Knowledge Engineering: Principles and Methods

Rudi Studer[1], V. Richard Benjamins[2], and Dieter Fensel[1]

[1]Institute AIFB, University of Karlsruhe, 76128 Karlsruhe, Germany
{studer, fensel}@aifb.uni-karlsruhe.de
http://www.aifb.uni-karlsruhe.de

[2]Artificial Intelligence Research Institute (IIIA),
Spanish Council for Scientific Research (CSIC), Campus UAB,
08193 Bellaterra, Barcelona, Spain
richard@iiia.csic.es, http://www.iiia.csic.es/~richard

[2]Dept. of Social Science Informatics (SWI),
University of Amsterdam, Roetersstraat 15, 1018 WB Amsterdam, The Netherlands
richard@swi.psy.uva.nl, http://www.swi.psy.uva.nl/usr/richard/home.html

Abstract

This paper gives an overview about the development of the field of Knowledge Engineering over the last 15 years. We discuss the paradigm shift from a transfer view to a modeling view and describe two approaches which considerably shaped research in Knowledge Engineering: Role-limiting Methods and Generic Tasks. To illustrate various concepts and methods which evolved in the last years we describe three modeling frameworks: CommonKADS, MIKE, and PROTÉGÉ-II. This description is supplemented by discussing some important methodological developments in more detail: specification languages for knowledge-based systems, problem-solving methods, and ontologies. We conclude with outlining the relationship of Knowledge Engineering to Software Engineering, Information Integration and Knowledge Management.

Key Words

Knowledge Engineering, Knowledge Acquisition, Problem-Solving Method, Ontology, Information Integration

1 Introduction

In earlier days research in Artificial Intelligence (AI) was focused on the development of formalisms, inference mechanisms and tools to operationalize Knowledge-based Systems (KBS). Typically, the development efforts were restricted to the realization of small KBSs 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 large KBSs failed in many cases. The situation was directly comparable to a similar situation in the construction of traditional software systems, called „software crisis“ in the late sixties: the means to develop small academic prototypes did not scale up to the design and maintenance of large, long living commercial systems. In the same way as the software crisis resulted in the establishment of the discipline Software Engineering the unsatisfactory situation in constructing KBSs made clear the need for more methodological approaches.

So the goal of the new discipline Knowledge Engineering (KE) is similar to that of Software Engineering: turning the process of constructing KBSs from an art into an engineering discipline. This requires the analysis of the building and maintenance process itself and the development of appropriate methods, languages, and tools specialized for developing KBSs.

Subsequently, we will first give an overview of some important historical developments in KE: special emphasis will be put on the paradigm shift from the so-called *transfer approach* to the so-called *modeling approach*. This paradigm shift is sometimes also considered as the transfer from first generation expert systems to second generation expert systems [43]. Based on this discussion Section 2 will be concluded by describing two prominent developments in the late eighties: *Role-limiting Methods* [99] and *Generic Tasks* [36]. In Section 3 we will present some modeling frameworks which have been developed in recent years: CommonKADS [129], MIKE [6], and PROTÈGÈ-II [123]. Section 4 gives a short overview of specification languages for KBSs. Problem-solving methods have been a major research topic in KE for the last decade. Basic characteristics of (libraries of) problem-solving methods are described in Section 5. Ontologies, which gained a lot of importance during the last years are discussed in Section 6. The paper concludes with a discussion of current developments in KE and their relationships to other disciplines.

In KE much effort has also been put in developing methods and supporting tools for knowledge elicitation (compare [48]). E.g. in the VITAL approach [130] a collection of elicitation tools, like e.g. repertory grids (see [65], [83]), are offered for supporting the elicitation of domain knowledge (compare also [49]). However, a discussion of the various elicitation methods is beyond the scope of this paper.

2 Historical Roots

2.1 Basic Notions

In this section we will first discuss some main principles which characterize the development of KE from the very beginning.

Knowledge Engineering as a Transfer 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.” [81]*

In the early eighties the development of a KBS has been seen as a *transfer process* of human knowledge into an implemented knowledge base. This transfer was based on the assumption that the knowledge which is required by the KBS already exists and just has to be collected and implemented. Most often, the required knowledge was obtained by interviewing experts on how they solve specific tasks [108]. Typically, this knowledge was implemented in some kind of production rules which were executed by an associated rule interpreter.

However, a careful analysis of the various rule knowledge bases showed that the rather simple representation formalism of production rules did not support an adequate representation of different types of knowledge [38]: e.g. in the MYCIN knowledge base [44] strategic knowledge about the order in which goals should be achieved (e.g. "consider common causes of a disease first") is mixed up with domain specific knowledge about for example causes for a specific disease. This mixture of knowledge types, together with the lack of adequate justifications of the different rules, makes the maintenance of such knowledge bases very difficult and time consuming. Therefore, this transfer approach was only feasible for the development of small prototypical systems, but it failed to produce large, reliable and maintainable knowledge bases.

Furthermore, it was recognized that the assumption of the transfer approach, that is that knowledge acquisition is the collection of already existing knowledge elements, was wrong due to the important role of tacit knowledge for an expert's problem-solving capabilities. These deficiencies resulted in a paradigm shift from the transfer approach to the modeling approach.

Knowledge Engineering as a Modeling Process

Nowadays there exists an overall consensus that the process of building a KBS may be seen as a *modeling activity*. Building a KBS means building a computer model with the aim of realizing problem-solving capabilities comparable to a domain expert. 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, but has to be built up and structured during the knowledge acquisition phase. Therefore this knowledge acquisition process is no longer seen as a transfer of knowledge into an appropriate computer representation, but as a model construction process ([41], [106]).

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

- Like every model, such a model is only an *approximation* of the reality. In principle, the modeling process is infinite, because it is an incessant activity with the aim of approximating 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 subjective interpretations of the knowledge engineer. Therefore this process is typically *faulty* and an evaluation of the model with respect to reality is indispensable for the creation of an adequate model. According to this feedback loop, the model must therefore be revisable in every stage of the modeling process.

Problem Solving Methods

In [39] Clancey reported on the analysis of a set of first generation expert systems developed to solve different tasks. 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 problem-solving behaviour of these systems on an abstract level, the so called *Knowledge Level* [113]. This knowledge 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 of a problem-solving process abstracts from details concerned with the implementation of the reasoning process and results in the notion of a *Problem-Solving Method* (PSM).

A PSM may be characterized as follows (compare [20]):

- A PSM specifies which *inference actions* have to be carried out for solving a given task.
- A PSM determines the sequence in which these actions have to be activated.
- In addition, so-called *knowledge roles* determine which role the domain knowledge plays in each inference action. These knowledge roles define a domain independent generic terminology.

When considering the PSM *Heuristic Classification* in some more detail (Figure 1) we can identify the three basic inference actions *abstract*, *heuristic match*, and *refine*. Furthermore, four knowledge roles are defined: *observables*, *abstract observables*, *solution abstractions*, and *solutions*. It is important to see that such a description of a PSM is given in a generic way. Thus the reuse of such a PSM in different domains is made possible. When considering a medical domain, an observable like „41^0 C“ may be abstracted to „high temperature“ by the inference action *abstract*. This abstracted observable may be matched to a solution abstraction, e.g. „infection“, and finally the solution abstraction may be hierarchically refined to a solution, e.g. the disease „influenca“.

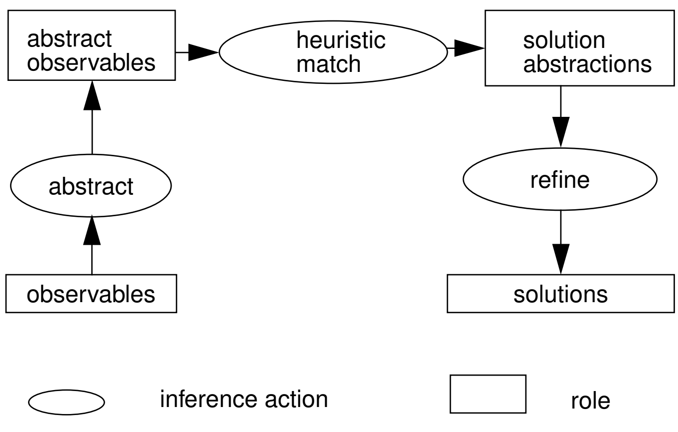

Fig. 1 The Problem-Solving Method *Heuristic Classification*

In the meantime various PSMs have been identified, like e.g. *Cover-and-Differentiate* for solving diagnostic tasks [99] or *Propose-and-Revise* [100] for parametric design tasks.

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

- PSMs contain inference actions which need specific knowledge in order to perform their task. For instance, *Heuristic Classification* needs a hierarchically structured model of observables and solutions for the inference actions *abstract* and *refine*, respectively. So a PSM may be used as a guideline to acquire static domain knowledge.
- A PSM allows to describe the main rationale of the reasoning process of a KBS which supports the validation of the KBS, because the expert is able to understand the problem solving process. In addition, this abstract description may be used during the problem-solving process itself for explanation facilities.
- Since PSMs may be reused for developing different KBSs, a library of PSMs can be exploited for constructing KBSs from reusable components.

The concept of PSMs has strongly stimulated research in KE and thus has influenced many approaches in this area. A more detailed discussion of PSMs is given in Section 5.

2.2 Specific Approaches

During the eighties two main approaches evolved which had significant influence on the development of modeling approaches in KE: Role-Limiting Methods and Generic Tasks.

Role-Limiting Methods

Role-Limiting Methods (RLM) ([99], [102]) have been one of the first attempts to support the development of KBSs 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 PSM and thus can only be used to solve a type of tasks for which the PSM is appropriate. The given PSM 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 1). A RLM based on Heuristic Classification offers a role *observables* 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 the kind of knowledge, which is used by the RLM, is predefined. Therefore, the acquisition of the required domain specific instances may be supported by (graphical) interfaces which are custom-tailored for the given PSM.

In the following we will discuss one RLM in some more detail: SALT ([100], [102]) which is used for solving constructive tasks.Then we will outline a generalization of RLMs to so-called Configurable RLMs.

SALT is a RLM for building KBSs which use the PSM Propose-and-Revise. Thus KBSs may be constructed for solving specific types of design tasks, e.g. parametric design tasks. The basic inference actions that Propose-and-Revise is composed of, may be characterized as follows:

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

In essence three generic roles may be identified for Propose-and-Revise ([100]):

- „design-extensions” refer to knowledge for proposing a new value for a design parameter,
- „constraints” provide knowledge restricting the admissible values for parameters, and
- „fixes” make potential remedies available for specific constraint violations.

From this characterization of the PSM Propose-and-Revise, one can easily see that the PSM is described in generic, domain-independent terms. 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 the VT-system [101], a KBS for configuring elevators, the domain expert used the form-oriented user interface of SALT for entering domain specific design extensions (see Figure 2). That is, the generic terminology of the knowledge roles, which is defined by object and relation types, is instantiated with 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.

(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 [100]))

Fig. 2 Design Extension Knowledge for VT

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 specific task may be solved by a given RLM. Such 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. Moreover, RLMs have a fixed structure and do not provide a good basis when a particular task can only be solved by a combination of several PSMs.

In order to overcome this inflexibility of RLMs, the concept of configurable RLMs has been proposed. *Configurable Role-Limiting Methods* (CRLMs) as discussed in [121] 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 Section 5). In [121], 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 that are predetermined 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 [121]. Having a fixed scheme of knowledge types and predefined communication paths between the various components is an important restriction distinguishing the CRLM framework from more flexible configuration approaches such as CommonKADS (see Section 3).

It should be clear that the introduction of such flexibility into the RLM approach removes one of its disadvantages while 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 system in the right way.

Generic Task and Task Structures

In the early eighties the analysis and construction of various KBSs for diagnostic and design tasks evolved gradually into the notion of a *Generic Task* (GT) [36]. GTs like *Hierarchical Classification* or *State Abstraction* are building blocks which can be reused for the construction of different KBSs.

The basic idea of GTs may be characterized as follows (see [36]):

- A GT is associated with a generic description of its input and output.
- A GT comes with a fixed scheme of knowledge types specifying the structure of domain knowledge needed to solve a task.
- A GT includes a fixed problem-solving strategy specifying the inference steps the strategy is composed of and the sequence in which these steps have to be carried out.

The GT approach is based on the *strong interaction problem hypothesis* which states that the structure and representation of domain knowledge is completely determined by its use [33]. Therefore, a GT comes with both, a fixed problem-solving strategy and a fixed collection of knowledge structures.

Since a GT fixes the type of knowledge which is needed to solve the associated task, a GT provides a task specific vocabulary which can be exploited to guide the knowledge acquisition process. Furthermore, by offering an executable shell for a GT, called a task specific architecture, the implementation of a specific KBS could be considered as the instantiation of the predefined knowledge types by domain specific terms (compare [34]). On a rather pragmatic basis several GTs have been identified including *Hierarchical Classification*, *Abductive Assembly* and *Hypothesis Matching*. This initial collection of GTs was considered as a starting point for building up an extended collection covering a wide range of relevant tasks.

However, when analyzed in more detail two main disadvantages of the GT approach have been identified (see [37]):

- The notion of task is conflated with the notion of the PSM used to solve the task, since each GT included a predetermined problem-solving strategy.

- The complexity of the proposed GTs was very different, i.e. it remained open what the appropriate level of granularity for the building blocks should be.

Based on this insight into the disadvantages of the notion of a GT, the so-called *Task Structure* approach was proposed [37]. The Task Structure approach makes a clear distinction between a task, which is used to refer to a type of problem, and a method, which is a way to accomplish a task. In that way a task structure may be defined as follows (see Figure 3): a task is associated with a set of alternative methods suitable for solving the task. Each method may be decomposed into several subtasks. The decomposition structure is refined to a level where elementary subtasks are introduced which can directly be solved by using available knowledge.

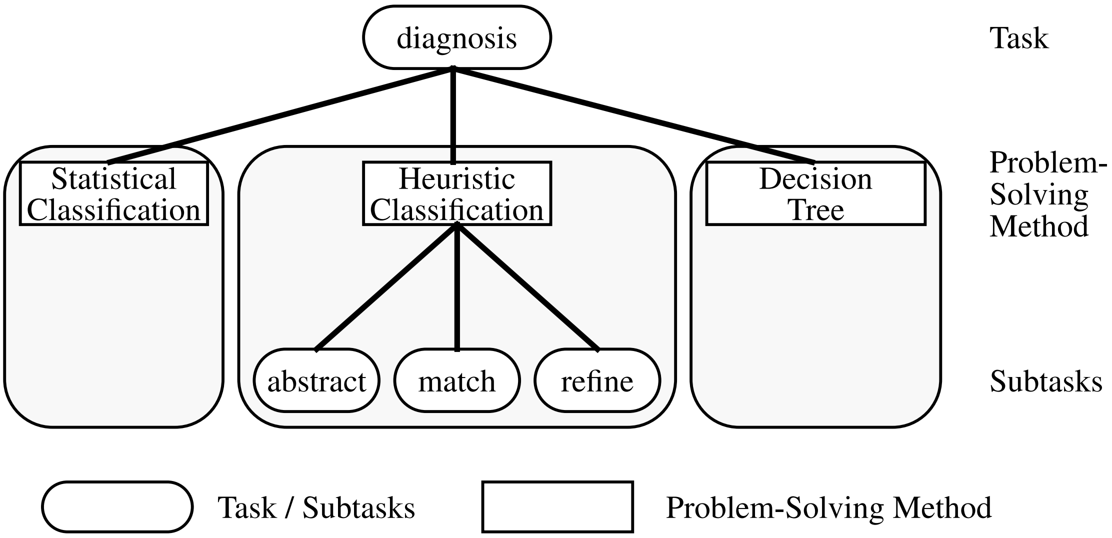

Fig. 3 Sample Task Structure for Diagnosis

As we will see in the following sections, the basic notion of task and (problem-solving) method, and their embedding into a task-method-decomposition structure are concepts which are nowadays shared among most of the knowledge engineering methodologies.

3 Modeling Frameworks

In this section we will describe three modeling frameworks which address various aspects of model-based KE approaches: CommonKADS [129] is prominent for having defined the structure of the Expertise Model, MIKE [6] puts emphasis on a formal and executable specification of the Expertise Model as the result of the knowledge acquisition phase, and PROTÉGÉ-II [51] exploits the notion of ontologies.

It should be clear that there exist further approaches which are well known in the KE community, like e.g VITAL [130], Commet [136], and EXPECT [72]. However, a discussion of all these approaches is beyond the scope of this paper.

3.1 The CommonKADS Approach

A prominent knowledge engineering approach is *KADS* [128] and its further development to *CommonKADS* [129]. A basic characteristic of KADS is the construction of a collection of models, where each model captures specific aspects of the KBS 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 (non-)functional aspects of the KBS under development.

Subsequently, we will briefly discuss each of these models and then provide a detailed description of the Expertise Model:

- Within the *Organization Model* the organizational structure is described together with a specification of the functions which are performed by each organizational unit. Furthermore, the deficiencies of the current business processes, as well as opportunities to improve these processes by introducing KBSs, are identified.
- The *Task Model* provides a hierarchical description of the tasks which are performed in the organizational unit in which the KBS will be installed. This includes a specification of which agents are assigned to the different tasks.
- The *Agent Model* specifies the capabilities of each agent involved in the execution of the tasks at hand. In general, an agent can be a human or some kind of software system, e.g. a KBS.
- Within the *Communication Model* the various interactions between the different agents are specified. Among others, it specifies which type of information is exchanged between the agents and which agent is initiating the interaction.

A major contribution of the KADS approach is its proposal for structuring the *Expertise Model*, which distinguishes three different types of knowledge required to solve a particular task. Basically, the three different types correspond to a static view, a functional view and a dynamic view of the KBS to be built (see in Figure 4 respectively “domain layer“, “inference layer“ and “task layer“):

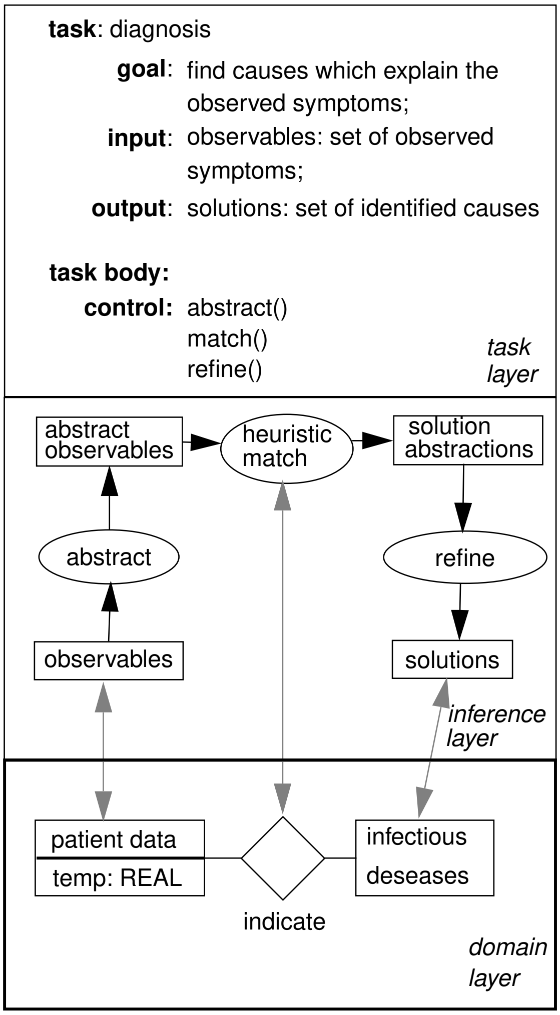

Fig. 4 Expertise Model for medical diagnosis (simplified CML notation)

- *Domain layer*: At 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 in a domain ontology (see Section 6), and a declarative theory of the required domain knowledge. One objective for structuring the domain layer is to model it as reusable as possible for solving different tasks.
- *Inference layer*: At the inference layer the reasoning process of the KBS is specified by exploiting the notion of a PSM. The inference layer describes the *inference actions* the generic PSM is composed of as well as the *roles*, which are played by the domain knowledge within the PSM. The dependencies between inference actions and roles are specified in what is called an *inference structure.* Furthermore, the notion of roles provides a domain independent view on the domain knowledge. In Figure 4 (middle part) we see the inference structure for the PSM *Heuristic Classification*. Among others we can see that „patient data” plays the role of „observables” within the inference structure of *Heuristic Classification*.
- *Task layer*: 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. The task layer also provides means for specifying the control over the subtasks and inference actions, which are defined at the inference layer.

Two types of languages are offered to describe an *Expertise Model*: CML (Conceptual Modeling Language) [127], which is a semi-formal language with a graphical notation, and $(ML)^2$ [79], which is a formal specification language based on first order predicate logic, meta-logic and dynamic logic (see Section 4). Whereas CML is oriented towards providing a communication basis between the knowledge engineer and the domain expert, $(ML)^2$ is oriented towards formalizing the *Expertise Model*.

The clear separation of the domain specific knowledge from the generic description of the PSM at 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. This reuse approach is a weakening of the strong interaction problem hypothesis [33] which was addressed in the GT approach (see Section 2). In [129] the notion of a *relative interaction hypothesis* is defined to indicate that some kind of dependency exists between the structure of the domain knowledge and the type of task which should be solved. To achieve a flexible adaptation of the domain layer to a new task environment, the notion of layered ontologies is proposed: *Task* and *PSM ontologies* may be defined as viewpoints on an underlying domain ontology.

Within CommonKADS a library of reusable and configurable components, which can be used to build up an *Expertise Model*, has been defined [29]. A more detailed discussion of PSM libraries is given in Section 5.

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 much as possible [129].

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 similar to Boehm's spiral model [21].

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 [155]): Whereas an inference layer is specified in generic terms and provides - via roles and domain views - a flexible connection to the data described at 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 [54]).

3.2 The MIKE Approach

The *MIKE approach* (*Model-based and Incremental Knowledge Engineering*) (cf. [6], [7]) provides a development method for KBSs covering all steps from the initial elicitation through specification to design and implementation. MIKE proposes the integration of *semiformal* and *formal specification techniques* and *prototyping* into an engineering framework. Integrating prototyping and support for an incremental and reversible system development process into a model-based framework, is actually the main distinction between MIKE and CommonKADS [129]:

- MIKE takes the *Expertise Model* of CommonKADS as its general model pattern and provides a smooth transition from a semiformal representation, the *Structure Model*, to a formal representation, the *KARL Model*, and further to an implementation oriented representation, the *Design Model*. The smooth transition between the different representation levels of the *Expertise Model* is essential for enabling incremental and reversible system development in practice.
- In MIKE the executability of the *KARL Model* enables validation of the *Expertise Model* by prototyping. This considerably enhances the integration of the expert in the development process.

The different MIKE development activities and the documents resulting from these activities are shown in Figure 5. In MIKE, the entire development process is divided into a number of subactivities: *Elicitation*, *Interpretation*, *Formalization/Operationalization*, *Design,* and *Implementation*. Each of these activities deals with different aspects of the system development.

The knowledge acquisition process starts with *Elicitation*. Methods like structured interviews [48] are used for acquiring informal descriptions of the knowledge about the specific domain and the problem-solving process itself. The resulting knowledge expressed in natural language is stored in so-called *knowledge protocols*.

During the *Interpretation* phase the knowledge structures which may be identified in the *knowledge protocols* are represented in a semi-formal variant of the *Expertise Model*: the *Structure Model* [112]. All structuring information in this model, like the data dependencies between two inferences, is expressed in a fixed, restricted language while the basic building blocks, e.g. the description of an inference, are represented by unrestricted texts. This representation provides an initial structured description of the emerging knowledge structures and can be used as a communication basis between the knowledge engineer and the expert. Thus the expert can be integrated in the process of structuring the knowledge.

The *Structure Model* is the foundation for the *Formalization/Operationalization* process which results in the formal *Expertise Model*: the *KARL Model*. The *KARL Model* has the same conceptual structure as the *Structure Model* while the basic building blocks which have been represented as natural language texts are now expressed in the formal specification language *KARL* (cf. [53], [55]). This representation avoids the vagueness and ambiguity of natural language descriptions and thus helps to get a clearer understanding of the entire problem-solving process. The *KARL Model* can be directly mapped to an operational representation because KARL (with some small limitations) is an executable language.

The result of the knowledge acquisition phase, the *KARL Model*, captures all functional requirements for the final KBS. During the *Design* phase additional non-functional requirements are considered. These non-functional requirements include e.g. efficiency and maintainability, but also the constraints imposed by target software and hardware environments. Efficiency is already partially covered in the knowledge acquisition phase, but only to the extent as it determines the PSM. Consequently, functional decomposition is already part of the knowledge acquisition phase. Therefore, the design phase in MIKE constitutes the equivalent of detailed design and unit design in software engineering approaches. The *Design Model* which is the result of this phase is expressed in the language *DesignKARL* [89]. DesignKARL extends KARL by providing additional primitives for structuring the *KARL Model* and for describing algorithms and data types. DesignKARL additionally allows to describe the design process itself and the interactions between design decisions [90].

The *Design Model* captures all functional and non-functional requirements posed to the KBS. In the *Implementation* process the *Design Model* is implemented in the target hardware and software environment.

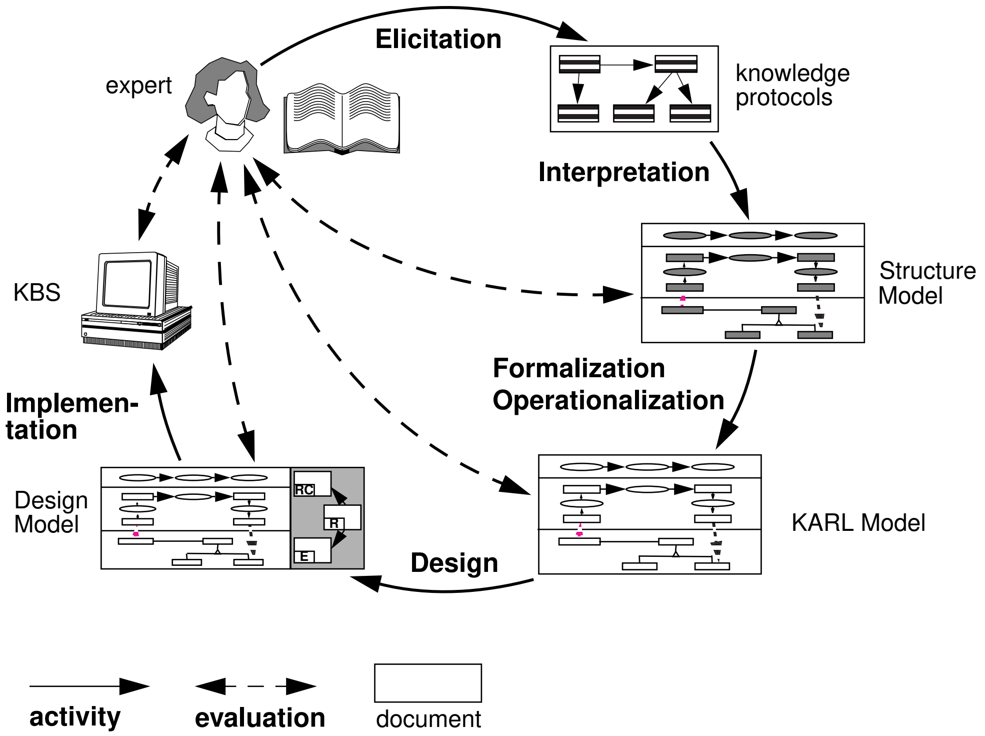

Fig. 5 Steps and documents in the MIKE development process

The result of all phases is a set of several interrelated refinement states of the *Expertise Model*. The knowledge in the *Structure Model* is related to the corresponding knowledge in the *knowledge protocols* via explicit *links*. Concepts and inference actions are related to protocol nodes, in which they have been described using natural language. The *Design Model* refines the *KARL Model* by refining inferences into algorithms and by introducing additional data structures. These parts of the *Design Mode*l are linked to the corresponding inferences of the *KARL Model* and are thus in turn linked to the *knowledge protocols*. Combined with the goal of preserving the structure of the *Expertise Model* during design, the links between the different model variants and the final implementation ensure traceability of (non-)functional requirements.

The entire development process, i.e. the sequence of knowledge acquisition, design, and implementation, is performed in a cycle guided by a *spiral model* [21] as process model.

Every cycle produces a prototype of the KBS which may be evaluated by testing it in the real target environment. The results of the evaluation are used in the next cycle to correct, modify, or extend this prototype.

The development process of MIKE inherently integrates different types of prototyping [7]: The executability of the language KARL allows the *Expertise Model* to be built by explorative prototyping. Thus the different steps of knowledge acquisition are performed cyclically until the desired functionality has been reached. In a similar way, experimental prototyping can be used in the design phase for evaluating the *Design Model* since DesignKARL can be mapped to an executable version. The *Design Model* is refined by iterating the subphases of the design phase until all non-functional requirements are met.

Recently, a new version of the specification language KARL has been developed [5] which integrates the notion of task and thus provides means to formally specify task-to-method decomposition structures.

The MIKE approach as described above is restricted to modeling the KBS under development. To capture the embedding of a KBS into a business environment, the MIKE approach is currently extended by new models which define different views on an enterprise. Main emphasis is put on a smooth transition from business modeling to the modeling of problem-solving processes [45].

3.3 The PROTÉGÉ-II Approach

The *PROTÉGÉ-II approach* (cf. [123], [51]) aims at supporting the development of KBSs by the reuse of PSMs and ontologies. In addition, PROTÉGÉ-II puts emphasis on the generation of custom-tailored knowledge-acquisition tools from ontologies [50].

PROTÉGÉ-II relies on the task-method-decomposition structure (compare Section 2.2): By applying a PSM a task is decomposed into corresponding subtasks. This decomposition structure is refined down to a level at which primitive methods, so-called *mechanisms*, are available for solving the subtasks. Up to now, PROTÉGÉ-II offers a small library of implemented PSMs (see Section 5) which have been used to solve a variety of tasks (see [51], [69] for more details).

In PROTÉGÉ-II the input and output of a method is specified by a so-called *method ontology* (cf. [51], [137]) (see Figure 6): such a method ontology defines the concepts and relationships that are used by the PSM for providing its functionality. E.g. the *Board-Game Method* [51] uses among others the notions of ‘pieces’, ‘locations’, and ‘moves’ to provide its functionality, that is to move pieces between locations on a board. In this way a method ontology corresponds to the generic terminology as introduced by the collection of knowledge roles of a PSM (compare Section 2.1).

A second type of ontology used within PROTÉGÉ-II are *domain ontologies*: they define a shared conceptualization of a domain (see Section 6). Both PSMs and domain ontologies are reusable components for building up a KBS. However, due to the interaction problem the interdependence between domain ontologies and PSMs with their associated method ontologies has to be taken into account when constructing a KBS from reusable components. Therefore, PROTÉGÉ-II proposes the notion of an *application ontology* to extend domain ontologies with PSM specific concepts and relationships [71]. In order to associate an application ontology with a method ontology, PROTÉGÉ-II offers different types of mapping relations [71]:

- *Renaming mappings* are used for translating domain specific terms into method specific terms.
- *Filtering mappings* provide means for selecting a subset of domain instances as instances of the corresponding method concept.
- *Class mappings* provide functions to compute instances of method concepts from application concept definitions rather than from application instances.

Mappings are, on the one hand, similar to the schema translation rules as discussed in the area of interoperable database systems (see e.g. [126]), and, on the other hand, they correspond e.g. to the lift operator in $(ML)^2$ [79] or the view definitions in KARL [53] (compare Section 4). PROTÉGÉ-II aims at providing a small collection of rather simple mappings to limit the reuse effort needed to specify these mappings. Thus PROTÉGÉ-II recommends to reuse domain knowledge only in situations where the required mappings can be kept relatively simple [71].

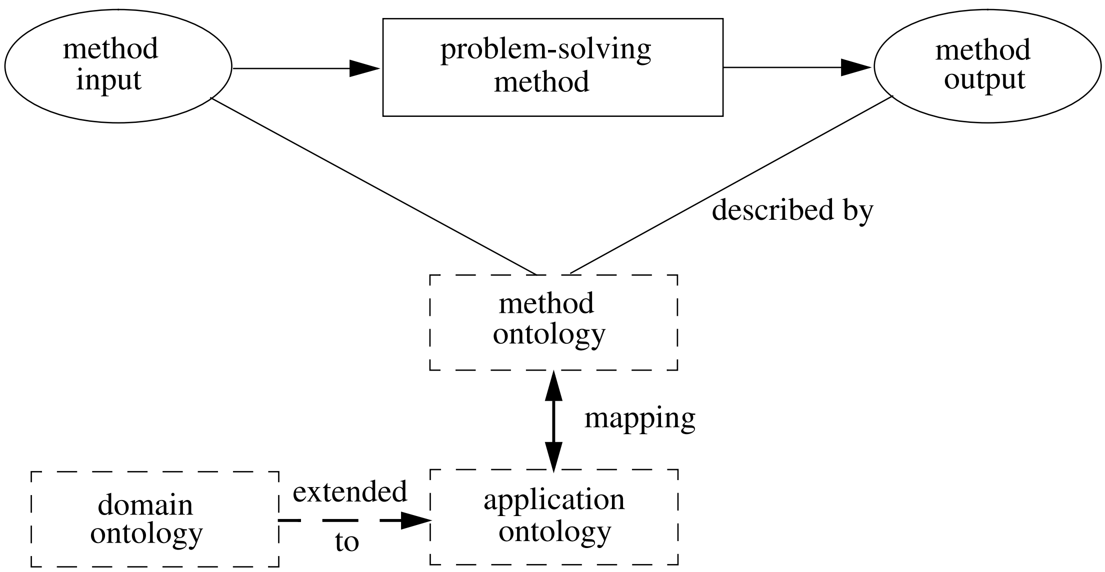

Fig. 6 Ontologies in PROTÉGÉ-II

A feature of PROTÉGÉ-II is that it can generate knowledge-acquisition tools from domain or application ontologies [50]. Therefore, the PROTÉGÉ-II system includes the DASH component which takes an ontology as input and generates as output a knowledge-acquisition tool that allows domain specialists to enter instances of the domain concepts. Thus domain facts can be specified by the domain expert himself.

Recently, the PROTÉGÉ-II approach has been extended to CORBA-based PSMs and ontologies which enables the reuse of these components in an Internet environment [70].

4 Specification Approaches in Knowledge Engineering

Over the last ten years a number of specification languages have been developed for describing KBSs. These specification languages can be used to specify the knowledge required by the system as well as the reasoning process which uses this knowledge to solve the task which is assigned to the system. On the one hand, these languages should enable a specification which abstracts from implementation details. On the other hand, they should enable a detailed and precise specification of a KBS at a level which is beyond the scope of specifications in natural language. This area of research is quite well documented by a number of workshops[1] and comparison papers that were based on these workshops. Surveys of these languages can be found in [143] and [61]. A short description of their history and usefulness is provided by [80], and [54] provides a comparison to similar approaches in software engineering. In this article, we will focus on the main principles of these languages. Basically, we will discuss the general need for specification languages for KBSs, we show their essence, we sketch some approaches, add a comparison to related areas of research and conclude by outlining lines of current and future research.

4.1 Why did the Need Arise for Specification Languages in the Late 1980's

As mentioned above, we can roughly divide the development of knowledge engineering into the knowledge transfer and the knowledge modelling period. During the former period, knowledge was directly encoded using rule-based implementation languages or frame-based systems. The (implicit) assumption was that these representation formalisms are adequate to express knowledge, reasoning, and functionality of a KBSs in a way which is understandable for humans and for computers. However, as mentioned in Section 2, severe difficulties arose [40]:

- different types of knowledge were represented uniformly,
- other types of knowledge were not presented explicitly,
- the level of detail was too high to present abstract models of the KBS,
- and knowledge level aspects got constantly mixed with aspects of the implementation.

As a consequence, such systems were hard to build and to maintain when they become larger or used over a longer period. In consequence, many research groups worked on more abstract description means for KBSs. Some of them were still executable (like the generic tasks [37]) whereas others combined natural language descriptions with semiformal specifications. The most prominent approach in the latter area is the KADS and CommonKADS approach [129] that introduced a conceptual model (the *Expertise Model*) to describe KBSs at an abstract and implementation independent level. As explained above, the *Expertise Mode*l distinguishes different knowledge types (called layers) and provides for each knowledge type different primitives (for example, knowledge roles and inference actions at the inference layer) to express the knowledge in a structured manner. A semiformal specification language CML [127] arose that incooperates these structuring mechanisms in the knowledge level models of KBSs. However, the elementary primitives of each model were still defined by using natural language. Using natural language as a device to specify computer programs has well known advantages and disadvantages. It provides freedom, richness, easiness in use and understanding, which makes it a comfortable tool in sketching what one expects from a program. However, its inherent vagueness and implicitness make it often very hard to answer questions whether the system really does what is expected, or whether the model is consistent or correct (cf. [2], [80]). Formal specification techniques arose that overcome these shortcomings. Usually they were not meant as a replacement of semiformal specifications but as the possibility to improve the preciseness of a specification when required.

---

1. See ftp://swi.psy.uva.nl/pub/keml/keml.html at the World Wide Web.

Meanwhile, around twenty different approaches can be found in the literature ([61], [54]). Some of them aim mainly at formalization. A formal semantics is provided that enables the unique definition of knowledge, reasoning, or functionality along with manual or automated proofs. Other approaches aim at operationalization, that is, the specification of a system can be executed which enables prototyping in the early phase of system development. Here, the evaluation of the specification is the main interest. They help to answer the question whether the specification really specifies what the user is expecting or the expert is providing. Some approaches aim at formalizing and operationalizing, however they have to tackle conflicting requirements that arise from these two goals.

In the following subsection, we will discuss the main common features of these approaches.

4.2 The Essence of Specification Languages for KBSs

We identify three key features of specification languages for KBSs. First, most languages make use of a strong conceptual model to structure formal specifications. This reflects the fact that these languages were motivated by formalizing semiformal notations to describe KBSs. Therefore, these languages offer more than just a mathematical notation to define a computer program as an input-output relationship. Second, these languages have to provide means to specify the dynamic reasoning of a KBSs because this establishes a significant piece of the expertise required by such systems. Third, a KBS uses a large body of knowledge requiring structured and rich primitives for representing it. In the following, we will discuss these different aspects.

Formalising a Conceptual Model

Specification languages for KBSs arose to formalize conceptual models of KBSs. They use the structuring principles of semiformal specifications and add formal semantics to the elementary primitives and their composition (cf. [46], [47], [154]). As introduced above, the *Expertise Model* [129] describes the different types of knowledge required by a KBS as well as the role of this knowledge in the reasoning process of the KBS. Based on this, specification languages provide formal means for precisely defining:

- the goals and the process to achieve them,
- the functionality of the inference actions, and
- the precise semantics of the different elements of the domain knowledge.

Definitions in natural language are supplemented with formal definitions to ensure disambiguity and preciseness. The structure of the conceptual models naturally organizes the formal specification, improving understandability and simplifying the specification process.

The What and the How: Specification of Reasoning

In Software Engineering, the distinction between a functional specification and the design/implementation of a system is often discussed as a separation of *what* and *how*. During the specification phase, *what* the system should do is established in interaction with the users. *How* the system functionality is realized is defined during design and implementation (e.g., which algorithmic solution can be applied). This separation - which even in the domain of Software Engineering is often not practicable in the strict sense - does *not* work in the same way for KBSs: A high amount of the problem-solving knowledge, i.e. knowledge about *how* to meet the requirements, is not a question of efficient algorithms and data structures, but exists as domain-specific and task-specific heuristics as a result of the experience of an expert. For many problems which are completely specifiable it is not possible to find an efficient algorithmic solution. Such problems are easy to specify but it is not necessarily possible to derive an efficient algorithm from these specifications (cf. [32], [110]); domain-specific heuristics or domain-specific inference knowledge is needed for the efficient derivation of a solution. “In simple terms this means that analysis is not simply interested in *what* happens, as in conventional systems, but also with *how* and *why”* [31]. One must not only acquire knowledge about what a solution for a given problem is, but also knowledge about how to derive such a solution in an efficient manner [60]. Already at the knowledge level there must be a description of the domain knowledge and the problem-solving method which is required by an agent to solve the problem effectively and efficiently. In addition, the symbol level has to provide a description of efficient algorithmic solutions and data structures for implementing an efficient computer program. As in Software Engineering, this type of knowledge can be added during the design and implementation of the system. Therefore a specification language for KBSs must *combine non-functional and functional specification techniques*: On the one hand, it must be possible to express algorithmic control over the execution of substeps. On the other hand, it must be possible to characterize the overall functionality and the functionality of the substeps without making commitments to their algorithmic realization. The former is necessary to express problem solving at the knowledge level. The latter is necessary to abstract from aspects that are only of interest during implementation of the system.

Representing Rich Knowledge Structures

Most of the specification languages provide at least epistemological primitives like constants, sorts/types, functions, predicates/relations, and some mathematical toolkit as a means to specify the static aspects of a system. However, richer languages provide additional syntactical sugar on top of these mathematical primitives. In particular, object-oriented or frame-based specification languages provide a rich variety of appropriate modelling primitives for expressing static system aspects: values, objects, classes, attributes with domain and range restrictions, set-valued attributes, is-a relationships with attribute inheritance, aggregation, grouping etc. For example, KARL [53] provides such modelling primitives which enables a smooth transformation from semiformal specification languages like CML to formal specifications of these models.

4.3 Some Approaches And Their Technical Means

In the following, we briefly sketch three different specification languages for KBSs: $(ML)^2$, KARL, and DESIRE. A more detailed discussion and comparison can be found in [54].

$(ML)^2$ [79], which was developed as part of the KADS projects, is a *formalization* language for KADS *Expertise Models*. The language provides a formal specification language for the KADS *Expertise Mode*l by combining three types of logic: order-sorted first-order logic extended by modularization for specifying the domain layer, first-order meta-logic for specifying the inference layer, and quantified dynamic logic [77] for specifying the task layer.

KARL [53] is an *operational* language which restricts the expressive power of the object logic by using a variant of Horn logic. It was developed as part of the MIKE project [6] and provides a formal and executable specification language for the KADS *Expertise Model* by combining two types of logic: L-KARL and P-KARL. L-KARL, a variant of Frame Logic [84], is provided to specify domain and inference layers. It combines first-order logic with semantic data modelling primitives (see [30] for an introduction to semantic data models). A restricted version of dynamic logic is provided by P-KARL to specify a task layer. Executability is achieved by restricting Frame logic to Horn logic with stratified negation and by restricting dynamic logic to regular and deterministic programs.

The language DESIRE (*DEsign and Specification of Interacting REasoning components* [91],[92]) relies on a different conceptual model for describing a KBS: the notion of a compositional architecture. A KBS is decomposed into several interacting components. Each component contains a piece of knowledge at its object-layer and has its own control defined at its internal meta-layer. The interaction between components is represented by transactions and the control flow between these modules is defined by a set of control rules. DESIRE extensively uses object-meta relationships to structure specifications of KBSs. At the object-level, the system reasons about the world state. Knowledge about how to use this knowledge to guide the reasoning process is specified at the meta-level. The meta-level reasons about controlling the use of the knowledge specified at the object-level during the reasoning process. The meta-level describes the dynamic aspects of the object-level in a declarative fashion. A module may reason on its object-level about the meta-level of another module.

From a semantic point of view a significant difference between DESIRE on the one hand and $(ML)^2$ and KARL on the other hand lies in the fact that the former uses temporal logics for specifying the dynamic reasoning process whereas the latter use dynamic logic. In dynamic logic, the semantics of the overall program is a binary relation between its input and output sets. In DESIRE, the entire reasoning trace which leads to the derived output is used as semantics [142].

4.4 Comparison with Related Work

In the following, we investigate the relationships with three other areas. We take a look at the work that is done on validation and verification of KBSs, we discuss related work on specification languages in Software Engineering, and finally we discuss a recent trend in Software Engineering that makes use of the knowledge level for specifying other types of software products than KBSs only.

Comparison with V&V

The work on validation and verification of KBSs only partially follows the paradigm shift in Knowledge Engineering. Most of the work is still oriented to specific implementation formalisms like rule-based languages or languages stemming from the knowledge representation area. Surveys can be found in [96], [120]. A typical example is the work of [122] that provides algorithms to verify knowledge that is represented with production rules. Examples of such properties are:

- Unsatisfiable rule: the precondition of a rule cannot be true for any legal input.
- Unusable rule: the postcondition of the rule cannot be used to infer the goal nor any preconditions of another rule.
- Subsumed rule: there exists are more general rule in the rule set.

Other approaches apply testing as a means to validate KBSs. Most approaches do not rely on a formal specification in addition to the implemented knowledge. As a consequence, verification of the functionality of a system with respect to its formal specification is not possible. Only recently, the need was recognized for formal specifications and conceptual models that guide and structure the validation and verification process ([103],[78], [59]).

Comparison with Traditional Specification Approaches in SE

Work in Software Engineering on formal specification languages has a more than thirty year old tradition which has lead to a large number of approaches. An important line of research are algebraic specifications (cf. [46], [154]) that provide well-studied means for the mathematical definitions of the functionality of software systems. Some of the existing specification languages rely directly on these techniques (e.g., [134], [117]), other use similar ideas. The main extensions that are necessary to use them for KBSs are:

- Algebraic specification formalisms provide generic mathematical notations that have to be integrated into existing conceptual modelling techniques for KBSs to enable smooth transitions from semiformal to formal specifications.
- Algebraic specification formalisms provide a means to define the functionality of a software system abstracting from how it is computed. We explained earlier that such an abstraction is too strong for the purpose of specifying KBSs as part of a specification is to define how the output can be derived.

Specification approaches like the Vienna Development Method (VDM) [82] and Z [135], which describe a system in terms of states and operations working on these states, are closer in spirit to most specification approaches customized for KBSs. However, their vocabulary for expressing control over these state transitions is rather restricted by their purpose. Algorithmic control is regarded as an aspect that is added during design and implementation of a system. Also, the need for a rich conceptual model that structures a specification of a KBSs becomes apparent through the large case study of [105]. There the architecture of Soar is specified with more than hundred pages filled with formal specification “code“ in Z. Still, work on specification languages for KBSs must always be aware of approaches from Software Engineering or Information Systems Engineering because at a technical level many problems are basically the same.

Comparison with Recent Approaches in SE

> “USER: I need a software system that will help me manage my factory.
> SOFTWARE ENGINEER: Well, let´s see. I can put together components that do sorting, searching, stacks, queues, and so forth.
> USER: Hmm, that’s interesting. But how would those fit into my system?“[132]

Recently, the knowledge level has been encountered in Software Engineering (cf. [132]). Work on software architectures establishes a much higher level to describe the functionality and the structure of software artefacts. The main concern of this new area is the description of generic architectures that describe the essence of large and complex software systems. Such architectures specific classes of application problems instead of focusing on the small and generic components from which a system is built up.

The conceptual models developed in Knowledge Engineering for KBSs fit nicely in this recent trend. They describe an *architecture* for a specific class of systems: KBSs. Work on formalizing software architectures characterizes the functionality of architectures in terms of assumptions over the functionality of its components [118], [119]. This shows strong similarities to recent work on problem-solving methods that define the competence of problem-solving methods in terms of assumptions over domain knowledge (which can be viewed as one or several components of a KBSs) and the functionality of elementary inference steps (cf. [59]).

4.5 Recent Issues

One of the goals of formal specification approaches is to provide a clear formal notation. However, some constructs in the existing languages are rather ad-hoc and work is going on to provide well-defined formal ground for these languages. These activities also enable another goal of formal specifications: semiautomatic or automated proof support which helps to deal with the overwhelming amount of details that arise in formal proofs of specifications. Above, we mentioned recent work that integrate conceptual models into existing verification tools similar to earlier activities that integrates conceptual models into mathematical notations to provide high-level specification support. In addition to formal support in verification, one could also wish to provide automated support in deriving refined specifications or implementations of KBSs with techniques of automated program development (cf.[133]).

In the following sections we will discuss the subfields problem-solving methods and ontologies which are fruitful research areas generalizing the original notions of inference engines and domain knowledge bases. Clearly, these fields introduce new interesting requirements for formal approaches. The competence of problem-solving methods (i.e., their functionality and utility) need to be characterized in terms of assumptions on available knowledge. Ontologies need to provide meta-level characterizations of knowledge bases to support their reuse and to solve the interaction problem between the problem-solving process and the domain knowledge. Building large knowledge bases requires sophisticated structuring of, and mediation facilities between different ontologies.

5 Problem-Solving Methods

Originally, KBSs used simple and generic inference mechanisms to infer outputs for provided cases. The knowledge was assumed to be given “declaratively“ by a set of Horn clauses, production rules, or frames. Inference engines like unification, forward or backward resolution, and inheritance captured the dynamic part of deriving new information. However, human experts have exploit knowledge about the dynamics of the problem-solving *process* and such knowledge is required to enable problem-solving in practice and not only in principle [60]. [38] provided several examples where knowledge engineers implicitly encoded control knowledge by ordering production rules and premises of these rules that together with the generic inference engine, delivered the desired dynamic behaviour. Making this knowledge explicit and regarding it as an important part of the entire knowledge contained by a KBS, is the rationale that underlies *Problem-Solving Methods (PSMs)*. PSMs refine generic inference engines mentioned above to allow a more direct control of the reasoning process. PSMs describe this control knowledge independent from the application domain enabling reuse of this strategical knowledge for different domains and applications. Finally, PSMs abstract from a specific representation formalism as opposed to the general inference engines that rely on a specific representation of the knowledge. Meanwhile, a large number of such PSMs have been developed and libraries of such methods provide support in reusing them for building new applications.

In general, software and knowledge engineers agree that reuse is a promising way to reduce development costs of software systems and knowledge-based systems. The basic idea is that a KBS can be constructed from ready-made parts instead of being built up from scratch. Research on PSMs has adopted this philosophy and several PSM libraries have been developed.

In the following, we will discuss several issues involved in developing such libraries.

5.1 Types of PSM Libraries

Currently, there exist several libraries with PSMs. They all aim at facilitating the knowledge-engineering process, yet they differ in various ways. In particular, libraries differ along dimensions such as genericness, formality, granularity and size:

- The *genericness* dimension describes whether PSMs in a library are developed for a particular task. Task-specific libraries contain PSMs that are specialised in solving (parts of) a specific task such as diagnosis or design. Their 'task-specificness' resides mainly in the terminology in which the PSMs are formulated. Examples include libraries for design ([35], [107]), assessment [145], diagnosis [13] and planning ([10], [9]). The CommonKADS library can be viewed as an extensive collection of task-specific PSMs [29]. Task-independent libraries provide problem-solving methods that are not formulated in task-specific terminology [2].
- The *formality* dimension divides the libraries in informal, formal and implemented ones. Implemented libraries provide operational specifications of PSMs, which are directly executable ([123], [71]). Formal libraries allow for formal verification of properties of PSMs ([2],[3], [15], [139]). Finally, informal libraries provide structured textual representations of PSMs. Note that within the informal approaches, PSM descriptions can vary from just textual descriptions [35], to highly structured descriptions using diagrams [13].
- The *granularity* dimension distinguishes between libraries with complex components, in the sense that the PSMs realise a complete task [107], and libraries with fine-grained PSMs that realise a small part of the task [2]. Several libraries contain both large and small building-blocks where the former are built up from the latter ([13], [35], [10]).
- The *size* dimension. The most comprehensive general library is the CommonKADS library [29] which contains PSMs for diagnosis, prediction of behaviour, assessment, design, planning, assignment and scheduling and engineering modelling. The most extensive library for diagnosis [13] contains 38 PSMs for realising 14 tasks related to diagnosis. The library for parametric design [107] consists of five PSMs, several of them being variations of Propose-and-Revise [100]. The design library of [35] mentions about 15 PSMs.

The type of a library is determined by its characterisation in terms of the above dimensions. Each type has a specific role in the knowledge engineering process and has strong and weak points. The more general (i.e. task-neutral) PSMs in a library are, the more reusable they are, because they do not make any commitment to particular tasks. However, at the same time, applying such a PSM in a particular application requires considerable refinement and adaptation. This phenomenon is known as the reusability-usability trade-off [85]. Recently, research has been conducted to overcome this dichotomy by introducing adapters that gradually adapt task-neutral PSMs to task-specific ones [58] and by semi-automatically constructing the mappings between task-neutral PSMs and domain knowledge [19].

Libraries with informal PSMs provide above all support for the conceptual specification phase of the KBS, that is, they help significantly in constructing the reasoning part of the *Expertise Model* of a KBS [128]. Because such PSMs are informal, they are relatively easy to understand and malleable to fit a particular application. The disadvantage is - not surprisingly - that still much work has to be done before arriving at an implemented system. Libraries with formal PSMs are particularly important if the KBS needs to have some guaranteed properties, e.g. for use in safety-critical systems such as nuclear power plants. Their disadvantage is that they are not easy to understand for humans [24] and limit the expressiveness of the knowledge engineer. Apart from the possibility to prove properties, formal PSMs have the additional advantage of being a step closer to an implemented system. Libraries with implemented PSMs allow the construction of fully operational systems. The other side of the coin is, however, that the probability that operational PSMs exactly match the requirements of the knowledge engineer, is lower.

Developing a KBS using libraries with coarse-grained PSMs, amounts to selecting the most suitable PSM and then adapt it to the particular needs of the application [107]. The advantage is that this process is relatively simple as it involves only one component. The disadvantage is, however, that it is unlikely that such a library will have broad coverage, since each application might need a different (coarse-grained) PSM. The alternative approach is to have a library with fine-grained PSMs, which are then combined together (i.e. configured) into a reasoner, either manually ([123], [137]) or automatically ([14], [9]).

5.2 Organisation of Libraries

There are several alternatives for organising a library and each of them has consequences for indexing PSMs and for their selection. Finding the 'best' organisation principle for such libraries is still an issue of debate. In the following, we will present some organisation principles.

Several researchers propose to organise libraries as a *task-method decomposition* structure ([37], [123], [136], [130], [140], [153]), and some available libraries are indeed organised in this way ([13], [29], [10]). According to this organisation structure, a task can be realised by several PSMs, each consisting of primitive and/or composite subtasks. Composite subtasks can again be realized by alternative methods, etc. Guidelines for library design according to this principle are discussed in ([116], [115]). In a library organised according to the task-method principle, PSMs are indexed, based on two factors: (1) on the competence of the PSMs - which specifies what a PSM can achieve [8], and (2) on their assumptions - which specify the assumptions under which the PSM can be applied correctly, such as its requirements on domain knowledge ([17], [18]). Selection of PSMs from such libraries first considers the competence of PSMs (selecting those whose competences match the task at hand), and then the assumptions of PSMs (selecting those whose assumptions are satisfied).

Libraries can also be organised, based on the functionality of PSMs, in which case PSMs with similar functionality are stored together. In addition, the functionality of PSMs can be configured from pre-established parameters and values [139]. An opposite way to organize PSMs is proposed in [124] where PSMs are indexed according to the algorithms they use.

Another criterion to structure libraries of PSMs is based on solely assumptions, which specify under what conditions PSMs can be applied. Assumptions can refer to domain knowledge (e.g. a certain PSM needs a causal domain model) or to task knowledge (a certain PSM generates locally optimal solutions). To our knowledge, there does not exist a library organised following this principle, but work is currently being performed to shed more light on the role of assumptions in libraries for knowledge engineering ([17], [18], [56], [58]).

A last proposal to organise libraries of PSMs is based on a suite of so-called problem types ([27], [28]) (or tasks, for the purpose of this article, tasks and problem types are treated as synonyms). The suite describes problem types according to the way that problems depend on each other. The solution to one problem forms the input to another problem. For example, the output of a prediction task is a certain state, which can form the input to a monitoring task that tries to detect problems, which on their turn can be the input to a diagnosis task. It turns out that these problem dependencies recur in many different tasks. According to this principle, PSMs are indexed under the problem type they can solve. Selection of PSMs in such a library would first identify the problem type involved (or task), and then look at the respective PSMs for this task.

5.3 Selection of PSMs - Assumptions

Whatever the organisational structure of the library, PSMs are used to realise tasks (tasks describe the *what*, PSMs describe the *how*) by applying domain knowledge. Therefore, there are two possible causes why a PSM cannot be applied to solve a particular problem: (1) if its requirements on domain knowledge are not fulfilled, or (2) if it cannot deliver what the task requires, that is, if its competence or functionality is not sufficient for the task. In practical knowledge engineering, there are two respective solutions to this problem. In case the requirements on domain knowledge are not fulfilled, we can assume that they are fulfilled or acquire extra domain knowledge, and still apply the PSM. If the competence of the PSM is not sufficient for the task to be realised, we can weaken the task in such a way that the competence of the PSMs does deliver the (now weakened) task requirement [17].

5.4 Declarative versus Operational Specifications of PSMs

Traditionally, PSMs are described in an operational style. They are described as decomposing a task into a set of subtasks, by introducing their dataflows and knowledge roles, and by defining some control on how to execute the subtasks. However, from the standpoint of reuse these are not the most important aspects. As mentioned earlier, two main aspects decide about the applicability of a PSM in a given application. Whether the competence of the method is able to achieve the goal of the task and whether the domain knowledge required by the method is available. [8] discussed the characterizations of PSMs by their functionality where the functionality is defined in terms of assumptions over available domain knowledge. Meanwhile several papers have appeared providing declarative characterizations of PSMs (e.g. [17], [59], [139], [42]) and we assume that this will become an important line of future works on PSMs.

6 Ontologies

Since the beginning of the nineties ontologies have become a popular research topic investigated by several Artificial Intelligence research communities, including knowledge engineering, natural-language processing and knowledge representation. More recently, the notion of ontology is also becoming widespread in fields such as intelligent information integration, information retrieval on the Internet, and knowledge management. The reason for ontologies being so popular is in large part due to what they promise: a shared and common understanding of some domain that can be communicated across people and computers.

The main motivation behind ontologies is that they allow for sharing and reuse of knowledge bodies in computational form. In the Knowledge Sharing Effort (KSE) project [111], ontologies are put forward as a means to share knowledge bases between various KBSs. The basic idea was to develop a library of reusable ontologies in a standard formalism, that each system developer was supposed to adopt.

6.1 Definition of Ontology

Originally, the term ontology comes from philosophy - it goes as far back as Aristotle's attempt to classify the things in the world - where it is employed to describe the existence of beings in the world. Artificial Intelligence (AI) deals with reasoning about models of the world. Therefore, it is not strange that AI researchers adopted the term ontology to describe what can be (computationally) represented of the world in a program. Many definitions of ontologies have been given in the last decade, but one that characterises best, in our opinion, the essence of an ontology is based on the related definitions in ([74], [22]): *An ontology is a formal, explicit specification of a shared conceptualisation.* A 'conceptualisation' refers to an abstract model of some phenomenon in the world by having identified the relevant concepts of that phenomenon. 'Explicit' means that the type of concepts used, and the constraints on their use are explicitly defined. For example, in medical domains, the concepts are diseases and symptoms, the relations between them are causal and a constraint is that a disease cannot cause itself. 'Formal' refers to the fact that the ontology should be machine readable, which excludes natural language. 'Shared' reflects the notion that an ontology captures consensual knowledge, that is, it is not private to some individual, but accepted by a group.

Almost all ontologies that are nowadays available are concerned with modelling static domain knowledge, as opposed to dynamic reasoning knowledge (e.g., domain models in medicine, power plant, cars, mechanic machines). In its strongest form, an ontology tries to capture universally valid knowledge, independent of its use, a view closely related to its philosophical origin. However, AI researchers quickly gave up this view, because it turned out that specific use of knowledge influenced its modelling and representation. A weaker, but still strong, notion of ontology is CYC's aim to capture human common-sense knowledge [93]. Other researchers aim at capturing domain knowledge, independent of the task or method that might use the knowledge [76].

Within a given domain, an ontology is not just a representation - in a computer - of that domain. An ontology also claims to reflect a certain rate of consensus about the knowledge in that domain. However, in practical ontological engineering research, the definition of ontology has been somewhat diluted, in the sense that taxonomies are considered to be full ontologies. Ontologies differ in two respects from taxonomies as such: they have richer internal structure and reflect some consensus. The question is then of course, consensus between whom? In practice, this question does not have one unique answer; it depends on the context. For example, if a hospital is building up an ontology with knowledge about a particular disease - say AIDS - that can be consulted by all doctors in a hospital, then the consensus should be between the doctors involved. If, on the other hand, a government wants to setup a nation-wide network of bibliographic, ontology-based databases that can be consulted from nearly every terminal with an Internet connection in the country, then the consensus should be nation-wide (i.e. everybody should accept this ontology as workable). In the library example, consensus should be reached about terms such as books, authors, journals, etc., and about axioms such as: if two articles appear in the same journal, they should have the same volume and issue number.

Especially because ontologies aim at consensual domain knowledge, their development is often a cooperative process involving different people, possibly at different locations. People who agree to accept an ontology are said to *commit* themselves to that ontology.

6.2 Ontologies outside Knowledge Engineering

In this article, we are concerned with the role of ontologies in knowledge engineering. That is, how ontologies can help in building KBSs. Any KBS comprises at least two fundamental parts: domain knowledge and problem-solving knowledge. Ontologies mainly play a role in analysing, modelling and implementing the domain knowledge, although they also influence problem-solving knowledge.

Due to the fact that ontologies capture commonly agreed, static knowledge of a particular domain, they are also valuable for other research areas [63]. In natural language applications, ontologies can be used for natural language processing, which is the aim of the Generalised Upper Model ([12], [11]) and of the SENSUS ontology ([138], [86]), or to automatically extract knowledge from (scientific) texts [146]. Wordnet [104] is one of the largest lexical ontologies.

In the database and information retrieval areas, ontologies can be used for interoperability of heterogeneous information sources (databases or information systems): each information source needs to be provided with a wrapper that maps the data-scheme of the source to the ontology ([150], [151], [152]). The retriever only knows about the ontology and not about each individual information source.

Ontologies are interesting candidates to facilitate communication between people in organisations. Ontologies provide the terms, their meanings, their relations and constraints, etc. and in the communication process all participants should commit to these definitions. A currently popular and commercially successful application of this is in knowledge management [62]. Companies realize more and more that the knowledge they possess (aka corporate memory) is of essential importance for successful operation on the market ([87], [141]). Such knowledge should be accessible for the appropriate people and should be maintained to be always up-to-date. It is still an open debate whether corporate memories should be consistent or not. Inconsistent knowledge is sometimes very valuable. In any case, having knowledge about the inconsistency is already very useful information. It has turned out that ontologies, coupled to Intranets are good candidates to improve knowledge management.

6.3 The Role of Ontologies in Knowledge Engineering

Basically, the role of ontologies in the knowledge engineering process is to facilitate the construction of a domain model. An ontology provides a vocabulary of terms and relations with which to model the domain. Depending on how close the domain at hand is to the ontology, the support is different. For instance, if the ontology perfectly suites the domain, then a domain model can be obtained by only filling the ontology with the instances. However, this situation rarely occurs because the nature of an ontology prevents it from being directly applicable to particular domains.

There are several types of ontologies, and each type fulfils a different role in the process of building a domain model. In the following sections, we will discuss different types of ontologies, how to build them in the first place, how to organise them and how to assemble them from smaller ontologies.

6.4 Types of Ontologies

Although the term ontology is widely used in Knowledge Engineering, there are different types of ontologies around. While they all share - to some extent - the underlying idea of capturing explicitly static knowledge about some domain, they also vary considerably. Most researchers agree that it is useful to distinguish between different generality levels of ontologies ([147], [148], [4], [22]):

- *Domain ontologies* capture the knowledge valid for a particular type of domain (e.g. electronic, medical, mechanic, digital domain).
- *Generic ontologies* are valid across several domains. For example, an ontology about mereology (part-of relations) is applicable in many technical domains. Generic ontologies are also referred to as *super theories* [23] and as *core ontologies* [148].
- *Application ontologies* contain all the necessary knowledge for modelling a particular domain (usually a combination of domain and method ontologies) [71].
- *Representational ontologies* do not commit to any particular domain. Such ontologies provide representational entities without stating what should be represented. A well-known representational ontology is the *Frame Ontology* [74], which defines concepts such as frames, slots and slot constraints allowing to express knowledge in an object-oriented or frame-based way.

The ontologies mentioned above all capture static knowledge in a problem-solving independent way. Knowledge Engineering, however, is also concerned with problem-solving knowledge, therefore another useful type of ontology are so-called *method* and *task ontologies* ([58], [137]). Task ontologies provide terms specific for particular tasks (e.g. 'hypothesis' belongs to the diagnosis task ontology), and method ontologies provide terms specific to particular PSMs [71] (e.g. 'correct state' belongs to the Propose-and-Revise method ontology). Task and method ontologies provide a reasoning point of view on domain knowledge. In this way, these ontologies help to solve the 'interaction problem' [33], which states that domain knowledge cannot be independently represented from how it will be used in problem solving, and vice versa. Method and task ontologies enable to make explicit the interaction between problem-solving and domain knowledge through assumptions ([17], [60], [18]).

6.5 Building an Ontology from Scratch

Part of the research on ontology is concerned with envisioning and building enabling technology for large-scale reuse of ontologies at a world-wide level. However, before we can reuse ontologies, they need to be available in the first place. Today, many ontologies are already available, but even more will have to be built in the future. Basically, building an ontology for a particular domain requires a profound analysis, revealing the relevant concepts, attributes, relations, constraints, instances and axioms of that domain. Such knowledge analysis typically results in a taxonomy (an isa hierarchy) of concepts with their attributes, values and relations. Additional information about the classes and their relations to each other, as well as constraints on attribute values for each class, are captured in axioms.

Once a satisfying model of the domain has been built, two things have to be done before it can be considered an ontology. (1) Different generality levels have to distinguished, corresponding to different levels of reusability (see ontology types). (2) The domain model should reflect common understanding or consensus of the domain. Many of the current ontologies are valuable domain models but do not qualify as ontologies because they do not fulfil these two criteria.

6.6 Internal Organisation of an Ontology

In order to enable reuse as much as possible, ontologies should be small modules with high internal coherence and limited amount of interaction between the modules. This requirement and others are expressed in design principles for ontologies ([75], [73], [144], [148]). An ontology is optimal if it satisfies as much as possible all design principles. Notice however that design principles may be contradictory to each other; satisfying one principle may violate another. Some of these design principles are: modularity, internal coherence, extensibility, minimal encoding bias, centre definitions around natural categories, keep the number of theory inclusions to a minimum, and minimal ontological commitment. Most of these principles are understandable from their name. A remark can be made, however, about the 'minimal ontological commitment' principle. Minimal ontological commitment assures maximum reusability, but there is a well-known trade-off between reusability and usability (the more reusable, the less usable and vice versa) [85].

6.7 Constructing Ontologies from Reusable Ontologies

Assuming that the world is full of well-designed modular ontologies, constructing a new ontology is a matter of assembling existing ones. There are several ways to combine ontologies. Here, we will only give the most frequently occurring ones. The simplest way to combine ontologies is through *inclusion*. Inclusion of one ontology into another has the effect that the composed ontology consists of the union of the two ontologies (their classes, relations, axioms). In other words, the starting ontology is extended with the included ontology. Conflicts between names have to be resolved, and a good tool for engineering ontologies should take care of that. For example, a developer building an ontology of a research community might want to include an existing ontology about bibliographic data to model the publications of researchers.

Another way to combine ontologies is by *restriction* [52]. This means that the added ontology only is applied on a restricted subset of what it was originally designed for. [52] gives the example of including an ontology for numbers (where '+' applies to all numbers) in an ontology for integer arithmetic (where ‘+’ only is applied to integers, since all numbers are integers in that world). The last way to assemble ontologies that we discuss here is *polymorphic refinement*, known from object-oriented approaches. Suppose we want to include an ontology about numbers - that defines ’+’ - in two other ontologies. We can include this numbers ontology in one ontology - say about vectors - where we want ‘+’ to work on vectors, and in another ontology - say about strings - where ‘+’ does concatenation of strings.

The KACTUS project [16] was concerned with constructing large ontologies for technical devices through incremental refinement of general ontologies into technical ontologies.

6.8 Specification and Implementation of Ontologies

In order to actually enable sharing of ontologies in electronic form, they have to be implemented in some formal or computer language. Earlier we already pointed out the close relation between ontologies and taxonomies. Traditionally, the AI subfield of knowledge representation (KR) has dedicated much effort in representing taxonomies, and it is therefore not surprisingly that KR languages and techniques are used to implement ontologies. Most KR languages and techniques allow to represent classes (also known as objects, concepts), attributes (aka properties), relations and instances, and have the ‘isa’-relation - which allows for inheritance - built-in as a primitive along with its associated semantics.

However, as we stated earlier, ontologies are more than taxonomies. Ontologies include all relevant constraints between classes, attribute values, instances and relations (i.e. axioms). For example, in some domains the ‘part-of’ relation is fundamental, and an ontology of such domain has to include and define this relation (see the mereological super theory of the PHYSSYS ontology [22]).

Typical AI languages that can be used for implementing ontologies are *description logics* for representing declarative knowledge. Most of such languages support subsumption checking, automatic classification and consistency maintenance. Examples of such description logics include KL-ONE [26], LOOM [97], KRYPTON [25], CYCL [94].

A dedicated language for specifying ontologies is Ontolingua [74], which has become quite popular in the last years. Ontolingua is based on KIF - the Knowledge Interchange Format [67] - which is basically first-order predicate logic extended with meta-capabilities to reason *about* relations. Ontolingua's main aim is to allow sharing and communication of ontologies, and not so much reasoning with them. Therefore, Ontolingua comes with several translators able to generate code in Prolog, CORBA's IDL [114], CLIPS, LOOM [97], KIF, Epikit [66].

7 Conclusion and Related Work

During the last decade research in Knowledge Engineering (KE) resulted in several important achievements which are also relevant for other disciplines like Software Engineering, Information Integration or Knowledge Management. Notable developments are:

- Within the framework of model-based KE, model structures have been defined which clearly separate the different types of knowledge which are important in the context of Knowledge-based Systems (KBSs). The *Expertise Model* is the most prominent example of these models.

- The clear separation of the notions of task, problem-solving method, and domain knowledge provides a promising basis for making the reuse-oriented development of KBSs more feasible.
- The integration of a strong conceptual model is a distinctive feature of formal specification languages in Knowledge Engineering.

Subsequently, we will discuss some relationships between methods in Knowledge Engineering and other disciplines.

7.1 Software Engineering

Analogously to model-based approaches in KE a lot of approaches in Software Engineering consider the development of software systems as a model construction process. E.g. in *Structured Analysis* [155] different types of models are constructed for capturing different aspects of the system under development: functional aspects are specified in data flow diagrams, whereas process aspects are described by using statecharts. In the same way various models are constructed within the *OMT methodology* [125]: e.g. OMT object diagrams correspond to domain models in CommonKADS or MIKE, OMT design models to MIKE design models. Furthermore, a lot of primitives for modeling static system aspects are shared among most approaches, e.g. the notion of classes and instances as well as the embedding of classes in a generalization hierarchy combined with (multiple) inheritance.

We already mentioned the relationships to work on software architectures [132]. Conceptual models of KBSs describe an architecture for a specific class of systems, namely KBSs. Interestingly, both fields currently show the trend to characterize architectures in a declarative way. E.g. [119] defines an architecture in terms of assumptions of its components and [139] defines a class of problem-solving methods in terms of the assumed functionality of its components. Therefore, referring to work on software architectures enables to put work on Knowledge Engineering in a broader context. Reversible, an architecture specifies a problem decomposition (each component deals with a subproblem) and work on software architectures could profit heavily from the large body of work on problem decompositions in Knowledge Engineering.

In KE, model-based approaches exploit the structure of the models to guide the knowledge elicitation process. E.g. the notion of roles in problem-solving methods gives a strong hint concerning what type of knowledge has to be gathered for being able to apply the selected problem-solving method. In a similar way some Software Engineering approaches exploit the structure of system models to guide the acquisition of requirements. E.g. in the *NATURE framework* an approach is described which uses the structure of predefined object system models to generate different types of questions for eliciting system requirements [98].

7.2 Information Integration and Information Services

During the last years a strong demand for information services evolved which are integrated and global in their scope [109]. One specific challenge to be addressed is the development of methods and tools for integrating partially incompatible information sources. In that context the notion of mediators is proposed as a middle layer between information sources and applications [152]. Among others, mediators rely on the notion of ontologies for defining the conceptualization of the underlying information sources. Therefore, methods for constructing and reusing ontologies as described in Section 6 are directly relevant for developing mediators.

In [68], the *Infomaster system* is described which is a generic tool for integrating different types of information sources, like e.g. relational databases or Web pages. Each information source is associated with a wrapper that hides the source specific information structure of the information sources. Internally, Infomaster uses the Knowledge Interchange Format for representing knowledge (compare Section 6). In Infomaster so-called base relations are used for mediating between the conceptual structure of the information sources and the user applications. The collection of base relations may be seen as a restricted domain ontology for integrating the different heterogeneous information sources.

In the meantime ontologies are also used for supporting the semantic retrieval of information from the World-Wide Web. The *SHOE approach* [95] proposes to annotate Web pages with ontological information which can then be exploited for answering queries. Thus, the syntactic based retrieval of information from the Web as known from the various Web search engines is replaced by a semantic based retrieval process. A further step is taken in the *Ontobroker project* [57] which proposes to use a more expressive ontology combined with a corresponding inference mechanism. Thus, the search metaphor of SHOE is replaced by an inference metaphor for retrieving information from the Web since the inference mechanism can use complex inferences as part of the query answering process.

7.3 Knowledge Management

During the last years more and more companies became aware of the fact that knowledge is one of their important assets. Therefore, active management of knowledge is nowadays considered as an important means to achieve an enterprise's effectiveness and competitiveness [1]. Overall consensus is that knowledge management requires an interdisciplinary approach including technical support by IT technology, but also for example human resource management ([88], [64]).

A central technical aspect of knowledge management is the construction and maintenance of an *Organizational Memory* as a means for knowledge conservation, distribution and reuse [149]. Typically, the knowledge within an Organizational Memory will be a combination of informal, semi-formal, and formal knowledge. Since enterprise-wide Organizational Memories will evolve to very large collections of rather diverse knowledge elements, novel methods for accessing and distributing these knowledge elements are required. In that context, ontologies may be exploited for defining the concepts which are used for organizing and structuring the knowledge elements in the Organizational Memory. Furthermore, such ontologies may be used for supporting the users in finding relevant knowledge, e.g. by offering the appropriate concepts for posing queries [88]. Nevertheless, one should be aware, that although a considerable effort is put into knowledge management, the construction and application of Organizational Memories is still in a very early stage.

In general, we can see that recent developments in several disciplines rely on extracting, modeling and exploiting various types of knowledge in order to address the demands which arise among others from the growing complexity of applications. Methods and concepts from Knowledge Engineering are certainly one of the promising approaches for developing solutions which are able to meet these demands.

Acknowledgement

Thanks are due to Stefan Decker for valuable comments on a draft version of the paper. Rainer Perkuhn provided valuable editorial support.

Richard Benjamins was partially supported by the Netherlands Computer Science Research Foundation with financial support from the Netherlands Organisation for Scientific Research (NWO), and by the European Commission through a Marie Curie Research Grant (TMR).

References

[1] A. Abecker, S. Decker, K. Hinkelmann, and U. Reimer, *Proc. Workshop Knowledge-based Systems for Knowledge Management in Enterprises, 21st Annual German Conference on AI (KI'97)*, Freiburg, 1977. URL: http://www.dfki.uni-kl.de/km/ws-ki-97.html

[2] M. Aben, Formally Specifying Re-usable Knowledge Model Components, *Knowledge Acquisition* 5:119-141, 1993.

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

[4] A. Abu-Hanna, *Multiple Domain Models in Diagnostic Reasoning,* PhD Thesis, University of Amsterdam, Amsterdam 1994.

[5] J. Angele, S. Decker, R. Perkuhn, and R. Studer, Modeling Problem-Solving Methods in NewKARL, in: *Proc. of the 10th Knowledge Acquisition for Knowledge-based Systems Workshop (KAW'96)*, Banff, 1996.

[6] J. Angele, D. Fensel, and R. Studer, Developing Knowledge-Based Systems with MIKE, to appear in *Journal of Automated Software Engineering*, 1998.

[7] J. Angele, D. Fensel, and R. Studer, Domain and Task Modeling in MIKE, in: A. Sutcliffe et al., eds., *Domain Knowledge for Interactive System Design*, Chapman & Hall, 1996.

[8] H. Akkermans, B. Wielinga, and A.Th. Schreiber, Steps in Constructing Problem-Solving Methods, in: N. Aussenac et al., eds., *Knowledge Acquisition for Knowledge-based Systems, 7th European Workshop (EKAW'93)*, Toulouse, Lecture Notes in AI 723, Springer-Verlag, 1993.

[9] Barros, L. Nunes de, J. Hendler, and V. R. Benjamins, Par-KAP: A Knowledge Acquisition Tool for Building Practical Planning System, in: *Proc. of the 15th International Joint Conference on Artificial Intelligence (IJCAI ´97)*, pages 1246-1251, Japan, 1997, Morgan Kaufmann Publishers, Inc.

[10] Barros, L. Nunes de, A. Valente, and V. R. Benjamins, Modeling Planning Tasks, in *Third International Conference on Artificial Intelligence Planning Systems* (*AIPS-96*), pages 11-18. American Association of Artificial Intelligence (AAAI), 1996.

[11] J. A. Bateman, On the Relationship Betweem Ontology Construction and Natural Language: A Socia-semiotic View, *International Journal of Human-Computer Studies*, 43(2/3):929-944, 1995.

[12] J. A. Bateman, B. Magini, and F. Rinaldi, The Generalized Upper Model, in: N. J. I. Mars, editor, *Working Papers European Conference on Artificial Intelligence ECAI'94 Workshop on Implemented Ontologies*, pages 35-45, Amsterdam, 1994.

[13] V. R. Benjamins, *Problem Solving Methods for Diagnosis*. PhD Thesis, University of Amsterdam, Amsterdam, The Netherlands, 1993.

[14] V. R. Benjamins, Problem-solving Methods for Diagnosis and Their Role in Knowledge Acquisition, *International Journal of Expert Systems: Research and Applications*, 8(2):93-120, 1995.

[15] V. R. Benjamins and M.Aben, Structure-preserving KBS Development Through Reusable Libraries: A Case-study in Diagnosis, *International Journal of Human-Computer Studies*, 47:259-288, 1997.

[16] J. Benjamin, P. Borst, H. Akkermans, and B. Wielinga, Ontology Construction for Technical Domains, in: N. Shadbolt et al., eds., *Advances in Knowledge Acquisition*, Lecture Notes in Artificial Intelligence 1076, Springer-Verlag, Berlin, 1996.

[17] V. R. Benjamins, D. Fensel, and R. Straatman, Assumptions of Problem-solving Methods and Their Role in Knowledge Engineering, in: W. Wahlster, editor, *Proc. ECAI-96*, pages 408-412. J. Wiley & Sons, Ltd., 1996.

[18] V. R. Benjamins and C. Pierret-Golbreich, Assumptions of Problem-solving Methods, in: N. Shadbolt et al., eds., Lecture Notes in Artificial Intelligence, 1076, *9th European Knowledge Acquisition Workshop,*

EKAW-96, pages 1-16, Berlin, 1996. Springer-Verlag.

[19] P. Beys, V. R. Benjamins, and G. van Heijst, Remedying the Reusability-usability Tradeoff for Problem-solving Methods, in: B. R. Gaines and M. A. Musen, editors, *Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, pages 2.1-2.20, Alberta, Canada, 1996. SRDG Publications, University of Calgary. http://ksi.cpsc.ucalgary.ca:80/KAW/KAW96/KAW96Proc.html.

[20] W. Birmingham and G. Klinker, Knowledge Acquisition Tools with Explicit Problem-Solving Models, *The Knowledge Engineering Review* 8, 1 (1993), 5-25.

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

[22] W. N. Borst, *Construction of Engineering Ontologies*, PhD Thesis, University of Twente, Enschede, 1997.

[23] W. N. Borst and J. M. Akkermans, Engineering Ontologies, *International Journal of Human-Computer Studies*, 46(2/3):365-406, 1997.

[24] J. P. Bowen and M. G. Hinchey, Ten Commands of Formal Methods, *IEEE Computer*, 28(4):56-63, 1995.

[25] R. J. Brachman, V. P. Gilbert, and H. J. Levesque, An Essential Hybrid Reasoning System: Knowledge and Symbol Level Accounts of KRYPTON, in: *Proceedings IJCAI-85*, 1985.

[26] R. J. Brachman and J. Schmolze, An Overview of the KL-ONE Knowledge Representation System, *Cognitive Science*, 9(2), 1985.

[27] J. Breuker, Components of Problem Solving and Types of Problems, 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.

[28] J. Breuker, A Suite of Problem Types, in: J. A. Breuker and W. van de Velde, eds., *The CommonKADS Library For Expertise Modelling*, IOS Press, Amsterdam, 1994.

[29] J. A. Breuker and W. van de Velde, eds., *The CommonKADS Library For Expertise Modelling*, IOS Press, Amsterdam, 1994.

[30] M. L. Brodie, On the Development of Data Models, in: Brodie et al., eds., *On Conceptual Modeling*, Springer-Verlag, Berlin, 1984.

[31] A. G. Brooking, The Analysis Phase in Development of Knowledge-Based Systems, in: W. A. Gale, ed., *AI and Statistic*, Addison-Wesley Publishing Company, Reading, Massachusetts, 1986.

[32] T. Bylander, D. Allemang, M.C. Tanner, and J.R. Josephon, The Computational Complexity of Abduction, *Artificial Intelligence* 49, 1991.

[33] T. Bylander and B. Chandrasekaran, Generic Tasks in Knowledge-based Reasoning: The Right Level of Abstraction for Knowledge Acquisition, in: B. Gaines and J. Boose, eds., *Knowledge Acquisition for Knowledge Based Systems*, Vol. 1, Academic Press, London, 1988.

[34] T. Bylander and S. Mittal, CSRL, A Language for Classificatory Problem Solving, *AI Magazine* 8, 3, 1986, 66-77.

[35] B. Chandrasekaran, Design Problem Solving: A Task Analysis, *AI Magazine*, 11:59-71, 1990.

[36] B. Chandrasekaran, Generic Tasks in Knowledge-based Reasoning: High-level Building Blocks for Expert System Design, *IEEE Expert* 1, 3 1986, 23-30.

[37] B. Chandrasekaran, T. R. Johnson, and J. W. Smith, Task Structure Analysis for Knowledge Modeling, *Communications of the ACM* 35, 9 (1992), 124—137.

[38] W.J. Clancey, The Epistemology of a Rule-Based Expert System - a Framework for Explanation, *Artificial Intelligence* 20 (1983), 215-251.

[39] W.J. Clancey, Heuristic Classification, *Artificial Intelligence* 27 (1985), 289-350.

[40] W.J. Clancey, From Guidon to Neomycin and Heracles in Twenty Short Lessons, in: A. van Lamsweerde, ed., *Current Issues in Expert Systems,* Academic Press, 1987.

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

[42] F. Cornelissen, C.M. Jonker, and J. Treur: Compositional Verification of Knowledge-based Systems: A Case Study for Diagnostic Reasoning, in: E. Plaza and R. Benjamins, eds., *Knowledge Acquisition, Modeling, and Management, 10th European Workshop (EKAW'97)*, Sant Feliu de Guixols, Lecture Notes in Artificial Intelligence 1319, Springer-Verlag, 1997.

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

[44] R. Davis, B. Buchanan, and E.H. Shortcliffe, Production Rules as a Representation for a Knowledge-base Consultation Program, *Artificial Intelligence* 8 (1977), 15-45.

[45] S. Decker, M. Daniel, M. Erdmann, and R. Studer, An Enterprise Reference Scheme for Integrating Model-based Knowledge Engineering and Enterprise Modeling, in: E. Plaza and R. Benjamins, eds., *Knowledge Acquisition, Modeling, and Management, 10th European Workshop (EKAW'97)*, Sant Feliu de Guixols, Lecture Notes in Artificial Intelligence 1319, Springer-Verlag, 1997.

[46] H. Ehrig and B. Mahr, eds., *Fundamentals of Algebraic Specifications 1*, Springer-Verlag, Berlin, 1985.

[47] H. Ehrig and B. Mahr, eds., *Fundamentals of Algebraic Specifications 2*, Springer-Verlag, Berlin, 1990.

[48] H. Eriksson, A Survey of Knowledge Acquisition Techniques and Tools and their Relationship to Software Engineering, *Journal of Systems and Software* 19, 1992, 97-107.

[49] Epistemics, PCPACK Portable KA Toolkit, 1995.

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

[51] H. Eriksson, Y. Shahar, S.W. Tu, A.R. Puerta, and M.A. Musen, Task Modeling with Reusable Problem-Solving Methods, *Artificial Intelligence* 79 (1995), 293-326.

[52] A. Farquhar, R. Fikes, and J. Rice, The Ontolingua Server: A Tool for Collaborative Ontology Construction, *International Journal of Human-Computer Studies*, 46:707-728, 1997.

[53] D. Fensel, *The Knowledge Acquisition and Representation Language KARL*, Kluwer Academic Publ., Boston, 1995.

[54] D. Fensel, Formal Specification Languages in Knowledge and Software Engineering, *The Knowledge Engineering Review* 10, 4, 1995.

[55] D. Fensel, J. Angele, and R. Studer, The Knowledge Acquisition and Representation Language KARL, to appear in *IEEE Transactions on Knowledge and Data Engineering*.

[56] D. Fensel and V. R. Benjamins, Assumptions in Model-based Diagnosis, in: B. R. Gaines and M. A. Musen, editors, *Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, pages 5.1-5.18, Alberta, Canada, 1996. SRDG Publications, University of Calgary. http://ksi.cpsc.ucalgary.ca:80/KAW/KAW96/KAW96Proc.html.

[57] D. Fensel, S. Decker, M. Erdmann, and R. Studer, Ontobroker: Transforming the WWW into a Knowledge Base, submitted for publication.

[58] D. Fensel and R. Groenboom, Specifying Knowledge-based Systems with Reusable Components, in: *Proceedings 9th Int. Conference on Software Engineering and Knowledge Engineering (SEKE '97)*, pages 349-357, Madrid, 1997.

[59] D. Fensel and A. Schönegge, Using KIV to Specify and Verify Architectures of Knowledge-Based Systems, in: *Proceedings of the 12th IEEE International Conference on Automated Software Engineering (ASEC-97)*, Incline Village, Nevada, November 1997.

[60] D. Fensel and R. Straatman, The Essence of Problem-Solving Methods: Making Assumptions for Efficiency Reasons, in: N. Shadbolt et al., eds., *Advances in Knowledge Acquisiiton,* Lecture Notes in Artificial Intelligence (LNAI) 1076, Springer-Verlag, Berlin, 1996.

[61] D. Fensel and F. van Harmelen, A Comparison of Languages which Operationalize and Formalize KADS Models of Expertise, *The Knowledge Engineering Review* 9, 2, 1994.

[62] M. S. Fox, J. Chionglo, and F. Fadel, A Common-sense Model of the Enterprise, in*Proceedings of the Industrial Engineering Research Conference*, 1993.

[63] N. Fridman-Noy and C.D. Hafner, The State of the Art in Ontology Design,*AI Magazine*, 18(3):53-74, 1997.

[64] B. Gaines et al., eds., *Working Notes AAAI-97 Spring Symposium Artificial Intelligence in Knowledge Management*, Stanford, March 1977.

[65] B. Gaines and M.L.G. Shaw, New Directions in the Analysis and Interactive Elicitation of Personal Construct Systems, *Int. J. Man-Machine Studies* 13 (1980), 81-116.

[66] M.R. Genesereth, ed., *The Epikit Manual*, Epistmemics, Inc, Palo Alto, CA, 1992.

[67] M.R. Genesereth and R.E. Fikes, Knowledge Interchange Format, Version 3.0, Reference Manual. Technical Report, Logic-92-1, Computer Science Dept., Stanford University, 1992. http://

www.cs.umbc.edu/kse/.

[68] M.R. Genesereth, A.M. Keller, and O.M. Duschka, Infomaster: An Information Integration System, in: *Proc. ACM SIGMOD Conference*, Tucson, 1997.

[69] J.H. Gennari, R.B. Altman, and M.A. Musen, Reuse with PROTÉGÉ-II: From Elevators to Ribosomes, in: *Proceedings of the Symposium on Software Reuse*, Seattle, 1995.

[70] J.H. Gennari, A.R. Stein, and M.A. Musen, Reuse for Knowledge-based Systems and CORBA Components, in: *Proc. of the 10th Knowledge Acquisition for Knowledge-based Systems Workshop*, Banff, 1996.

[71] J.H. Gennari, S.W. Tu, T.E. Rothenfluh, and M.A. Musen, Mappings Domains to Methods in Support of Reuse, *Int. J. on Human-Computer Studies* 41 (1994), 399-424.

[72] Y. Gil and C. Paris, Towards Method-independent Knowledge Acquisition, *Knowledge Acquisition* 6, 2 (1994), 163-178.

[73] A. Gomez-Perez, A. Fernandez, and M. De Vicente, Towards a Method to Conceptualize Domain Ontologies, in: *Working Notes of the Workshop on Ontological Engineering, ECAI'96*, pages 41-52. ECCAI, 1996.

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

[75] T. R. Gruber, Towards Principles for the Design of Ontologies used for Knowledge Sharing, *International Journal of Human-Computer Studies*, 43:907-928, 1995.

[76] N. Guarino, Formal Ontology, Conceptual Analysis and Knowledge Representation, *International Journal of Human-Computer Studies*, 43(2/3):625-640, 1995.

[77] D. Harel, Dynamic Logic, in: D. Gabby et al., eds., *Handook of Philosophical Logic*, vol. II, *Extensions of Classical Logic*, Publishing Company, Dordrecht (NL), 1984.

[78] F. van Harmelen and M. Aben, Structure-preserving Specification Languages for Knowledge-based Systems, *International Journal of Human-Computer Studies* 44, 1996.

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

[80] F. van Harmelen and D. Fensel, Formal Methods in Knowledge Engineering, *The Knowledge Engineering Review* 9, 2, 1994.

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

[82] C. B. Jones, *Systematic Software Development Using VDM*, 2nd ed., Prentice Hall, 1990.

[83] G.A. Kelly, *The Psychology of Personal Constructs*, Norton, New York, 1955.

[84] M. Kifer, G. Lausen, and J. Wu, *Logical Foundations of Object-Oriented and Frame-Based Languages*, *Journal of the ACM* 42 (1995), 741-843.

[85] G. Klinker, C. Bhola, G. Dallemagne, D. Marques, and J. McDermott, Usable and Reusable Programming Constructs, *Knowledge Acquisition*, 3:117-136, 1991.

[86] K. Knight and S. Luk, Building a Large Knowledge Base for Machine Translation, in:*Proc. AAAI-94*, Seattle, 1994.

[87] O. Kühn, An Ontology for the Conservation of Corporate Knowledge About Crankshaft Design, in: N. J. I. Mars, editor,*Working Papers European Conference on Artificial Intelligence ECAI'94 Workshop on Implemented Ontologies*, pages 141-152, Amsterdam, 1994.

[88] O. Kühn and A. Abecker, Corporate Memories for Knowledge Management in Industrial Practice: Prospects and Challenges, *J. of Universal Computer Science* 3, 8 (August 1977), Special Issue on Information Technology for Knowledge Management, Springer Science Online. URL: http://www.iicm.edu/jucs_3_8/corporate_memories_for_knowledge.

[89] 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.

[90] D. Landes and R. Studer, The Treatment of Non-Functional Requirements in MIKE, in: W. Schaefer et al., eds., *Proc. of the 5th European Software Engineering Conference (ESEC´95)*, Sitges, Lecture Notes in Computer Science 989, Springer-Verlag, 1995.

[91] I. van Langevelde, A. Philipsen, and J. Treur, Formal Specification of Compositional Architectures, in:

Proceedings of the 10th European Conference on Artificial Intelligence (ECAI-92), Vienna, Austria, August, 1992.

[92] I. van Langevelde, A. Philipsen, and J. Treur, A Compositional Architecture for Simple Design Formally Specified in DESIRE, in: J. Treur and Th. Wetter, eds., *Formal Specification of Complex Reasoning Systems*, Ellis Horwood, New York, 1993.

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

[94] D. B. Lenat and R. V. Guha*Representation and Inference in the Cyc Projec,* Addison-Wesley, 1990.

[95] S. Luke, L. Spector, D. Rager, and J. Hendler, Ontology-based Web Agents, in: *Proc. 1st Int. Conf. on Autonomous Agents*, 1977.

[96] T. J. Lydiard, Overview of Current Practice and Reserach Initiatives for the Verification and Validation of KBS, *The Knowledge Engineering Review*, 7(2), 1992.

[97] R. MacGregor, Inside the LOOM Classifier,*SIGART Bulletin*, 2(3):70-76, June 1991.

[98] N.A.M. Maiden, Acquiring Requirements: a Domain-specific Approach, in: A. Sutcliffe et al., eds., *Domain Knowledge for Interactive System Design*, Chapman & Hall, 1996.

[99] S. Marcus, ed., *Automating Knowledge Acquisition for Experts Systems*, Kluwer Academic Publisher, Boston, 1988.

[100] S. Marcus, SALT: A Knowledge Acquisition Tool for Propose-and-Revise Systems, in: S. Marcus, ed., *Automating Knowledge Acquisition for Experts Systems*, Kluwer Academic Publisher, Boston, 1988.

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

[102] J. McDermott, Preliminary Steps toward a Taxonomy of Problem-solving Methods, in: S. Marcus, ed., *Automating Knowledge Acquisition for Experts Systems*, Kluwer Academic Publisher, Boston, 1988.

[103] P. Meseguer and A. D. Preece, Verification and Validation of Knowledge-Based Systems with Formal Specifications, *The Knowledge Engineering Review* 10(4), 1995.

[104] G. A. Miller, WORDNET: An Online Lexical Database, *International Journal of Lexicography*, 3(4):235-312, 1990.

[105] B. G. Milnes, A Specification of the Soar Cognitive Architecture in Z, Research Report CMU-CS-92-169, School of Computer Science, Carnegie Mellon University, Pittsburg, PA, 1992.

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

[107] E. Motta and Z. Zdrahal, Parametric Design Problem Solving, in: B. R. Gaines and M. A. Musen, editors, *Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, pages 9.1-9.20, Alberta, Canada, 1996. SRDG Publications, University of Calgary. http://ksi.cpsc.ucalgary.ca:80/KAW/KAW96/KAW96Proc.html.

[108] M.A. Musen, An Overview of Knowledge Acquisition, in: J.-M. David et al., eds., *Second Generation Expert Systems*, Springer-Verlag, 1993.

[109] J. Myplopoulos and M. Papazoglu, Cooperative Information Systems, Guest Editors' Introduction, *IEEE Intelligent Systems* 12, 5 (September/October 1997), 28-31.

[110] B. Nebel, Artificial Intelligence: A Computational Perspective, in: G. Brewka, ed., *Principles of Knowledge Representation*, CSLI Publications, Studies in Logic, Language and Information, Stanford, 1996.

[111] R. Neches, R. E. Fikes, T. Finin, T. R. Gruber, T. Senator, and W. R. Swartout, Enabling Technology for Knowledge Sharing, *AI Magazine*, 12(3):36-56, 1991.

[112] S. Neubert, Model Construction in MIKE, in: N. Aussenac et al., eds., *Knowledge Acquisition for Knowledge-based Systems, Proc. 7th European Workshop (EKAW'93)*, Toulouse, Lecture Notes in Artificial Intelligence 723, Springer-Verlag, 1993.

[113] A. Newell, The Knowledge Level, *Artificial Intelligence* 18, 1982, 87-127.

[114] R. Orfali, D. Harkey, and J. Edwards, editors*The Essential Distributed Objects Survival Guide*, John Wiley & Sons, New York, 1996.

[115] K. Orsvärn,*Knowledge Modelling with Libraries of Task Decomposition Methods*, PhD Thesis, Swedish Institute of Computer Science, 1996.

[116] K. Orsvärn, Principles for Libraries of Task Decomposition Methods - Conclusions from a Case-study,

in: N. Shadbolt et al., eds., *Advances in Knowledge Acquisiiton,* Lecture Notes in Artificial Intelligence 1076, Springer-Verlag, Berlin, 1996.

[117] C. Pierret-Golbreich and X. Talon: An Algebraic Specification of the Dynamic Behaviour of Knowledge-Based Systems, *The Knowledge Engineering Review,* 11(2), 1996.

[118] J. Penix and P. Alexander, Toward Automated Component Adaption, in: *Proceedings of the 9th International Conference on Software Engineering & Knowledge Engineering (SEKE-97)*, Madrid, Spain, June 18-20, 1997.

[119] J. Penix, P. Alexander, and K. Havelund, Declarative Specifications of Software Architectures, in: *Proceedings of the 12th IEEE International Conference on Automated Software Engineering (ASEC-97)*, Incline Village, Nevada, November 1997.

[120] R. Plant and A. D. Preece, Special Issue on Verification and Validation, *International Journal of Human-Computer Studies* (*IJHCS*), 44, 1996.

[121] 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, Lecture Notes in Artificial Intelligence 723, Springer-Verlag, 1993.

[122] A. D. Preece, Foundations and Applications of Knowledge Base Verification, *International Journal of Intelligent Systems*, 9, 1994.

[123] 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, *Knowledge Acquisition* 4, 1992, 171-196.

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

[125] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen, *Object-Oriented Modelling and Design*. Prentice Hall, Englewood Cliffs, New Jersey, 1991.

[126] F. Saltor, M.G. Castellanos, and M. Garcia-Solaco, Overcoming Schematic Discrepancies in Interoperable Databases, in: D.K. Hsiao et al., eds., *Interoperable Database Systems (DS-5)*, IFIP Transactions A-25, North-Holland, 1993.

[127] 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.

[128] A.Th. 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.

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

[130] N. Shadbolt, E. Motta, and A. Rouge, Constructing Knowledge-based Systems, *IEEE Software* 10, 6, 34-38.

[131] 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, Springer-Verlag, 1992.

[132] M. Shaw and D. Garlan, *Software Architecture: Perspectives on an Emerging Discipline*, Prentice Hall, 1996.

[133] D. R. Smith, Towards a Classification Approach to Design, in: *Proceedings of the 5th International Conference on Algebraic Methodology and Software Technology (AMAST-96)*, Munich, Germany, July, 1996.

[134] J. W. Spee and L. in ‘t Veld, The Semantics of K_{BS}SF: A Language For KBS Design, *Knowledge Acquisition* 6, 1994.

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

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

[137] R. Studer, H. Eriksson, J.H. Gennari, S.W. Tu, D. Fensel, and M.A. Musen, Ontologies and the Configuration of Problem-Solving Methods, in: *Proc. of the 10th Knowledge Acquisition for Knowledge-based Systems Workshop*, Banff, 1996.

[138] B. Swartout, R. Patil, K. Knight, and T. Russ, Toward Distributed Use of Large-scale Ontologies, in:

B. R. Gaines and M. A. Musen, editors,*Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, pages 32.1-32.19, Alberta, Canada, 1996. SRDG Publications, University of Calgary. http://ksi.cpsc.ucalgary.ca:80/KAW/KAW96/KAW96Proc.html.

[139] A. ten Teije, *Automated Configuration of Problem Solving Methods in Diagnosis*, PhD Thesis, University of Amsterdam, Amsterdam, The Netherlands, 1997.

[140] P. Terpstra, G. van Heijst, B. Wielinga, and N. Shadtbolt, Knowledge Acquisition Support through Generalised Directive Models, in: J.-M. David et al., eds., S*econd Generation Expert Systems*, Springer-Verlag, Berlin, 1993.

[141] Tove, Manual of the Toronto Virtual Enterprise, Technical Report, University of Toronto, 1995. Available at: www.ie.utoronto.ca/EIL/tove/ontoTOC.html.

[142] J. Treur, Temporal Semantics of Meta-Level Architectures for Dynamic Control of Reasoning, in: L. Fribourg et al., eds., *Logic Program Synthesis and Transformation - Meta Programming in Logic, Proceedings of the 4th International Workshops, LOPSTER-94 and META-94*, Lecture Notes in Computer Science 883, Springer Verlag-Berlin, 1994.

[143] J. Treur and Th. Wetter, eds., *Formal Specification of Complex Reasoning Systems*, Ellis Horwood, New York, 1993.

[144] M. Uschold and M. Gruninger, Ontologies: Principles, Methods, and Applications, *Knowledge Engineering Review*, 11(2):93-155, 1996.

[145] A. Valente and C. Löckenhoff, Organization as Guidance: A Library of Assessment Models, in: N. Aussenac et al., eds., *Knowledge Acquisition for Knowledge-based Systems, Proc. 7th European Workshop (EKAW'93),* Toulouse, Lecture Notes in Artificial Intelligence 723, Springer-Verlag, 1993.

[146] P. E. van de Vet, P.-H. Speel, and N. J. I. Mars, The Plinius Ontology of Ceramic Materials, in: N. J. I. Mars, editor, *Working Papers European Conference on Artificial Intelligence ECAI'94 Workshop on Implemented Ontologies*, pages 187-206, Amsterdam, 1994.

[147] G. van Heijst,*The Role of Ontologies in Knowledge Engineering*, PhD Thesis, University of Amsterdam, May 1995.

[148] G. van Heijst, A. Th. Schreiber, and B. J. Wielinga, Using Explicit Ontologies in KBS Development, *International Journal of Human-Computer Studies*, 46(2/3):183-292, 1997.

[149] G. van Heijst, R. van der Spek, and E. Kruizinga, Organizing Corporate Memories, in: *Proc. of the 10th Knowledge Acquisition for Knowledge-based Systems Workshop*, Banff, 1996.

[150] G. Wiederhold, Mediators in the Architecture of Future Information Systems,*IEEE Computer*, 25(3):38-49, 1992.

[151] G. Wiederhold, Intelligent Integration of Information,*Journal of Intelligent Information Systems*, Special Issue on Intelligent Integration of Information, 1996.

[152] G. Wiederhold and M. Genesereth, The Conceptual Basis for Mediation Services, *IEEE Intelligent Systems* 12, 5 (September/October 1997), 38-47.

[153] B.J. Wielinga, A.Th. Schreiber, and J.A. Breuker, KADS: A Modelling Approach to Knowledge Engineering, *Knowledge Acquisition* 4, 1 (1992), 5-53.

[154] M. Wirsing, Algebraic Specification, in: J. van Leeuwen, ed., *Handbook of Theoretical Computer Science*, Elsevier Science Publ., 1990.

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