Requirements Specification and Model-based Knowledge Engineering

Jürgen Angele and Rudi Studer
Institut für Angewandte Informatik und
Formale Beschreibungsverfahren
Universität Karlsruhe (TH)
76128 Karlsruhe
e-mail: {angele | studer}@aifb.uni-karlsruhe.de

Abstract

Knowledge Engineering (KE) and Software Engineering (SE) have similar goals: developing methods, techniques, and tools for the building process of either knowledge based systems (kbs) or (complex) traditional software. Due to this kinship it seems obvious to analyse to which extent both areas, KE and SE, can benefit from each other. In this paper we want to argue from the KE point of view. The main paradigm in KE has switched from a transfer point of view to a modeling point of view. So a lot of emphasis has been put on the development of notions and methods for building and structuring models, which capture different results of the development process. Much effort has been investigated into the analysis and development of reusable components: problem solving methods describing the dynamic behaviour of kbs and ontologies defining the vocabulary and structure of (domain) models. We describe specific approaches in KE in some detail which exploit these ideas in different ways: Role-Limiting Methods, KADS and MIKE. Finally Life Cycle Models, Non Functional Requirements and Transformational Development are discussed as areas where we think KE might profit from research and experiences made in SE.

Key Words: Requirements analysis, knowledge engineering, knowledge acquisition, reuse

1 Introduction

In earlier days in AI there was no interest in methodological issues. Instead the research activities were focused on the development of formalisms, languages, inference mechanisms, and tools to describe and operationalize knowledge based systems and on the realization of small knowledge based systems in order to study the feasibility of the different approaches.

Though these studies offered rather promising results, the transfer of this technology into commercial use in order to build e.g. large expert systems failed in many cases. The situation was directly comparable to a similar situation in the construction of traditional software systems, called „software crisis“: the means to develop small academic prototypes were not sufficient for the design and maintenance of large, long living commercial systems. In the same way as the „software crisis“ brought the area of Software Engineering to birth the similar situation in AI brought the area of Knowledge Engineering to birth.

So the goal of KE is similar to that of SE: turning the process of building kbs from an art to an engineering discipline. This requires the analysis of the building and maintenance process itself and the development of appropriate methods, languages, and tools, which support these processes.

Because both disciplines, SE and KE, have similar roots and goals it is quite obvious that there are many common interests and that it may be fruitful for both areas to learn from each other.

Within this learning process the first question which arises is: what are the main differences between a „traditional“ software system, i.e. a system which has to be tackled by SE methods, and a kbs?

In [Par86] the concept „incompletely specified functions“ is used in order to characterize problems, which need a kbs to be solved. Typically kbs are developed to solve very complex problems or even problems which are not entirely understood [ShG92]. So the functionality of the system may only be partially specified in advance. Instead such a specification must be developed iteratively in cooperation with expert(s) and the user(s) of the system. In many cases the solution of such problems may only be called adequate or inadequate instead of correct or incorrect.

Even if the functionality of a kbs is entirely specifiable the complexity of every computational mechanism may be so high, that the problem in its entire generality may only be solved for small instances. Many AI-problems in their general formulation are NP-hard problems [Neb95]. Experts solve such problems by using a large amount of domain specific knowledge, which allows to restrict the problem, to approximate the problem or to reformulate the problem in order to solve a simpler problem efficiently or to use domain specific heuristics which allow to reduce the average complexity [Neb95]. So for such problems it is not sufficient to give a detailed functional specification and to build a solution using „normal“ computer science know-how. Instead domain specific knowledge is necessary in order to be able to build a solution which is able to solve larger instances of the problem in a reasonable time.

> *„In simple terms this means analysis is not simply interested in what happens, as in conventional systems, but with how and why.“[Bro86].*

The last difference between „traditional“ systems and kbs is a rather technical point. In kbs most knowledge is represented explicitly, whereas in „traditional“ systems a high amount of knowledge is compiled into the algorithms. In domains where this knowledge changes rapidly (e.g. configuring computer systems) the explicit representation has especially advantages for the maintenance of the system.

Nevertheless we don´t think there is a sharp border line which clearly allows to classify systems as either to be a „traditional“ system or to be a kbs. In contrast we think that both mark different points on a scale and that the transition between both is fluent.

In this paper we present methods from the area of KE, which we think may be useful for SE, and we will outline deficiencies in the actual methods of KE, where we think KE could learn from SE.

The paper is structured as follows. In section 2 principles and methods from KE, which could be beneficial for SE as well, are presented. Some well-known approaches of KE, incorporating these ideas are described in section 3. Some areas, in which in our opinion KE could profit from SE, are discussed in section 4. Finally, in section 5, we summarize and draw a conclusion.

2 Knowledge Engineering: Principles and Methods

In the following some of the main principles and methods which influence the existing KE approaches are presented.

2.1 KE as a Modeling Process

> *„This transfer and transformation of problem-solving expertise from a knowledge source to a program is the heart of the expert-system development process." [HWL83]*

In earlier days the development of a kbs has been seen as a transfer process of knowledge from the head of the expert into the computer system (Figure 1). It was assumed that the knowledge in the kbs is best structured in the same way as in the expert´s head. Nowadays this process of building a kbs is seen as a *modeling activity*. Building a kbs means building a computer model with the aim of imitating an expert´s approach. It is not intended to create a cognitive adequate model, i.e. to simulate the cognitive processes of an expert in general, but to create a model which offers similar results in problem-solving for problems in the area of concern. While the expert may consciously articulate some parts of his or her knowledge, he or she will not be aware of a significant part of this knowledge since it is hidden in his or her skills. This knowledge is not directly accessible by interviews but only by observing the expert and interpreting these observations. Therefore this knowledge acquisition process is no longer seen as a transfer of knowledge into an appropriate computer representation, but as a generation of a model of the knowledge by the observer ([Cla89], [Mor91]).

Fig. 1 Transfer View

This also reflects the two different roles of the model. On the one hand, the model is a result of the knowledge-generating process. It is gained by observing and interpreting the expert's behaviour. On the other hand, its role becomes a precondition for the further process. The model consists of a number of hypotheses about the experts behaviour, which have to be validated by additional observations. In this case, the already existing model serves as a basis for gathering and interpreting new observations.

This modeling view of the building process of a kbs has the following consequences:

- Like every model, such a model cannot reflect reality in general, but is only valid for a concrete purpose, i.e. it can only *approximate* the desired behaviour. In principle, the modeling process is infinite, because it is an incessant activity with the aim of approximating the model to the intended behaviour.
- The modeling process is a *cyclic* process. New observations may lead to a refinement, modification, or completion of the already built-up model. On the other side the model may guide the further acquisition of knowledge.

- The modeling process is dependent on the interpretation of the observations made by the observer. Therefore this process is very *faulty* and a feedback by reality in the cyclic process is indispensable for the creation of an adequate model. According to this feedback the model must therefore be revisable in every stage of the modeling process.

2.2 Problem Solving Methods

In [Cla85] Clancey analysed a set of first generation expert systems developed to solve different tasks, e.g. diagnosis. Though they were realized using different representation formalisms (e.g. production rules, frames, lisp) he discovered a common problem solving behaviour. Clancey was able to abstract this common behaviour to a generic inference pattern called Heuristic Classification, which describes the dynamic behaviour of these systems on an abstract level, the so called „knowledge level" [New82]. This level allows to describe reasoning in terms of goals to be achieved, actions necessary to achieve these goals and knowledge needed to perform these actions. A knowledge-level description abstracts from details concerned with the implementation of the system. A similar separation is done in software engineering where the description at the knowledge level corresponds to the specification of the system, which is clearly separated from its design and implementation.

Let us consider Heuristic classification in some more detail (Figure 2). First observables are abstracted maybe over several levels to abstract observables, e.g. the observable „41^0 C temperature" may be abstracted to „high temperature". These abstracted observables are matched to a solution class, e.g. „infection", and finally the solution class is hierarchically refined to a solution, e.g. the disease „influenca".

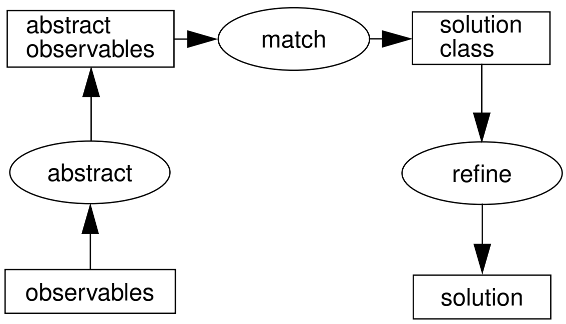

Fig. 2 Heuristic Classification

It turned out that Heuristic Classification may be used for specific tasks independently of the domain at hand. For instance it may be used to diagnose diseases in the same way as it may be used to diagnose faults in cars.

In the meantime a lot more such generic inference patterns, called *problem-solving methods (PSM)*, have been identified: Cover-and-Differentiate for diagnosis [Mar88], Propose-and-Revise [Mar88a] for parametric design, Skeletal-Plan-Refinement for hierarchical planning etc. A description and taxonomy of different PSMs may be found in [Pup93],[McD88].

PSMs may be exploited in the knowledge engineering process in different ways:

- PSMs imply a classification of tasks. An experienced knowledge engineer is able to select an appropriate PSM for the task to be solved. So PSMs are an important concept for the way knowledge engineers think and thus for educating knowledge engineers.

For this reason a set of semi-formally described PSMs together with the tasks they are applicable to is described in [BrV94]. For different kinds of diagnosis PSMs are described in [Ben93].

- PSMs contain actions which need specific knowledge in order to perform their task. For instance, Heuristic Classification needs a hierarchical classification of observables and solutions. A PSM generically describes the different *roles* which are played by different kinds of domain knowledge during the problem solving process. For instance the role „observables" may be instantiated by „patient data". So a PSM may be used as a guideline to acquire static domain knowledge and additional domain specific problem-solving knowledge like heuristics and constraints. Based on this idea powerful knowledge acquisition tools may be developed which allow to acquire exactly the knowledge needed by the PSM. In many cases the acquisition of domain knowledge and the creation of the PSM is an incremental process: First a PSM for the top-level task is selected. This guides the acquisition of additional knowledge which in turn leads to the selection of PSMs for the sub-tasks etc. [THW+93].
- A PSM allows to describe the main rationale of the reasoning process of a kbs. First, this supports the validation of the kbs, because the expert is able to understand the problem solving process. Second, it may be used for documentation purposes and thus eases the maintenance process. Third, this abstract information may be used during the problem solving process itself for explanation facilities.
- The problem solving process itself may be kept flexible by dynamically choosing an appropriate PSM for the actually to be solved (sub-)task [CJS92]. So for every possible (sub-)task the system designer describes a set of alternative PSMs, which solve this subtask. Dependent on application conditions the best PSM for the case at hand is chosen during the problem solving process.
- A PSM may be a component to be reused in different systems. Thus it is not left only to the experience of the knowledge engineer, which PSM may be able to solve the task at hand. Instead a library of PSMs described in an appropriate representation formalism should be available in order to support technical reuse of PSMs. PSMs may be reused on different levels. Reusing PSMs described at the knowledge level resembles reusing specification components in SE. Reusing PSMs at the code level is comparable to the idea of source-code libraries in SE [Kru92]. In the same way as in SE reuse allows to reduce the modeling and implementation effort and improves the quality (in the sense of software quality criteria) of the resulting kbs. Some work is in progress to identify the assumptions and limitations of a PSM [Fen95a]. This knowledge is necessary to match features of the task to features of the PSM and thus to support the selection process of PSMs. In future this may perhaps enable the partial automation of the selection process.
- PSMs (or combinations of PSMs) may be used as a skeletal description of the design model of the kbs. If the kbs is realized preserving this structure, the final implementation of the kbs reflects this conceptual structure.

The concept problem-solving method has strongly stimulated research in knowledge engineering and thus has influenced a lot of approaches in this area. It has resulted in so-called Role-Limiting Methods [McD88] (see section 3.1) and in libraries of PSMs (see e.g. [BrV94]).

For PSMs different levels of generality may be distinguished. The above mentioned PSMs are domain independent but specific for a certain type of task. This means that they are able to perform a specific type of task (e.g. diagnosis) in different domains. An instantiation of such a

PSM, i.e. the application in and adaptation to a specific domain, leads to a domain specific PSM (e.g. diagnosis of infectious diseases). In SE the notion of PSM like it is described above is not discussed yet. Nevertheless standardized software products have some properties with PSMs in common. For instance a standardized module for the administration of a stock may be seen as a domain specific PSM.

2.3 Ontologies

In addition to problem solving methods the notion of an *ontology* provides a further type of reusable model component. Whereas the meaning of a PSM has reached a rather large agreement in the KE community, the meaning of the term ontology is still defined in rather different ways. At least two basic types of ontologies may be distinguished:

- *Commonsense Ontologies*: Commonsense ontologies aim at defining a collection of top-level concepts and associated relations which are independent of a specific application domain. „...they are intended to be comprehensive - to be a ´conceptual coat rack´ on which to hang more specific ontologies and domain knowledge." [Gru93]. In order to make the development of these ontologies manageable, they are typically divided into different clusters, e.g. for time and events, space, or qualities and quantities (compare [HeR91] or [LeG90]). E.g. a time cluster could alternatively provide a notion of time which is based on time points or time intervals. Since such clusters are built on a sound theoretical basis, their reuse within a requirements modeling process results in well structured and robust model components and thus avoids the need for inventing such clusters on the fly. [Pir93] reports on the advantages of such an approach in the context of building up a model of business process management. Of course, appropriate methods are required for supporting the reuse of commonsense ontologies and for avoiding the well-known disadvantages of ´code scavenging´ [Kru92]. In [PiS94] the KARO approach is introduced which offers an integrated set of formal, lexical, and graphical methods for reusing commonsense ontologies (see [Pir95] for details).
- *Domain Ontologies*: Domain ontologies provide a conceptualization of a specific domain [Gru93]: they constrain the structure and contents of particular domain knowledge and define the vocabulary, which is used to model the domain. In that way, domain ontologies may be compared with conceptual schema descriptions of data bases. However, ontologies are typically specified in more expressive languages [vHe95]. Domain ontologies aim at making domain knowledge shareable across different tasks and reusable for different tasks (see [NFF+91]. Again that bears similarities with the problem of making databases interoperable.

In [SWD+94] and [SWJ95] a proposal is made to organize ontologies into layers, where each layer expresses a *meta-level* view on another layer. Thus a task specific ontology, e.g. for diagnosis, may be defined on top of a domain ontology, e.g. for a medical domain, and a PSM specific ontology, e.g. for heuristic classification, may in turn be specified on top of the task specific one.

A slightly different approach is described in [ToA94] for engineering applications. There, the notion of *ontological viewpoints* is introduced: ontologies are used to provide different, but well related views on the modeled systems and to make ontological assumptions of these different views explicit. [ToA94] proposes three ontologies to structure a library of reusable models for physical systems: a component, a process, and an information ontology.

Although ontologies, as discussed in the KE community, are defined differently and with different purposes, they all aim at making some kind of knowledge shareable and reusable in different contexts.

2.4 Integration of Informal and Formal Description Formalisms

A major concern within KE is the development of methods and description formalisms which support the process of building up an adequate and explicit model of the required expertise (see section 2.1). Since the participation of the expert in this modeling process is of high importance, emphasis has been put on constructing well structured and informally or semi-formally described conceptual models which can be used as a communication basis with the expert. The Componential Framework [Ste93] with its various models for describing tasks, methods, and domain knowledge is a notable example of this development. However, in contrast to SE where the development of informal specification languages (see e.g. Structured Analysis [You89]) and formal specification languages (see e.g. Z [Spi92]) has been done to a large extent independently from each other, executable and/or formal specification languages in KE exploit to a large extent the structure of such conceptual models. Typical examples of this tight integration of the structure of a conceptual model into the definition of a formal specification language are $(ML)^2$ [vHB92] and KARL ([Ang93], [Fen93]) (see sections 3.2 and 3.3). As a result, a smooth transition may be achieved between the informal conceptual model and the formal specification of the kbs [FLN+94].

3 Specific Approaches in KE

The ideas presented in the previous section have had influence on a lot of approaches in KE. Some of them are described in the following section.

3.1 Role-Limiting Methods

Role-Limiting Methods (RLM) ([Mar88], [McD88]) have been one of the first attempts to support the development of expert systems by exploiting the notion of a reusable problem solving method. The RLM approach may be characterized as a shell approach. Such a shell comes with an implementation of a specific problem solving method and thus can only be used to solve a predefined type of tasks. The given problem-solving method also defines the generic roles that knowledge can play during the problem-solving process and it completely fixes the knowledge representation for the roles such that the expert only has to instantiate the generic concepts and relationships, which are defined by these roles.

Let us consider as an example the PSM Heuristic Classification (see Figure 2). A RLM based on Heuristic Classification offers a role „observable” to the expert. Using that role the expert (i) has to specify which domain specific concept corresponds to that role, e.g. „patient data” (see Figure 4), and (ii) has to provide domain instances for that concept, e.g. concrete facts about patients.

It is important to see that a description of a problem solving method is given in a generic, domain independent way. Since the kind of knowledge, which is used by the problem solving method, is predefined, the acquisition of the required domain specific instances may be supported by (graphical) interfaces which are custom-tailored for the given problem solving method.

Two RLMs will now be described in some more detail: SALT ([Mar88a]) which is used for solving constructive tasks and MED2/CLASSIKA ([GaP92], [Pup93]) which is used for solving classification tasks.

SALT

SALT [Mar88] is a knowledge acquisition tool for generating expert systems which use the PSM Propose-and-Revise. Thus expert systems may be constructed for solving specific types of design tasks, e.g. parametric design tasks. The basic inference actions Propose-and-Revise is composed of may be characterized as follows (see [McD88] for more details):

- (i) extend a partial design by proposing a value for a not yet computed design parameter,
- (ii) determine whether all computed parameters fulfil the relevant constraints, and
- (iii) apply fixes to remove constraint violations.

In essence three generic roles may be identified for Propose-and-Revise ([Mar88]):

- (i) „design-extensions” refers to knowledge for proposing a new value for a design parameter,
- (ii) „constraints” provides constraint knowledge restricting the admissible values for parameters, and
- (iii) „fixes” makes potential remedies available for specific constraint violations.

From that characterization of the PSM Propose-and-Revise one can easily see that only generic PSM specific terms are used which are totally domain independent. Thus the PSM may be used for solving design tasks in different domains by specifying the required domain knowledge for the different predefined generic knowledge roles.

E.g. when SALT was used for building VT [MSM88], an expert system for configuring elevators, the domain expert used the form-oriented user interface of SALT for entering e.g. domain specific design extensions (see Figure 3). I.e. the generic terminology of the knowledge roles is instantiated by VT specific instances.

1	Name:	CAR-JAMB-RETURN
2	Precondition:	DOOR-OPENING = CENTER
3	Procedure:	CALCULATION
4	Formula:	[PLATFORM-WIDTH - OPENING-WIDTH] / 2
5	Justification:	CENTER-OPENING DOORS LOOK BEST WHEN CENTERED ON PLATFORM.

Fig. 3 Design Extension Knowledge for VT

(the value of the design parameter CAR-JUMB-RETURN is calculated according to the formula - in case the precondition is fulfilled; the justification gives a description why this parameter value is preferred over other values (example taken from [Mar88a]))

MED2 / CLASSIKA

MED2 ([GaP92], [Pup93]) is an expert system shell for solving classification tasks. MED2 relies on a Heuristic Classification approach [Cla85]: the identification of diagnoses is based on the observation of symptoms and on rules describing which observations indicate which diagnoses. By using evidence factors these rules also include a specification of the strength of the relationship between an observation and a diagnosis. An example of such a rule in the domain of fault-finding in automobile motors would read as follows: „If the fuel consumption interpretation is slightly too high, then rate the diagnosis 'air filter foal' with evidence 60%" ([GaP92]). Typically, observations are first abstracted to abstract observations before they are used in such heuristic rules. Furthermore, diagnoses which are proposed as solutions have to be evaluated and in the end refined to more specific diagnoses.

From the description of the PSM as used in MED2 it becomes evident, that few knowledge roles such as symptoms or diagnoses are sufficient for characterizing the types of knowledge which are handled within MED2. These predefined knowledge roles are used within the MED2 knowledge acquisition tool CLASSIKA ([GaP92]) to generate appropriate knowledge acquisition editors. For that purpose CLASSIKA includes predefined graphical structures ([GPS93]) and associated editors, e.g. hierarchies, tables and forms. In that way diagnoses and symptoms are acquired by filling in domain specific diagnoses (e.g. „fuel tank empty" [GaP92]) and symptoms (e.g. „not intended engine stop" [GaP92]) using the given hierarchy editors, whereas heuristic rules are acquired by using the given table editor (see [GaP92] for details). By offering these various knowledge acquisition editors and by exploiting the predefined knowledge roles of heuristic classification, knowledge bases can be developed and maintained to some extent by the domain experts themselves.

On the one hand, the predefined knowledge roles and thus the predefined structure of the knowledge base may be used as a guideline for the knowledge acquisition process: it is clearly specified what kind of knowledge has to be provided by the domain expert. On the other hand, in most real-life situations the problem arises of how to determine whether a given domain specific task may be solved by a given RLM. This task analysis is still a crucial problem, since up to now there does not exist a well-defined collection of features for characterizing a domain task in a way which would allow a straightforward mapping to appropriate RLMs. In addition, if a given application task may only be solved by combining different PSMs, the fixed structure of RLMs does not provide a good basis for handling that configuration problem.

In order to overcome this inflexibility of RLMs the concept of configurable RLMs has been proposed.

Configurable Role-Limiting Methods

Configurable Role-Limiting Methods (CRLM) as proposed in [PoG93] exploit the idea that a complex PSM may be decomposed into several subtasks where each of these subtasks may be solved by different methods (see e.g. [CJS92]). In [PoG93] various PSMs for solving classification tasks, like Heuristic Classification or Set-covering Classification, have been analysed with respect to common subtasks. This analysis resulted in the identification of shared subtasks like „data abstraction" or „hypothesis generation and test". Within the CRLM framework a predefined set of different methods are offered for solving each of these subtasks. Thus a PSM may be configured by selecting a method for each of the identified subtasks. In that way the CRLM approach provides means for configuring the shell for different types of tasks. It should be noted that each method offered for solving a specific subtask has to meet the knowledge role specifications given for the CRLM shell, i.e. the CRLM shell comes with a fixed scheme of knowledge types. As a consequence, the introduction of a new method into the shell typically involves the modification and/or extension of the current scheme of knowledge types [PoG93]. Having a fixed scheme of knowledge types and predefined communication paths between the various components is an important restriction distinguishing the CRLM framework from the flexible configuration approaches like e.g. KADS (see section 3.2) or PROTEGE-II [PET+92].

It should be clear that the introduction of such a flexibility into the RLM approach removes one of its disadvantages by still exploiting the advantage of having a fixed scheme of knowledge types, which build the basis for generating effective knowledge-acquisition tools. On the other hand, configuring a CRLM shell increases the burden for the system developer since he has to have the knowledge and the ability to configure the shell in the right way.

Generation of Knowledge-Acquisition Tools

Obviously, a big effort is required for implementing a (graphical) knowledge acquisition tool for all the specific RLMs. Therefore, some proposals have been made to generate such a tool from the predefined knowledge representation scheme of the shell. META-KA [Gap95] is such a tool generator, which uses the specification of the knowledge representation together with some additional information concerning the screen representation for generating various types of graphical editors. In essence, editors for forms, hierarchies, graphs, and tables may be generated.

Another approach is the PROTEGE-II approach ([PET+92], [EPM94]), which aims at supporting the development of expert systems by reusing domain and problem solving knowledge as well as by providing a meta-level tool, called DASH, to generate (domain-specific) knowledge-acquisition tools. DASH uses an ontology - specified in a frame like language - as input and generates a forms oriented tool for instantiating the concepts and relationships defined in the ontology. DASH also generates a basic dialogue structure and provides means for customizing the generated tool to the specific needs and preferences of the user.

RLMs and their various generalizations build a good basis for offering knowledge-acquisition tools that allow domain specialists to provide themselves much of the domain knowledge required [EPM94]. Thus, domain specialists may actively participate in the development process of expert systems, although they are typically non-programmers.

3.2 The KADS Approach

One of the most prominent knowledge engineering approaches is the *KADS approach* ([SWB93], [WSB92]) and its further development to *CommonKADS* [SWD+94]. A basic characteristic of KADS is the construction of a collection of models, where each model captures specific aspects of the system to be developed as well as of its environment. In CommonKADS the *organization model*, *the task model*, the *agent model*, the *communication model*, the *expertise model*, and *the design model* are distinguished. Whereas the first four models aim at modeling the organizational environment the kbs will operate in as well as the tasks that are performed in the organization, the expertise and design model describe functional aspects of the kbs under development.

A major contribution of the KADS approach is its proposal for structuring the expertise model. This knowledge-level model distinguishes different types of knowledge to describe the knowledge which is required to solve a given task. Basically, static domain knowledge and two types of knowledge which constitute the PSM specification are organized into different layers of the expertise model[1] [SWB93] (see Figure 4):

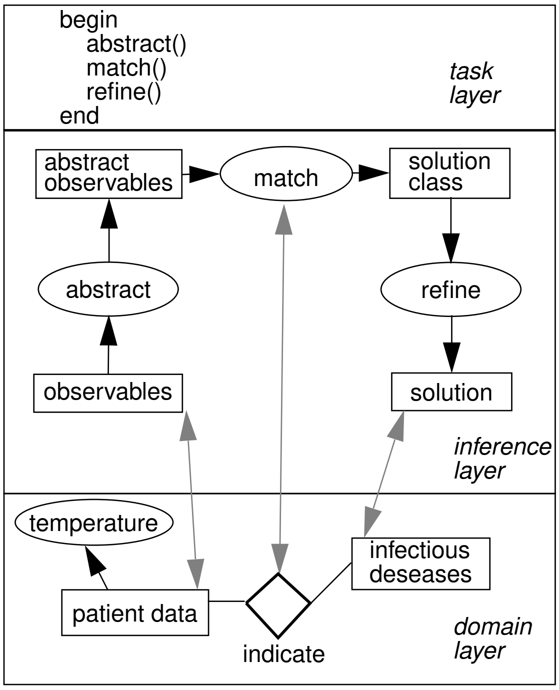

Fig. 4 Expertise model

- *Domain layer*: On the domain layer all the domain specific knowledge is modeled which is needed to solve the task at hand. This includes a conceptualization of the domain, a so-called ontology [SWD+94], and a declarative theory of the required domain knowledge. One objective for the construction of the domain layer is to model the domain layer as independent as possible from the way it will be used later on for solving specific tasks. However, this objective is restricted by the so-called *relative interaction hypothesis* [SWD+94], which assumes that some kind of interaction exists between the structure of the domain knowledge and the given task.
- *Inference layer*: On the inference layer the *inference actions* the generic PSM is composed of as well as the *roles*, which are played by the domain knowledge within the PSM, are described. The dependencies between inference actions and roles are specified in what is called an inference structure. In that way the inference layer restricts the set of all possible inferences to those which are defined as inference actions. In Figure 4 we see the inference structure for the PSM Heuristic Classification. Furthermore, the notion of roles provides a domain independent view on the domain knowledge. E.g. „patient data" plays the role of „observables" within the inference structure of Heuristic Classification (see Figure 4).
- *Task layer*: The task layer provides means for specifying the control over the inference actions, which are defined on the inference layer. Thus, the task layer specifies the

1. We do not consider the *strategic layer* which should provide knowledge for selecting the best task structure for achieving a given goal.

control flow of the PSM. Basically, the task layer provides a decomposition of tasks into subtasks and inference actions including a goal specification for each task, and a specification of how these goals are achieved.

Two types of languages are offered to describe an expertise model: CML (Conceptual Modeling Language) [SWA+94], which is a semi-formal language with a graphic notation, and $(ML)^2$ [vHB92], which is a formal specification language based on first order predicate logic and dynamic logic. Whereas CML is oriented towards providing a communication basis between the knowledge engineer and the domain expert, $(ML)^2$ is oriented towards validating a constructed model through (partial) simulation.

The clear separation of the domain specific knowledge from the generic description of the PSM on the inference and task layer enables in principle two kinds of reuse: on the one hand, a domain layer description may be reused for solving different tasks by different PSMs, on the other hand a given PSM may be reused in a different domain by defining a new view to another domain layer. Of course, in both cases the interaction hypothesis may make necessary some modifications of the given model structures. Within CommonKADS a library of reusable and configurable components, which can be used to build up expertise models, has been defined [BrV94]. The library covers nine different types of problems, among others diagnosis, assessment, planning, and configuration.

In essence, the expertise model and the communication model capture the functional requirements for the target system. Based on these requirements the design model is developed, which specifies among others the system architecture and the computational mechanisms for realizing the inference actions. KADS aims at achieving a *structure-preserving design*, i.e. the structure of the design model should reflect the structure of the expertise model as far as possible [SWD+94].

All the development activities, which result in a stepwise construction of the different models, are embedded in a cyclic and risk-driven life cycle model (see [Tay92]) similar to Boehm's spiral model [Boe88].

The basic structure of the expertise model has some similarities with the data, functional, and control view of a system as known from software engineering. However, a major difference may be seen between an inference layer and a typical data-flow diagram (compare [You89]): Whereas an inference layer is specified in generic terms and provides - via roles and domain views - a flexible connection to the data described on the domain layer, a data-flow diagram is completely specified in domain specific terms. Moreover, the data dictionary does not correspond to the domain layer, since the domain layer may provide a complete model of the domain at hand which is only partially used by the inference layer, whereas the data dictionary is describing exactly those data which are used to specify the data flow within the data flow diagram (see also [FAL+93]).

3.3 The MIKE Approach

MIKE (Model-based and Incremental Knowledge Engineering) [AFL+93] defines an engineering framework for eliciting, interpreting, formalizing, and implementing knowledge in order to build kbs. It aims at integrating the advantages of life cycle models, prototyping, and formal specification techniques into a coherent framework for the knowledge engineering process. Subsequently, we will discuss the main principles and methods of MIKE.

In contrast to other approaches which assume that the expert creates the model himself, it is assumed that the knowledge engineer is the moderator of this modeling process. This is necessary due to the following reasons:

- Often the knowledge about the problem-solving process becomes conscious to the expert only during the communication with the knowledge engineer.
- The approach that the expert completes the model of expertise by filling facts of the domain into the system is only applicable for simple domains or by using predefined shells which are specialized to one problem-solving method or one application domain.
- Often the expert is overstrained in learning to use acquisition tools or representation formalisms in order to model the expertise himself.

Considering knowledge engineering as a modeling activity implies that this process is *cyclic*, *faulty* and *approximative*.

Within the MIKE approach these properties have been considered as the main reasons in order to develop methods and techniques which provide feedback for the knowledge engineer as early as possible within the modeling process. Therefore prototyping of the acquired expertise using executable models has been integrated into the modeling process as one of its main features.

Within the modeling process a large gap has to be bridged between informal descriptions of the expertise which have been gained from the expert using knowledge elicitation methods and the final realization of the kbs. Dividing this gap into smaller ones reduces the complexity of the whole modeling process because in every step different aspects may be considered independently from other aspects.

The knowledge gained from the expert in the elicitation phase is described in natural language. It mainly consists of interview protocols, protocols of verbal reports, etc. These knowledge protocols define the *elicitation model* [Neu94](Figure 5). This knowledge represented in natural language must be interpreted and structured. The result of this step is described semi-formally in the so-called *structure model* [Neu94], using predefined types of nodes and links.

According to the KADS approach the knowledge-level description of the functionality of the system is given in the *model of expertise* (cf. [SWB93]). For describing the model of expertise the formal and operational specification language KARL ([Ang93], [Fen93]) has been developed. KARL is based on first order logic and dynamic logic and offers language primitives for each of the three different layers of the model of expertise.

The model of expertise finally includes all functional requirements of the desired system. For the realization of the final system, additional requirements have to be considered which are still independent of the final implementation of the system. These requirements are non-functional requirements such as efficiency of the problem-solving method, maintainability of the system, persistency of data etc. Capturing such decisions within the *design model* divides the gap between the model of expertise and the implementation of the final system. For the description of the design model the language KARL has been extended to the language DesignKARL [Lan94] which allows to describe data structures, algorithms and offers additional structuring primitives like clusters and modules.

The different representations of knowledge, i.e. elicitation model, structure model, model of expertise, design model and the final system either represent the same knowledge in a different way or contain additional knowledge which is closely related to other knowledge in another representation. To gain the full benefits for documentation, maintenance, and explainability these different models are interrelated explicitly. Thus traceability of the system development process is achieved.

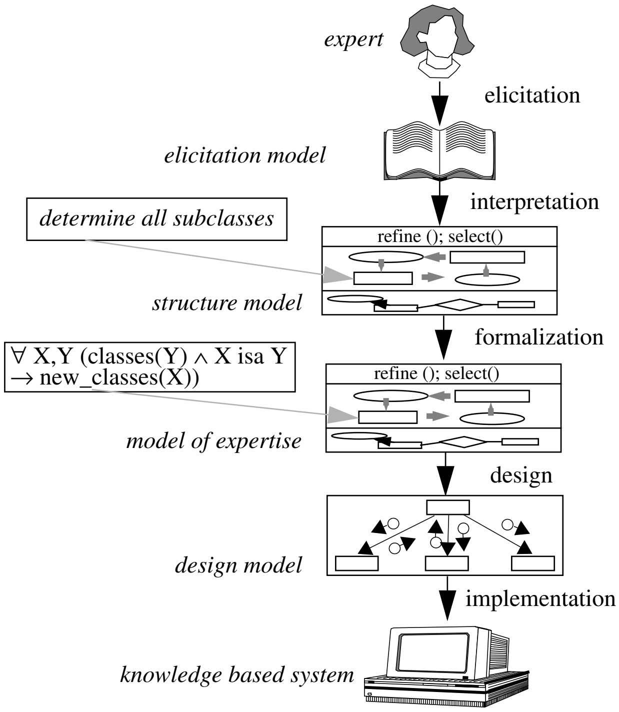

Fig. 5 Representation levels in MIKE

Using the MIKE approach it has been started to build up a library of formally specified problem solving methods: Hill Climbing [AFL+92], Chronological Backtracking [Fen93], Beam Search [Fen93], Cover-and-Differentiate [Ang92], Board-game Method [FEM+93], Propose-and-Exchange [LFA93], and Propose-and-Revise [PFL+94]. In [Fen95a] limitations and assumptions of the PSM Propose-and-Revise have been analysed which is a precondition to match features of a given task to appropriate PSMs.

The different knowledge representations are results of different steps of the building process. For this process a spiral process model (cf. [Boe88]) has been defined (see [Ang93], [Neu94]), which describes the different activities, their resulting models and the sequence of the activities within the whole building process. Due to the above mentioned properties of this modeling process explorative and experimental prototyping have been integrated [Flo84] as a main feature.

For the design process the language DesignKARL additionally allows to describe the design process itself and to describe interactions between design decisions. Thus, the design process is documented and the maintainability of the final kbs is improved.

4 Exploitable Results from SE

Although KE has developed a lot of methods in order to improve the process of building kbs, a lot remains to be done. Due to the kinship of both areas it seems obvious, that a lot of methods which have resulted from research and experience in SE, may be adopted in KE. A small choice of such methods, from our subjective point of view, are presented in the following.

4.1 Life Cycle Models

In earlier days the prevailing paradigm for developing kbs was *rapid prototyping*. The acquired knowledge was immediately operationalized using an operational representation formalism. The running program delivered immediate feed-back for validation purposes. Based on this feed-back the represented model was modified and revised. Rapid prototyping fits well to the transfer view of knowledge acquisition: no intermediate steps like specification and design are necessary, because it is assumed that the knowledge is represented in the computer similarly to the knowledge in the experts head. This has severe disadvantages [Ang93]:

- The knowledge engineer is overstrained by simultaneously eliciting, interpreting, and structuring knowledge, as well as designing and implementing the system. Thus he has to consider the whole complexity of the system and of the development process at the same time.
- Different aspects of knowledge, like implementation aspects, functional requirements, non-functional requirements etc. have to be considered simultaneously and as a consequence are mixed inseparable in the final system.
- The functionality of the system is extended from a small functionality at the beginning to the final functionality at the end. Thus the architecture of the system is not planned in advance but evolves over time.
- The running system is the only representation of the knowledge. The reader of such a representation gets lost in detail. This representation is not understandable and is thus not suitable for documentation or explanation purposes.
- The represented knowledge is determined and limited by the used implementation formalism.
- Because of the lack of defined mile stones the progress of the project is difficult to control by the project management.

At the beginning of the KADS project a variant of the waterfall model has been adopted. This model represents the diametrical opposite to rapid prototyping. But as it has been already recognized a long time ago in SE, this model is not very well suited for the construction of complex software and in the same sense it is not very well suited for building kbs.

So later on, advanced process models from SE like the Spiral Model [Boe88] have been adopted from SE and adapted to the special needs in KE (see [Ang93], [Tay92]).

The research for suitable process models and the experience in the application of these models in SE in the past have strongly influenced corresponding activities in KE and we think they will influence them in future.

4.2 Non-Functional Requirements and Design Rationales

Up to now, the treatment of non-functional requirements has almost been neglected in KE. This is in contrast to the software engineering and the information systems engineering community, which have proposed several approaches for handling non-functional requirements (see e.g [MCN92]) and for capturing the rationale for design decisions (see e.g. [Lee91] and [RoP94]). When considering the types of non-functional requirements, which are discussed in these approaches or summarized in [IEEE93], it is obvious that these categories are to a large extent relevant categories for the development of kbs as well [LaS95]. Obviously, some categories like understandability may be more important for kbs, whereas other categories like accuracy [MCN92] may be more important for information systems.

In KE proposals for handling non-functional requirements and building up design rationales have been made only rather recently. When looking at the models, which are proposed in [StM95] and [Van95] for capturing design decisions, we can see that rather similar model structures are proposed in both communities. This holds for the approach taken in MIKE ([Lan95], [LaS95]) as well. However, within MIKE more emphasis is put on supporting the process of eliciting and structuring non-functional requirements and on relating elementary design decisions to the modeled requirements.

4.3 Transformational development

The development of a new PSM from scratch may be seen as a transformational process [Fen95b]. This process starts with the definition of a PSM Generate-and-Test for the task at hand. Generate-and-Test contains two subtasks: Enumerate all solutions in the generate step and test the generated solutions against constraints which describe valid solutions. Starting from this very general PSM and taking characteristics of the task and / or domain into account a more efficient PSM is developed in a transformational process.

For this kind of transformational development there has been a lot of research in SE (cf. [PaS83], [Smi90]). E.g. in [Smi90] an efficient algorithm for the k-queens problem is developed in the following steps: choose a global search strategy to enumerate solutions, simplify expressions, apply partial evaluation and specialization, use finite differencing, perform case analysis and finally refine data. Each step is guaranteed to be consistent with the original functional specification. Such a transformational approach may also be exploited in the design phase of a kbs for transforming a declarative specification of the functionality of the kbs into a more algorithmic one.

5 Conclusion

During the last ten years, the knowledge engineering community has put a lot of emphasis on developing notions and methods for building up and structuring (reusable) models.

The development of formal specification languages which are based on a highly structured conceptual models provides a sound framework for supporting a stepwise transition from informal system specifications to formal ones. This approach is obviously interesting for SE as well and is reflected in developments in the context of VDM [LPT93] or TROLL [JWH+94].

A major achievement in KE is the development of the notion of a reusable generic problem solving method, i.e. the identification of generic structures to describe the problem solving behaviour of a kbs. Obviously, the notion of a problem solving method can not be directly transferred into the field of software engineering. Nevertheless, it could be worthwhile to investigate, to which extent the development and maintenance process of software systems could benefit from the identification of reusable structures, which describe the functional behaviour in a similar way as problem solving methods specify the behaviour of kbs.

The construction of domain models as part of the requirements analysis activities has gained a lot of interest in recent years [JBR+93]. As a consequence, the principles and methods for building up domain models, which have been developed in KE, are of relevance for the requirements engineering process as well. E.g. the notion of reusable domain abstractions [SuM94] bears a lot of similarities with libraries of reusable model components, as e.g. described in [BrV94]. Especially, the granularity problem and the problem of matching the characteristics of an application task with features of stored models are also subject of current research in KE. Here, a mutual exchange of ideas and experiences could be beneficial for both communities.

On the other hand, since KE is faced with all the problems, which are involved in the development of software systems, as well, KE can clearly profit from the methods, which result from research and experiences in the software engineering community. In addition to the topics, which have been discussed in section 4, we think e.g. of methods for system design and of methods for verifying system properties.

Acknowledgement

Several discussions with Dieter Fensel and Dieter Landes about the relationship between knowledge and software engineering are gratefully acknowledged. Rainer Perkuhn provided valuable comments to a preliminary version of the paper.

References

[AFL+92] J. Angele, D. Fensel, D. Landes, and R. Studer: An Assignment Problem in Sisyphus - No Problem with KARL. In: M. Linster (ed.): Sisyphus ´91: Models of Problem Solving, Arbeitspapiere der GMD, no 630, March 1992.

[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 (ed.), *Knowledge Oriented Software Design, IFIP Transactions A-27*, North Holland, Amsterdam, 1993, 139-168.

[Ang92] J. Angele: Cover and Differentiate Remodeled in KARL. In: Proceedings of the 2nd KADS User Meeting, Munich, February 17-18, 1992, C. Bauer et al. (eds.), Interpretation Models for KADS - Proceedings of the 2nd KADS User Meeting (KUM´92), GMD report, no 212, 1992.

[Ang93] J. Angele: Operationalisierung des Modells der Expertise mit KARL (Operationalization of the Model of Expertise with KARL), Ph.D. Theses in Artificial Intelligence, No. 53 , infix, St. Augustin, 1993 (in German).

[Ben93] V.R. Benjamins: *Problem Solving Methods for Diagnosis*. Ph. D. Thesis, University of Amsterdam, Amsterdam, June 1993.

[Boe88] B.W. Boehm: A Spiral Model of Software Development and Enhancement. In: *Computer 21*, 5 (May 1988), 61-72.

[Bro86] A. G. Brooking: The Analysis Phase in Development of Knowledge-Based Systems. In: W. A. Gale (ed.): *AI and Statistic*, Addison-Wesley Publishing Company, 1986, 321-334.

[BrV94] J.A. Breuker and W. Van de Velde (eds.). *The CommonKADS Library for Expertise Modeling*. IOS Press, Amsterdam, 1994.

[CJS92] B. Chandrasekaran, T.R. Johnson, and J.W. Smith. Task-Structure Analysis for Knowledge Modeling. In: *Communications of the ACM, 3*5(9), 1992, 124-137.

[Cla85] W.J. Clancey: Heuristic Classification. In: *Artificial Intelligence, 27*, 1985, 289-350.

[Cla89] W.J. Clancey: The Knowledge Level Reinterpreted: Modeling How Systems Interact. In: *Machine Learning 4*, 1989, 285-291.

[DKS93] J.-M. David, J.-P. Krivine, and R. Simmons: *Second Generation Expert Systems*, Springer-Verlag, Berlin, 1993.

[EPM94] H. Eriksson, A. R. Puerta, and M. A. Musen: Generation of Knowledge Acquisition Tools from Domain Ontologies. In: *Int. J. Human-Computer Studies* 41, 1994, 425-453.

[FAL+93] D. Fensel, J. Angele, D. Landes, and R. Studer: Giving Structured Analysis Techniques a Formal and Operational Semantics with KARL. In: Züllighoven et al. (eds): *Requirements Engineering '93: Prototyping*, Teubner Verlag, Stuttgart, 1993.

[FEM+93] D. Fensel, H. Eriksson, M. A. Musen, and R. Studer: Description and Formalization of Problem-Solving Methods for Reusability: A Case Study. In: *Complement Proceedings of the 7th European Knowledge Acquisition Workshop (EKAW´93)*, Toulouse, France, September 6-10, 1993.

[Fen93] D. Fensel: *The Knowledge Acquisition and Representation Language KARL*, Ph. D. thesis, University of Karlsruhe, 1993. Kluwer Academic Publisher, Boston, 1995.

[Fen95a] D. Fensel: A Case Study on Assumptions and Limitations of a Problem Solving Method. In: *Proceedings of the 9th Banff Knowledge Acquisition for Knowledge-Based System Workshop (KAW´95), Banff, Canada,* 1995.

[Fen95b] D. Fensel: The Mincer Metaphor: A New View on Problem-Solving Methods for Knowledge-Based Systems? Department SWI, University of Amsterdam, 1995.

[FLN+94] D. Fensel, D. Landes, S. Neubert, and R. Studer: Integrating Semiformal and Formal Methods in Knowledge-Based Systems Development. In: *Proc. 3rd Japanese Knowledge Acquisition for Knowledge-Based Systems Workshop (JKAW'94)*, Hotoyama, 1994.

[Flo84] C. Floyd: A Systematic Look at Prototyping. In: R. Budde et al. (eds.), *Approaches to Prototyping*, Springer-Verlag, Berlin, 1984, 1-18.

[Gap95] U. Gappa: *Grafische Wissensakquisitionssysteme und ihre Generierung*, Ph.D. Theses in Artificial Intelligence, No. 100, infix, St. Augustin, 1995 (in German).

[GaP92] U. Gappa and K. Poeck: Common Ground and Differences of the KADS and the Strong Problem-Solving-Shell Approach. In: Th. Wetter et al. (eds): *Current Developments in Knowledge Acquisition, Proc. 6th European Knowledge Acquisition Workshop (EKAW`92)*, Heidelberg/ Kaiserslautern, May 1992, Lecture Notes in Artificial Intelligence 599, Springer Verlag, 1993.

[GPS93] U. Gappa, F. Puppe, and S. Schewe: Graphical Knowledge Acquisition for Medical Diagnostic Expert Systems. In: *Artificial Intelligence in Medicine 5*, Special Issue "Knowledge Acquisition", 1993, 185-211.

[Gru93] T.R. Gruber: A Translation Approach to Portable Ontology Specifications. In: *Knowledge Acquisition 5*, 2, 1993, 199-221.

[HWL83] F. Hayes-Roth, D.A. Waterman, and D.B. Lenat. *Building Expert Systems*, New York, Addison-Wesley, 1983

[HeR91] O. Herzog and C.R. Rollinger (eds): *Text Understanding in LILOG*. Lecture Notes in Artificial Intelligence 546, Springer-Verlag, 1991.

[IEEE93] IEEE Computer Society: *IEEE Standard for a Software Qualitiy Metrics Methodology*. IEEE Std. 1061-1992, 1993.

[JBR+93] M. Jarke, J. Bubenko, C. Rolland, A. Sutcliffe, and Y. Vassiliou: Theories Underlying Requirements Engineering: An Overview of NATURE at Genesis. In: *Proc. IEEE Int. Symposium on Requirements Engineering*, San Diego, 1993.

[JWH+94] R. Jungclaus, R. Wieringa, P. Hartel, G. Saake, and T. Hartmann: Combining TROLL with the Object Modeling Technique. In: Wolfinger (ed): *Innovationen bei Rechen- und Kommunikationssystemen, GI-Fachgespräch: Integration von semiformalen und formalen Methoden für die Spezifikation von Software*, Hamburg, Springer-Verlag, 1994.

[Kru92] C. W. Krueger: Software Reuse. In: *ACM Computing Surveys, 24*, 2, Junie 1992, 131-184.

[Lan94] D. Landes: DesignKARL - A language for the design of knowledge-based systems. In: *Proc. 6th International Conference on Software Engineering and Knowledge Engineering SEKE'94*, Jurmala, Lettland, 1994, 78-85.

[Lan95] D. Landes: *Die Entwurfsphase in MIKE - Methode und Beschreibungssprache (The Design Phase in MIKE - Method and Description Language)*. Ph.D. Theses in Artificial Intelligence, No. 84, infix, St. Augustin, 1995 (in German).

[LaS95] D. Landes and R. Studer: The Treatment of Non-Functional Requirements in MIKE. In: *Proc. of the 5th European Software Engineering Conference (ESEC´95)*, Sitges, 1995.

[Lee91] J. Lee: Extending the Potts and Bruns Model for Recording Design Rationale. In: *Proc. of the 13th Int. Conf. on Software Engineering*, Austin, 1991, 114-125.

[LeG90] D. Lenat and R.V. Guha: *Building Large Knowledge-Based Systems: Representation and Inference in the CYC Project*. Addison-Wesley Publ. Co., 1990.

[LFA93] D. Landes, D. Fensel, and J. Angele: Formalizing and Operationalizing a Design Task with KARL.

In: J. Treur and Th. Wetter (eds.): *Formal Specification of Complex Reasoning Systems*, Ellis Horwood, New York, 1993.

[LPT93] P. G. Larsen, N. Plat, and H. Toetenel: A Formal Semantics of Data Flow Diagrams, In: *Formal Aspects of Computing*, Vol. 3, 1993.

[Mar88] S. Marcus (ed.): *Automating Knowledge Acquisition for Experts Systems*. Kluwer Academic Publisher, Boston, 1988.

[Mar88a] S. Marcus: SALT: A Knowledge Acquisition Tool for Propose-and-Revise Systems. In: [Mar88].

[MCN92] J. Mylopoulos, L. Chung, and B. Nixon: Representing and Using Non-Functional Requirements: A Process-Oriented Approach. In: *IEEE Trans. on Software Engineering 18*, 6, 1992, 483-497.

[McD88] J. McDermott: Preliminary Steps toward a Taxonomy of Problem-solving Methods. In: [Mar88], 225-255.

[Mor91] K. Morik: Underlying Assumptions of Knowledge Acquisition as a Process of Model Refinement. In: Knowledge Acquisition 2, 1, March 1990, 21-49.

[MSM88] S. Marcus, J. Stout, and J. McDermott: VT: An Expert Elevator Configurer that Uses Knowledge-based Backtracking. In: *AI Magazine* 9 (1), 1988, 95-112.

[Neb95] B. Nebel: Artificial Intelligence: A Computational Perspective. To appear in: G. Brewka (ed.): Essentials in Knowledge Representation.

[New82] A. Newell: The knowledge level. In: *Artificial Intelligence 18*, 1982, 87-127.

[Neu94] S. Neubert: *Modellkonstruktion in MIKE - Methoden und Werkzeuge*. Ph.D. Theses in Artificial Intelligence, No. 60, infix, St. Augustin, 1994 (in German).

[NFF+91] R. Neches, R.E. Fikes, T. Finin, T.R. Gruber, R. Patil, T. Senator, and W.R. Swartout: Enabling Technology for Knowledge Sharing. In: *AI Magazine 12*, 3, 1991, 16-36.

[PaS83] H. Partsch and R. Steinbrüggen: Program Transformation Systems. In: *ACM Computing Surveys 15,* 3, September 1983, 199-236.

[Par86] D. Partridge: *Artificial Intelligence. Application In The Future Of Software Engineering*. Ellis Horwood Limited, Großbritanien, 1986.

[PET+92] A. R. Puerta, J. W. Egar, S. W. Tu, and M. A. Musen: A Multiple-Method Knowledge Acquisition Shell for the Automatic Generation of Knowledge Acquisition Tools. In: *Knowledge Acquisition* 4, 1992, 171-196.

[PFL+94] K. Poeck, D. Fensel, D. Landes, and J. Angele: Combining KARL and Configurable Role Limiting Methods for Configuring Elevator Systems. In: *Proceedings of the 8th Banff Knowledge Acquisition for Knowledge-Based System Workshop (KAW´94)*, Banff, Canada, Januar 30th - February 4th, 1994.

[Pir93] Th. Pirlein: Reusing a Large Domain-Independent Knowledge Base. In: *Proc. 5th Int. Conf. on Software Engineering and Knowledge Engineering*, 1993.

[Pir95] Th. Pirlein: Wiederverwendung von Commonsense Ontologien im Knowledge Engineering: Methoden und Werkzeuge. Ph.D. Thesis in Artificial Intelligence, No. 85, infix, St. Augustin, 1995 (in German).

[PiS94] Th. Pirlein and R. Studer: KARO: An Integrated Environment for Reusing Ontologies. In: Steels et al. (eds): *A Future of Knowledge Acquisition, Proc. 8th European Knowledge Acquisition Workshop (EKAW´94)*, Hoegaarden, Lecture Notes in Artificial Intelligence 867, Springer-Verlag, 1994.

[PoG93] K. Poeck and U. Gappa: Making Role-Limiting Shells More Flexible. In: N. Aussenac et al. (eds): *Knowledge Acquisition for Knowledge-Based Systems, Proc. 7th European Knowledge Acquisition Workshop (EKAW`´93)* Toulouse, September 1993, Lecture Notes in Artificial Intelligence 723, Springer Verlag, 1993.

[Pup93] F. Puppe: *Systematic Introduction to Expert Systems: Knowledge Representation and Problem-Solving Methods*. Springer Verlag, Berlin, 1993.

[RoP94] C. Rolland and N. Prakash: Tracing the Evolution of Artifacts. In: Karagiannis (ed): *Database and Expert System Applications*, Lecture Notes in Computer Science 856, Springer-Verlag, 1994.

[ShG92] M.L.G. Shaw and B.R. Gaines: The Synthesis of Knowledge Engineering and Software Engineering. In: P. Loucopoulos (ed.): *Advanced Information Systems Engineering*, LNCS 593, 1992, 208-220.

[Smi90] D.R. Smith: KIDS - a Semi-automatic Program Development System. In: *IEEE Transactions on Software Engineering. Special Issue on Formal Methods in Software Engineering 16*, 9, September

1990, 1024-1043.

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

[StM95] A. Stutt and E. Motta: Recording Design Decisions of Knowledge Engineers to Facilitate Reuse of Design Models. In: Gaines et al. (eds): *Proc. of the 9th Knowledge Acquisition for Knowledge-Based Systems Workshop (KAW´95)*, Banff, SRDG Publ., Univ. of Calgary, 1995.

[Ste93] L. Steels: The Componential Framework and its Role in Reusability. In: David et al. (eds): *Second Generation Expert Systems*, Springer-Verlag, Berlin, 1993.

[SuM94] A.G. Sutcliffe and N.A.M. Maiden: Domain Modeling for Reuse. In: *Proc. 3rd Int. Conf. on Software Reuse*, Rio de Janeiro, 1994.

[SWA+94] A.Th. Schreiber, B. Wielinga, H. Akkermans, W. van de Velde, and A. Anjewierden: CML: The CommonKADS Conceptual Modeling Language. In: Steels et al. (eds): *A Future of Knowledge Acquisition, Proc. 8th European Knowledge Acquisition Workshop (EKAW´94)*, Hoegaarden, Lecture Notes in Artificial Intelligence 867, Springer-Verlag, 1994.

[SWB93] G. Schreiber, B. Wielinga, and J. Breuker (eds.): *KADS. A Principled Approach to Knowledge-Based System Development*, Knowledge-Based Systems, vol 11, Academic Press, London, 1993.

[SWD+94] A.Th. Schreiber, B.J. Wielinga, R. de Hoog, H. Akkermans, and W. van de Velde: CommonKADS: A Comprehensive Methodology for KBS Development. In: *IEEE Expert*, December 1994, 28-37.

[SWJ95] G. Schreiber, B. Wielinga, and W. Jansweijer: The KACTUS View on the 'O' World. In: *Proc. NAIC-95*, 1995.

[Tay92] R.M. Taylor (ed.): Principles of the LCM, vol 1. Technical Report KADS-II/T2.1/TR/TRMC/010/1.0, Touche Ross, London, 1992.

[THW+93] P. Terpstra, G. van Heijst, B. Wielinga, and N. Shadtbolt: Knowledge Acquisition Support through Generalised Directive Models. In: Jean-Marc David, Jean-Paul Krivine, Reid Simmons (eds.), S*econd Generation Expert Systems*. Springer-Verlag, Berlin, 1993, 428-455.

[ToA94] J. Top and H. Akkermans: Tasks and Ontologies in Engineering Modeling. In: *Int. J. Human-Computer Studies 41*, 1994, 585-617.

[Van95] J. Vanwelkenhuysen: Using DRE to Augment Generic Conceptual Design. In: *IEEE Expert*, February 1995, 50-56.

[vHB92] F. van Harmelen and J. Balder: $(ML)^2$: A Formal Language for KADS Conceptual Models. In: *Knowledge Acquisition 4*, 1, 1992.

[vHe95] G. van Heijst: The Role of Ontologies in Knowledge Engineering. Ph.D. Thesis, Univ. of Amsterdam, 1995.

[WSB92] B.J. Wielinga, A.Th. Schreiber, and J.A. Breuker: KADS: A Modeling Approach to Knowledge Engineering. In: *Knowledge Acquisition* 4, 1, 1992.

[You89] E. Yourdon: *Modern Structured Analysis*. Prentice-Hall, Englewood Cliffs, 1989.