditions is not taken into consideration there.

[5] Example: *The doctor diagnoses the patient's disease.*

(Fillmore proposes a notation independent from syntax which makes clear the semantic role a constituent plays in the semantics of the whole sentence.)

- semantic roles: to_diagnose(agent: doctor; object: patient; result: disease)

The links between the nodes of this taxonomy are really is_a links, so every node inherits all the information of his ancestors, i.e. all the verbs have a role named agent to fill, and every node of a lower level has a role named object to fill.

We choose the representation for two purposes. First, we believe that this helps to compare different approaches because one is really forced to explain what one is talking about, eg. using the verb "to-generate". And second, we want to avoid that the pre- and postconditions are formulated only over given terms; we want to ground the requirements on the experts utterances as far as possible.

We assume that the knowledge engineer in the modeling process can describe the (overall) task in one sentence or otherwise he can put it in a sequence of sentences which corresponds to a sequence of (sub-)tasks. In the latter case one can think about a suite (cf. [BvdV94]) and one can iterate the elicitation process over the subtasks; in the following we want to concentrate on the former case for reasons of simplification.

We reduce Fillmore's idea to a schematic description of the task which has to be instantiated during knowledge acquisition. This schematic description provides the vocabulary for the formulation of the pre- and postconditions. Similar to software engineering specification notations we can treat some elements as given if we are not interested in describing their internal structure. But if we are interested in the internal structure we expect every role to be filled with an object described by a set of attributes and a corresponding set of values; we describe the elements in these terms. This enables us to be very detailed in the formulation of the conditions and can be used for the domain knowledge acquisition process. Since it is not very satisfying only to enumerate that all the input must be provided and that we have a certain output, we have to make explicit all the interdependencies between all these elements which have to be realized by all instances of the specification - as it is usual in the software engineering notion of a specification.

Our notation is deeply influenced by the specification notation Z [Spi92]. We leave apart all the special features of Z concerning schemas; we take over the part of the grammar to formulate boolean expressions and expressions on sets. In addition we introduce some abbreviations: $attribute^{OBJECT}$ means a certain attribute of the object OBJECT, and $value_a^{OBJECT}$ the value of the attribute $a$ of the object OBJECT.

In the following we sketch how some high level task types can be described via pre- and postconditions.

The top level node "to_do" is specified with both conditions as true, i.e. that every component would be acceptable due to the precondition ($TRUE \Rightarrow X$ is always true) but no component will fit the postcondition since every component is aimed at fulfilling a certain goal and has a postcondition different from true.

to_generate(agent: AGENT; object: OBJECT)

/* AGENT generates OBJECT */
PRE:
$\exists a | a = attribute^{OBJECT} \bullet value_a^{OBJECT} = \perp$
POST:
$\forall a | a = attribute^{OBJECT} \exists value \bullet value_a^{OBJECT} = value$

"to_generate" means that in the situation before accomplishing the task there was at least one attribute of the object one wants to generate with an undefined value. The precondition of the "to_generate" task type is specified this way on purpose to express the difference to the "to_modify" task type since the "to_modify" task type presumes that all the attributes of the object one wants to modify have defined values. After accomplishing the generation or modification task again all the attributes of the modified object must have - potentially different - defined values.
The analyzing task expects that all the attributes have defined values. Of course after analyzing one will have a result that the postcondition should specify - but due to the variety of specialications of the analyzing task with different kinds of results it does not make sense to specify a sophisticated postcondition on this abstract level. This has to be done on the more concrete levels.

### 4.3 Integrating an ML-component for solving the assessment-task

We want to illustrate our point of view (i.e. describing and integrating PSM-components and ML-algorithms at the knowledge level) with an example.

Let us assume that an expert formulated requirements according to which the knowledge engineer classified his task as a "to_assess" task. The assessment-task (see figure 4) is used as defined in [BvdV94].

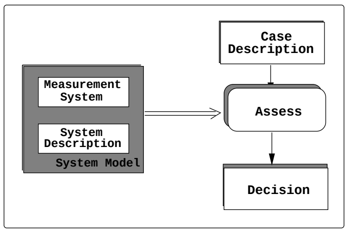


Figure 4: *Assessment in the CommonKADS-library [BvdV94].*

[VL94] define the assessment-task as:

**Definition:** Assessment is a problem type (task) in which a `case description` (input) is mapped onto a `decision` (output) according to a `system model`.

Where a `case description` contains a structured description of a case, a `decision` is described as a role that contains either a *decision class* or a *grade* that describes how well the decision is, according to the case description and the measurement system. The `system model` that is defined in the assessment model consists of two parts, namely a `system description` that can be seen as an abstraction of the data that form case-descriptions and the `measurement system`, on which to found `decisions`. The `system description` defines the "ontology"[6] on which an assessment-system is based. In the next part we want to give a more formal description of an assessment-task that posts certain pre- and postconditions on the structure that it is used in, and a classification subtask that fulfills certain requirements of the assessment-task (in this case: finding the

[6]With ontology we mean ontology as is it is usually used in knowledge acquisition, although we are aware of discrepancies with its original meaning in philosophy.

instrument respectively the measurement system to assess a case).

For this task we provide the following schema:

to_assess(agent: AGENT; object: OBJECT;
instrument: INSTRUMENT; result: RESULT)

/* AGENT assesses OBJECT with INSTRUMENT as RESULT */
PRE:
$INSTRUMENT : OBJECT \rightarrow RESULT$
POST:
$RESULT = INSTRUMENT(OBJECT)$

The elicitation process yielded the following description of the task as a partial instantiation of the schema.

to_assess$_{requirements}$(agent: expert; object: db_entry;
instrument: INSTRUMENT; result: class)

/* expert assesses db_entry with INSTRUMENT as class */
PRE:
$INSTRUMENT = \perp$
POST:
$class = INSTRUMENT(db\_entry)$

The expert was able to fill all the slots of the case frame but the instrument. So far we can substitute its AGENT by the expert, OBJECT by db_entry and RESULT by class.

Our library contains a component which has the capability to accomplish the "to_assess" task. It must be provided with a case-ontology and decisions. The component can be "applied" reasonably only if the intersection of both system-ontology and case-ontology is not empty. Furthermore the component needs a relation which offers certain decisions depending on the values of the attributes of an object (of the system_ontology). This relation should offer only one unique value which can be used as the DECISION. If the relation maps the values of the attributes to no or to more than one decision a theory revision has to be triggered and the DECISION has no defined value.

assessment$_{component}$(agent: AGENT; object: CASE;
instrument: CLAUSES; result: DECISION)

/* AGENT assesses CASE with CLAUSES as DECISION */
PRE:
$given\ ontology$
$given\ decisions$
$system\_ontology = \{attr | attr = attribute^{CASE}\}$
$(system\_ontology \cap case\_ontology) \neq \emptyset$
$INSTRUMENT \subseteq system\_ontology \times decisions$
POST:
$M = \{d \in decisions | (CASE, d) \in INSTRUMENT\}$
$((\#M \neq 1 \wedge DECISION = \perp \wedge theory\_revision)$

$\vee(\#M = 1 \wedge DECISION \in M))$

If we match our requirements of our task schema (see above) with this component we get the following result: The postcondition of the task schema corresponds to the component assessment,(with [AGENT | expert, CASE | db_entry, DECISION | class, CLAUSES | INSTRUMENT]) but the pre-condition requires an instrument to assess the case (i.e. a mapping as a special case of a relation) . This pre-condition is not fulfilled and may trigger two possibilities: either to elicitate this knowledge by hand, or to look for a component with the respective capability. Our library also contains ML algorithms. The description of the functionality of ID3 [Qui86] looks like the following.

$\mathrm{ID3}_{component}$(agent: AGENT; object: OBJECT; instrument: DECISION)

/* AGENT classifies OBJECT wrt DECISION */
PRE:
$given\ training\_set \subseteq OBJECT \times DECISION$
$(o_1, d_1) \in training\_set \wedge (o_1, d_2) \in training\_set \Rightarrow d_1 = d_2$
POST:
$ontology = \{attr | o \in OBJECT \wedge attr = attribute^o\}$
$CLAUSES =$
$\{cl | o \in OBJECT \wedge d \in DECISION \wedge x \subseteq ontology \wedge$
$cl = (\bigwedge_{a \in x} P_a(value_a^o) \Rightarrow d) \wedge (o, d) \in training\_set \wedge$
$\neg \exists y | y \subset x \bullet (\bigwedge_{a \in y} P_a(value_a^o) \Rightarrow d)\}$

Given a training set which is subset of the cartesian product of objects and decisions this component offers a postcondition we looked for. The component ID3 learns a set of clauses, i.e. conjunctions (over a minimal set of attributes) of predicates which imply a certain decision for certain values of these attributes. We use predicate variables $P_a$ to express that we might have different predicates for the different attributes. A predicate describes a property which must hold for the value of an object's attribute, e.g. that this value is equal or greater or less than a constant expression (an instance of such a clause would look like $COLOUR = red \wedge AGE > 2 \Rightarrow problem\_class34$).

To use this set of clauses as input of the assessment component we have to define finally a mapping like the following:

$$\begin{aligned}(obj, d) \in INSTRUMENT \quad \Leftrightarrow \quad & \exists cl \in CLAUSES \bullet \\ & x = \{attr \mid attr = attribute^{obj}\} \\ & \wedge\, cl = (\bigwedge_{a \in x} P_a(value_a^{obj}) \Rightarrow d)\end{aligned}$$

Since the pre-condition of ID3 is fulfilled with this additional definition, we can combine the two components and together they realize the functionality of the desired system. So we achieved what the expert wanted even with the very sketchy requirements and without further knowledge elicitation.

## 4.4 Strategic aspects

In the "conventional" way of using the assessment PSM one uses the `measurement-system` as the key-role upon which to base a decision. The `measurement-`

`system` in this view forms an assumption of the assessment-PSM, which means that one needs to define such a system before one can use this PSM. In our case of integrating ML-algorithms at the knowledge-level we want to be able to redefine the pre- and postconditions of this PSM, and use the *case-description* and *decision* as preconditions and creating this `measurement-system` automatically (which is precisely what an ML-algorithm can perform). In such a case we can use the same PSM (namely assessment) for different purposes by simply redefining the pre- and postconditions of a PSM. When we describe our tasks according to figure 2, we get two possibilities to use it for selecting PSM-components. At the end, our framework does provide a way to describe functionality of PSM-components (whether one is interested in their internal structure or not is not the point). In the next section we want to relate this to related work performed for both describing functionalities of PSM's and ML-algorithms.

Describing the competence of knowledge level components in terms of their pre- and postconditions as presented above, gives us the opportunity to match inferences in processes according to their pre- and postconditions. It also enables a possibility to support the integration of ML-techniques in a task-decomposition when ML-techniques are described in the same framework as we propose for PSM's. The integration of these techniques at the knowledge level then plays a role in the user-guidance module as defined for multi-strategy learning systems.

# 5 Related work

As far as related research is relevant for our approach, or at least has some features that makes it significant for us to deal with, this section will provide a short discussion of them.

In [AP94] the idea of integration of learning techniques for solving impasses in inference-processes is introduced. Differences between the approach of [AP94] and our approach are that we do not see impasses as the trigger for learning, we try to build up a task-decomposition instead, given the functionality-descriptions of several machine-learning techniques. In our framework the learning is not meant to "bridge" impasses, but to use knowledge level descriptions to provide support for the use of machine learning techniques when decomposing complex tasks.

In [WJ95] an analysis of the underlying ontological commitments and assumptions is made. They show these assumptions for a few problem solving components used to model the VT task and also stress the need for making explicit the assumptions and commitments that underly problem-solving methods.

Another approach we want to relate to is the approach of [RA94]. This work is performed in the same philosophy as ours. However, where we want to provide a framework for integrating ML-algorithm "building blocks" with the normal PSM-building blocks settings the approach of [RA94] concentrates upon the description of ML-algorithms themselves. The goal we have in mind in our framework is to describe machine learning algorithms on a high-level abstraction so we can integrate them with the task-decompositions we want to generate for complex tasks. A description of the internals of machine learning techniques is only of limited importance to us, namely as far as it helps us to support a domain-expert to decompose his task and find appropriate ML-techniques to perform them (and get support to apply those techniques). The same holds for the approach of [Slo94] (although highly informal) where algorithms are also described on a very low level of detail and lack a formal description of their pre-and postconditions.

An approach that has a similar functionality, but deals with KA-algorithms instead, is the approach of Generalized Directive Models ([vH95]). In a GDM a task is described and an inference structure is generated that performs this task using a simple but effective task-grammar. In our case a description at the same level would be necessary in order for our tool to be able to select algorithms and integrate them with PSM-components.

A few approaches that describe (the need for) task-features are known to us. In [ABD+93] a more or less natural language description of features is given in the framework of the *CommonKADS*-approach [BvdV94]. These features are not formally described and (as far as we know) never used for describing the functionality of machine learning techniques.

One of the formal oriented approaches is presented in [Abe95]. What Aben did for inference actions we provide on a higher level for tasks and problem-solving components.

There is no real consensus about which features of tasks and ML-algorithms have to be described, although there are a few proposals that contain such listings of features ([KMG94], [vS95]). These listings are not very detailed and not stabilized at the moment.

With regard to the representation of problem descriptions that initiate the generation of a task-decomposition, there is to mention the approach of [PG96]. The research mentioned is directed towards the formal definition of goals, where [PG96] describe what we see as a problem description, rather than goals as such.

Finally there is the work of the MLT consortium [Con93] where multiple learning techniques were integrated. Especially the work on Consultant [CSG+92] is to be mentioned here. This work uses a kind of ML-algorithm description to select algorithms, but does not do that in an explicit manner. Comparing this approach with ours, we see that one of the differences is that the level of integration of these learning algorithms was not so strong, and the system assumed a good task definition to be explicitly known by the user. At the same time the system was meant to select one out of a set of ML-algorithms and there, we feel, our approach has simply another perspective since we want to integrate several techniques according to task-decompositional frameworks.

# 6 Conclusion

The aim of our paper is to present a formalism in which one can define the functionality of PSM's at the knowledge level. We propose to do so using formalized descriptions of the pre- and postconditions of such PSM's. When such a functionality is defined, an opportunity arises to extend these descriptions to the field of learning systems. Such a system can in general be very helpful when defining and refining a task-decomposition structure. In many knowledge acquisition projects making explicit the goals of an application and finding an appropriate task-decomposition for them is not as straightforward as it often seems to be. In our experience experts often need more than one try to come to a good problem definition. What makes our approach interesting for further research in the direction of multi-strategy learning systems is this process of defining the actual problem (problem definition in our framework), mapping goal descriptions on task-structures while providing a structured way of describing the whole process, and the combination of defining tasks with or without (depending upon goals and context-givens) learning capabilities in one system.

So, in the first place we have the possibility of extending our system easily with new ML-algorithms, if required, since we have a uniform, component-based representation of ML-algorithms. Secondly we can also deal with situations where learning is NOT the most ideal solution for solving a problem (i.e. in situations were a non automated expert system can perform better due to data constraints[7]). Furthermore we want to enable user support based upon the task definition as it is represented by the user and offer a framework to capture the requirements of the expert/user. We enable structuring the knowledge elicitation process and provide an explicit representation of the features of the special task (according to the framework of the task type).

Our future research will be concerned with the question which "semantic slots" one can and should include in the description of the functionality of tasks, and integrate these in the MIKE-approach. A new version of the KARL language (NewKARL) will be defined in order to provide the formal basics. An important topic will be the integration of a library and usage of the component-descriptions used in this paper as a guidance for component-selection. Future work also consists of the usage of these ideas for defining a multi-strategy learning system that can flexibly support a user defining his or her task decompositions and supporting the execution of such a system using the MIKE-approach.

### Acknowledgements

We thank Rudi Studer and the other members of our research team for their inspiration for and their comments on the ideas presented in this paper.

## References

[ABD+93] A. Aamodt, B. Benus, C. Duursma, Chr. Tomlinson, R. Schrooten, and W. v.d. Velde. Task Features and their Use in CommonKADS. Deliverable 1.5, version 1.0, Consortium, 1993.

[Abe95] M. Aben. *Formal Methods in Knowledge Engineering*. PhD thesis, University of Amsterdam, 1995.

[AFL+93] J. Angele, D. Fensel, D. Landes, S. Neubert, and R. Studer. Model based and Incremental Knowledge Engineering: The MIKE Approach. In J. Cuena, editor, *Knowledge Oriented Software Design*, volume A-27, pages 139 – 168, Amsterdam, 1993. IFIP Transactions, North Holland.

[AFS96] J. Angele, D. Fensel, and R. Studer. 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.*, Geneva, May 1996.

[Ang93] J. Angele. *Operationalisierung des Modells der Expertise mit KARL.* infix 53, St. Augustin, 1993. In German.

[AP94] J. L. Arcos and E. Plaza. Integration of Learning into a knowledge modelling framework. In L. Steels, G. Schreiber, and W v.d. Velde, editors, *A Future for Knowledge Acquisition: Proceedings of the 8th European Knowledge Acquisition Workshop*, volume 867 of *Lecture Notes in Artificial Intelligence*, pages 355 – 373. Springer Verlag, September 1994.

[BFL+94] J. C. Bicarrequi, J. S. Fitzgerald, P. A. Lindsey, R. Moore, and B. Ritchie. *Proof in VDM: A Practitioner's Guide.* Springer Verlag, Berlin, 1994.

[BvdV94] J. Breuker and W. van de Velde. *CommonKADS Library for Expertise Modelling.* IOS Press, 1994.

[Cha86] B. Chandrasekaran. Generic tasks in knowledge-based reasoning: High level building blocks for expert system design. *IEEE Expert*, 1, 1986.

[7] One can think of situations where statistics can do less due to data-characteristics, or situations where more knowledge is in the expert as in their databases(!)

[Cha88] B. Chandrasekaran. Generic Tasks as building blocks for knowledge-based systems: The diagnosis and routine design examples. *The Knowledge Engineering Review*, 3(3):183–210, 1988.

[Con93] MLT Consortium. Final public report. Technical report, 1993. Esprit II Project 2154.

[CSG+92] S. Craw, D. Sleeman, N. Granger, M. Rissakis, and S. Sharma. CONSULTANT: Providing Advice for the Machine Learning Toolbox. In M.A. Bramer and R.W. Milne, editors, *Research and Development in Expert Systems*, pages 5–23, 1992.

[Fen95] D. Fensel. *The Knowledge and Representation Language KARL*. Kluwer, Boston, 1995.

[Fil68] Ch. J. Fillmore. The case for case. In E. Bach and R. T. Harms, editors, *Universals in Linguistic Theory*, New York, 1968. Holt, Rinehart & Winston.

[Fil77] Ch. J. Fillmore. The case for case reopened. In P. Cole and J.L. Morgan, editors, *Grammatical relations*, pages 59 – 82, New York, 1977.

[FS96] D. Fensel and R. Straatman. Problem-Solving Methods: Making Assumptions for Efficiency Reasons. In *Proceedings of the 9th European Knowledge Acquisition Workshop, Nottingham, England*, Berlin, May, 14-17 1996. Lecture Notes in Artificial Intelligence, Springer-Verlag.

[FSvH96] D. Fensel, R. Straatman, and F. van Harmelen. The Mincer Methaphor for Problem-Solving Methods: Making Assumptions for Reasons of Efficiency. In Chr. Pierret-Golbreich, E. Motta, D. Fensel, and M. Willems, editors, *Proceedings of the Knowledge Engineering-Methods & Languages Workshop (KEML-95)*, Paris, January 15-16 1996.

[KMG94] Y. Kodratoff, V. Moustakis, and N. Graner. Can Machine Learning solve my problem? *Applied Artificial Intelligence*, 8:1–31, 1994.

[Lan95] D. Landes. *Die Entwurfsphase in MIKE (The Design Stage in MIKE)*. infix 84, St. Augustin, 1995. In German.

[LS95] D. Landes and R. Studer. The Treatment of Non-Functional Requirements in MIKE. In W. Schäfer and P. Botella, editors, *Proceedings of the 5th European Software Engineering Conference ESEC '95*, Sitges, Spain, September 25-28 1995. Springer-Verlag, Berlin. Lecture Notes in Computer Science (989).

[Mar88] S. Marcus. *Automating Knowledge Acquisition for Expert Systems*. Kluwer, Boston, 1988.

[McD88] J. McDermott. Preliminary Steps towards a Taxonomy of Problem-Solving Methods. *In: [?]*, 1988.

[Neu93] S. Neubert. Model construction in MIKE (Model-Based and Incremental Knowledge Engineering). In *Current Trends in Knowledge Acquisition, 7th European Knowledge Acquisition Workshop*, pages 200–219, Berlin, September 1993. Toulouse, France, Springer Verlag.

[Neu94] S. Neubert. *Modellkonstruktion in MIKE; Methoden und Werkzeuge*. infix 60, St. Augustin, 1994.

[PG96] Chr. Pierret-Goldbreich. Modular and Reusable Specifications in Knowledge Engineering: Formal Specification of Goals and their Development. In Chr. Pierret-Goldbreich, D. Fensel, E. Motta, and Mark Willems, editors, *Proceedings of the 6th Workshop on Knowledge Engineering Methods and Languages*, Paris, januari 15-16, 1996, 1996.

[Qui86] J. R. Quinlan. Induction of Decision Trees. *Machine Learning*, 1:81 – 106, 1986.

[RA94] C. Rouveirol and P. Albert. Knowledge level model of a configurable Learning System. In L. Steel, G. Schreiber, and W.v.d. Velde, editors, *A future for Knowledge Acquisition, 8th European Knowledge Acquisition Workshop, Belgium*, pages 374 – 393, 1994. Lecture Notes in Artificial Intelligence.

[SG95] B. Swartout and Y. Gill. EXPECT: Explicit Representation for Flexible Acquisition. In B.R. Gaines and M. Musen, editors, *Proceedings of the 9th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, volume 2, February 26- March 3 1995.

[Slo94] A. Slodzian. Configuring decision tree learning algorithms with KresT. In *Workshop Proceedings of the Workshop on Knowledge level models of machine learning; ECML '94*, 1994. Available from: "ftp://arti.vub.ac.be/pub/krest/appkits/learnkit/learnkit.ps.Z".

[Spi92] J. M. Spivey. *The Z Notation: A Reference Manual*. Prentice Hall, New Jersey, 2nd edition, 1992.

[Ste90] L. Steels. Components of Expertise. *AI Magazine*, 1990.

[vH95] G. van Heijst. *The Role of Ontologies in Knowledge Engineering*. PhD thesis, University of Amsterdam, 1995.

[VL94] A. Valente and Ch. Löckenhoff. Assessment. In *[BvdV94]*, pages 155 – 174. IOS-Press, 1994.

[vS95] M. van Someren. PhD thesis, University of Amsterdam, 1995. To appear.

[WJ95] B.J. Wielinga and J.M. A Formal Analysis of Parametric Design Problem Solving. In B.R. Gaines and M. Musen, editors, *Proceedings of the 9th Banff Knowledge Acquisition for Knowleddge Based Systems Workshop*, volume 2, pages 37/1–37/15, Banff, 1995.

[WSB92] B.J. Wielinga, A.T. Schreiber, and J.A. Breuker. KADS: A modelling approach to knowledge engineering. Special Issue "The KADS approach to knowledge engineering". *Knowledge Acquisition*, 4(1):5–53, 1992.