A Comparison of Two Approaches to Model-based Knowledge Acquisition

Dieter Fensel (+) and Karsten Poeck (*)

(+) Institut für Angewandte Informatik und Formale Beschreibungsverfahren
University of KARLsruhe, 76128 Karlsruhe, Germany
phone: 49-721-6084754, fax: 49-721-693717
e-mail: fensel@aifb.uni-karlsruhe.de

(*) Lehrstuhl für Informatik VI
University of Würzburg, Allesgrundweg 12, 97218 Gerbrunn, Germany
phone: 49-931-70561 18, fax: 49-931-7056120
e-mail: poeck@informatik.uni-wuerzburg.de

Abstract. This paper discusses and compares two different approaches to model-based knowledge acquisition. That is, we regard the Model-based and Incremental Knowledge Engineering (MIKE) approach and the Configurable Role-limiting Method approach (CRLM). MIKE is based on the distinction of different phases in the software development process. It uses the formal and operational knowledge specification language KARL allowing a precise and unique description of a model of expertise which is the outcome of the analysis phase. CRLM is based on the role-limiting method approach. Role limiting shells are implementations of strong problem-solving methods and substantially simplify knowledge acquisition through guidance by predefined models of problem-solving and by sophisticated graphical user interfaces. The main disadvantages, namely inflexibility and brittleness, are to some degree overcome by the CRLM where the shell's problem-solving methods are split into smaller parts, which can then be reconfigured allowing the integration of new methods or other method combinations. Although these two approaches are often discussed as contradictory, we, however, experienced that both approaches complete each other very well. As an outcome of our comparison, we outline topics of future research for both approaches.

Introduction

In the paper, we discuss and compare two different approaches to *model-based* knowledge acquisition. Both approaches are *model-based* in the sense that they *explicitly distinguish different types of knowledge* and use *generic problem-solving methods as the behaviour model* of an expert system. Apart from their similarities, the two approaches differ significantly in their underlying principles and points of interest.

Model-based and Incremental Knowledge Engineering (MIKE) [AFL+93] is strongly influenced by the results of the KADS and CommonKADS projects [SWB93] and works in the domain of software engineering and information system design (cf. [AFS90]). It is based on the distinction of different phases like analysis, design, and implementation, in the software development process. An important means of MIKE is

the *formal and operational knowledge specification language KARL* (cf. [FAL91], [Fen93a], [AFS94]), which allows a precise and unequivocal description of a model of expertise as the result of the analysis phase.

Configurable Role-limiting Method (CRLM) [PoG93] is based on the role-limiting method approach (see [Mar88], [McD88]). Role-limiting shells are implementations of strong problem-solving methods and substantially simplify knowledge acquisition through guidance by the given model of problem-solving. These shells pre-define the roles that knowledge can play during the problem-solving process and completely fix the knowledge representation scheme for the method such that the expert only has to instantiate given generic concepts. In most of these shells the expert is supported by a sophisticated graphical user interface for knowledge acquisition. Their main disadvantages, that is, their inflexibility and brittleness, are to some degree overcome by the CRLM. In this approach, the problem-solving methods of the shell are split into smaller parts, which can then be reconfigured allowing the integration of new methods or new method combinations. The corresponding knowledge acquisition components can be generated from a declarative description [Flo84]. CLRM tries to preserve the advantages of RLMs such as strong knowledge acquisition support and rapid prototyping, while extending their scope by being more adaptable and therefore less brittle.

The two examined approaches also reflect the two current main streams of research in knowledge acquisition. On the on hand, there are approaches like *KADS* and *CommonKADS* which view the knowledge engineering process of a process of building multiple models. On the other hand, approaches like *PROTÉGÉ-II* [PET+92] or *KREST* [Ste93] aim much stronger at an immediate implementation of a knowledge-based system. Although these different approaches are often discussed as contradictory, we experienced that both approaches complement each other very well. As both approaches emphasize different aspects in the development process of a knowledge-based system, their combination adds to the power of the achieved results. It was already shown in [FEM+93] how an implementation of the board-game method, that is, a role-limiting method, can be combined with a semiformal and formal description by using KARL. [PFL+94] shows a successful solution of the elevator-design problem (i.e., the Sisyphus´93 problem) based on the fruitful combination of the two approaches. In fact, the successful combination of both approaches by solving the elevator-design problem encouraged us to compare both approaches in more detail and depth. The purpose of our study is to get a better insight into the different assumptions implicitly underlying the different approaches by contrasting the two approaches. Additionally, we try to overcome some conceptual miss-matches because a term like knowledge base is associated with different meanings by the two approaches. Finally, we will show how both approaches fit together, that is, how both approaches can be improved by overtaking results of the competitor.

We identified four dimensions for the comparison of both approaches. In each item we *make the different assumptions explicit which underlie the two approaches to model-based knowledge acquisition*. First, we ask how both approaches bridge the gap between informal requirements and implemented systems which meets these requirements. CRLM tries more or less to bridge this gap in one go whereas MIKE makes a walk of several steps. Second, we discuss how both approaches view the difference of problem-solving methods and domain knowledge. The Role-limiting method views problem methods as fixed and implemented whereas MIKE views the problem-solving method as part of the knowledge and aims at its declarative description. The CRLM approach

converges into the direction of MIKE but does still regard the problem-solving method not as part of the knowledge base. Third, we ask to what degree both approaches try to mechanize the knowledge engineering process. Whereas MIKE view the knowledge engineer as a necessity CRLM aims on excluding him from the process by offering powerful tool support. The final dimension of our comparison regards the different concepts of reuse which underlie both approaches. In the conclusion, we answer the question whether both approaches are contradictory or rather complementary. It is shown how both approaches converge together starting from very contrary points. Libraries of reusable mechanisms or problem-solving methods seem to unify both approaches. In fact, they require the combination of both approaches.

Our comparison of different approaches in knowledge engineering is not the first one. An early attempt to survey and classify most approaches in knowledge acquisition was given by [Boo88]. Methodologically close in spirit to this work are [KLV90], [NPB+91], [Lin93], and [FeH94]. [KLV90] survey four approaches to model-based knowledge acquisition and extract three common hypotheses which are now common places in knowledge acquisition. [NPB+91] tries to give a complete classification on knowledge acquisition and [Lin93] focuses on solutions of the Sisyphus-I problem. [FeH94] discuss and compare eight formal knowledge specification languages. In this paper we have chosen a different methodological point of view:

- First, we neither aim to give a representative survey on all existing knowledge acquisition approaches nor to classify them. Instead, we chose two different paradigms in knowledge engineering and prototypical approaches for each of them. Therefore, we did not try to abstract and aggregate general features but instead we tried to elicit differences and mutualities of both approach in detail.
- Second, we use a common case study to understand mutualities and differences of both approaches in detail (like [Nwa93] does it for KADS and Generic Tasks). In fact in [PFL+94] we modelled a Sisyphus-II solution by combining both approaches. This case study, even not very often mentioned in the paper, was a rich and powerful source for understanding, comparing and analysing both approaches. In addition, it can be read as a very detailed and broad illustration of our more generic conclusions in this paper.

Therefore, we did not try to get a representative classification by regarding several approaches and aggregating them but we try to take a close view on two different paradigm by choosing two prototypical instances as input for a detailed case study.[1] Our study continues [GaP92] who implicitly compared KADS and their role-limiting method approach by describing how the later can be described using the further one. This is the one of the few approaches which investigates how role-limiting methods can be re-expressed in terms of the general methodological framework of KADS and therefore bridging the dichotomy between methodological and tool-oriented approaches. Compared to our study [GaP92] is much more focused as it asks how to represent a role-limiting method in terms of the KADS model of expertise.

1 How to Bridge the Representation Gap

The development of an (software) artifact contains two main activities. First, the problem to be solved by the artifact must be *analysed and specified*.[2] Second, the artifact

1. In social science, this distinction corresponds to the distinction of normative and interpretative oriented techniques and methods (see [Fen92]).

which is to solve the given problem must be *designed and implemented*. Therefore most, if not all, process models in software and knowledge engineering distinguish these two activities in a project even though, in detail, their distinctions are treated in a different way.[3]

MIKE clearly separates the analysis and the design/implementation of an expert system during the project. The outcome of the analysis phase is a (formal) specification of the task which is to be solved by the system and of the knowledge which is required to solve the task effectively and efficiently (without symbol-level control, cf. [Sch92]). The outcome of the implementation phase is a computational agent that solves the problem. This is achieved by a kind of refinement step. The declarative description of a model of expertise in KARL is refined by additional data structures and efficient algorithms which compute the semantics of the model more efficiently [LaS94].

In CRLM, the analysis phase consists of two steps. First, the appropriate problem-solving method and its knowledge representation formalism is selected. Second, the predefined knowledge representation formalism of this problem-solving method guides the acquisition of domain knowledge. Ideally, there is no further design/implementation step in the project because the previous implementation of the problem-solving method (which has been carried out independently of the current project) is reused. The result of the analysis phase is a running system. Yet, it may be necessary to reconfigure a selected method to a specific problem or to implement a new sub-method for a task only partially covered by the selected method combination. While new sub-methods must be explicitly coded, the corresponding knowledge acquisition components can be automatically generated from a declarative description of the knowledge representations and editors. This knowledgeacquisition tools allow domain experts after a training phase to develop the knowledge base by themselves without further guidance by a knowledge engineer [GPS93]. The experts may directly evaluate the knowledge base with test cases without the need for a further precompilation step. While this model of direct knowledge acquisition by domain experts is seen by some as the most important advantage of CRLM, it is for others the cause of important drawbacks. On one hand:

- The quality of the knowledge base is greatly improved since no translation errors occur between the knowledge engineer and the domain experts.
- The maintenance of the knowledge base can be done by the domain experts.

On the other hand:

- The expert has to enter the knowledge without further guidance by intermediate models. He has to analyse and represent his knowledge at the same time.
- Due to the lack of intermediate models the executable knowledge base is the only documentation of the expertise.
- The view of knowledge is obviously determined by the knowledge representation formalism of the chosen shell although the expert only has to deal with the corresponding graphical representations.

Whether the advantages of this process model outweigh the disadvantages cannot be answered in general but depends on the circumstances of the actual project. But we found that least for classification the experts are comfortable with it and develop

---

2. In the case of a knowledge-based system an integrated part of the analysis phase is the modelling of the knowledge which is required to solve the task.
3. A survey of discussion in software engineering can be found in [ThD90].

knowledge bases with high competence, c.f. [ScS93].

The process model of MIKE is influenced by work in conventional software engineering. In fact it is based on the *spiral model* of [Boe88] and different approaches to *prototyping* of [Flo84]. Its main principles are:

- The entire development process is subdivided into several different phases. Each phase is concerned with a specific aspect of the development process. This defines a clear focus of interest for every activity and reduces its complexity. The four main phases of the process model of MIKE are analysis, design, implementation, and evaluation. Each phase is again split into several subphases.
- As the original life cycle-oriented process models in software engineering rely on unrealistic assumptions (e.g., the waterfall model), the process model of MIKE regards these phases as incremental and cyclic. [Boe88] views this process as a spiral where the entire functionality of the system is achieved by several iterations of these different phases.
- An important aspect of each phase is the evaluation of the achieved results. The main means of evaluation is *prototyping*. The outcome of every phase is an operational description which can be evaluated by prototyping.

The process model of MIKE tries to integrate the advantages of well-structured process models into incremental system development and prototyping. Looking in greater detail at its process model, the large *modelling gap* between informal descriptions of the expertise that are gained from the expert by using knowledge-acquisition methods, and the final realization of the expert system *is bridged* by several intermediate models. Decomposing this gap into smaller ones reduces the complexity of the whole modelling process since in every step particular aspects may be considered independently of other aspects. Five different descriptions of a task and the required knowledge exist in MIKE (see figure 1).

First, knowledge and task are described in *natural-language documents*. These documents may result from interviews or observations, or can already exist as manuals or books, etc. These documents can be structured and represented in a protocol model which uses a hyper media representation. Second, these informal descriptions are transformed into a *semiformal representation* called structure model. For building the protocol model as well as the structure model the hypermedia tool MeMo-Kit (Mediating Model Construction Kit) can be used [Neu93]. As a result, the knowledge and the task are described along the lines of a model of expertise as defined in KADS [SWB93]. The description of knowledge is structured in different layers by using appropriate primitives, which are also associated with a suitable graphical representation. The semantics of elementary knowledge pieces is still defined in natural-language. Such a mediating representation has the following advantages: The structuring process creating the mediating representation provides early feedback for the knowledge engineer and the expert; the semiformal representation of the expertise provides a useful basis for communicating with the expert; the contents of the model may be exploited for the explanation facility of the final system; and the model documents modelling decisions and thus may be used for the maintenance of the final system.

The third type of description is accomplished by using KARL. Knowledge which is represented informally or semiformally is formalized during the *knowledge-formalization step*. The main benefits of formal descriptions of expertise, compared to informal or semiformal representations, are the following: The vagueness and ambiguity

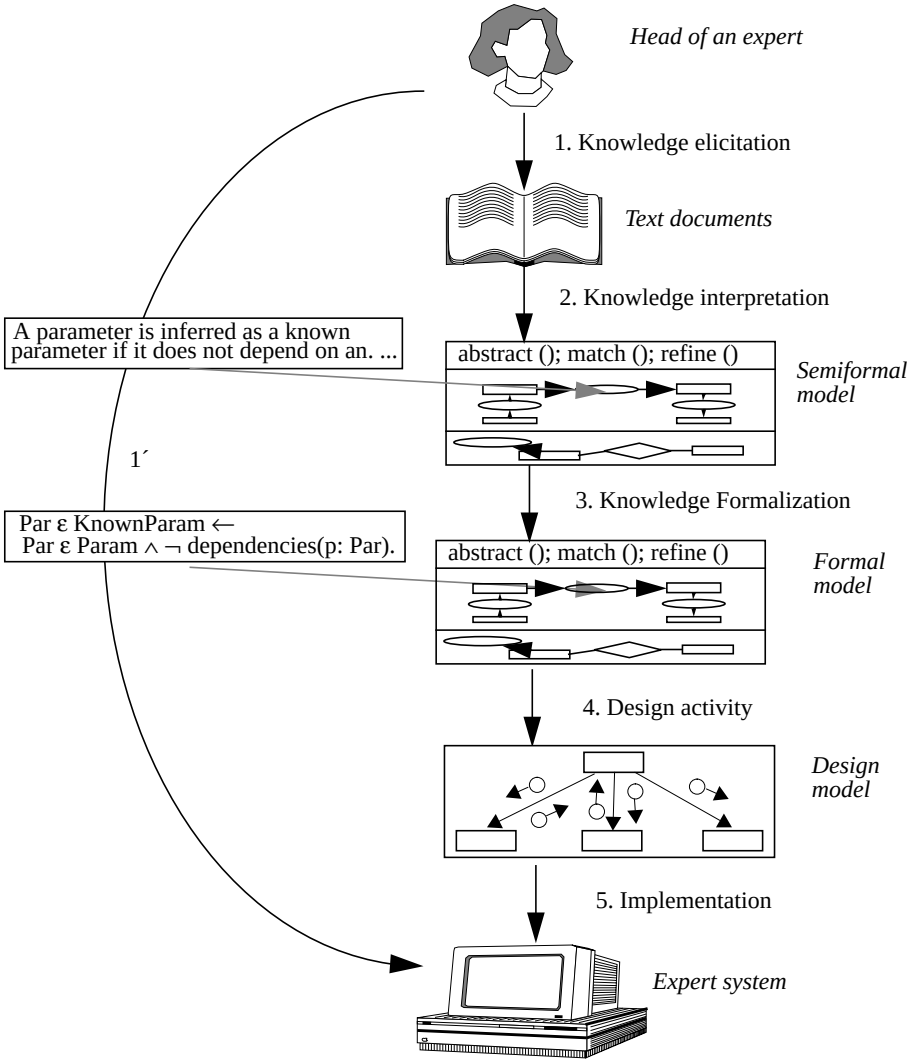

Fig. 1. The five steps of MIKE (1-5) and the one-step transition (1´) of CRLM.

of natural-language descriptions can be avoided; the formalized problem-solving method can be used to guide the collection of domain knowledge; the formal description can help to get a clearer understanding of single problem-solving steps as well as of complete problem-solving methods, it thus supports their reuse; and a formalized specification can be mapped to an operational one. This allows prototyping or a symbolic execution in order to evaluate the knowledge, thus supporting incremental modelling.

Formalization results in a formal and operational description of the model of expertise. Since a KARL specification is based on the structure of the KADS model of expertise there is a smooth transition from an semiformal to a formal description. The KARL

Fig. 2. The process model in CRLM

model is constructed by refining the semiformal model of expertise, e.g., by augmenting an informal description of an elementary inference step in the semiformal model by a formal description. Formal descriptions should not replace informal ones but rather define their meaning precisely and uniquely. Natural language is very useful to outline the general idea of an inference since in a formal description one often cannot see the wood for trees. One the other hand, it is very difficult if not impossible to define the exact meaning of an inference in a precise and unique manner by natural language only. KARL is a customization of first-order logic consisting of the two sublanguages L-KARL that represents static knowledge and P-KARL that represents dynamic knowledge. It has a declarative semantics [Fen93a] as well as an operational semantics [Ang93]. L-KARL is based on Frame-logic [KLW93], which integrates object-orientation into a declarative framework. P-KARL is based on dynamic logic [Koz90], which integrates the representation of procedural knowledge into a declarative framework. The modelling primitives of KARL reflect the structure of the model of expertise of KADS (i.e., the separation of different layers and different modelling primitives at each layer, e.g. inference actions and roles at the inference layer). As a specification in KARL is based on the structure of the model of expertise, there is a smooth transition from an informal to a formal description. Natural-language definitions of the meaning of graphically specified elements of a model are supplemented by formal definitions using KARL.

The fourth description level is defined by the *design model*. The model of expertise finally includes all functional requirements posed on the desired system. For the realization of the final system additional requirements have to be considered which are still independent of the final implementation of the system. They are non-functional requirements such as efficiency of the realization of the problem-solving method, maintainability of the system, persistency of data etc. The design model enriches and refines the model of expertise by taking these issues into account, e.g., by introducing appropriate algorithms and data structures, it takes care for a suitable modularization of the system, etc. Capturing such design decisions in the design model narrows the gap between the model of expertise and the implementation of the final system. For instance,

the informal and formal but declarative description of an inference action is supplemented by appropriate data structures and algorithms which support an efficient computation (cf. [LaS94] for more details). The final description is achieved by *implementing the system* in the given hardware and software environment.

When we compare these transitions with respect to the way in CRLM it becomes clear that the expert has to do the first three transitions in one go by directly specifying the domain knowledge in the graphical representation of the language of the chosen shell. He has not to deal with design decisions and the details the language is implemented in, since this is viewed as an fixed and given entity for him as described in the next section. Given feedback by cases from end uses he may have to maintain the knowledge base either by changing existing knowledge pieces or adding new ones. This process becomes more complicated in the case that an appropriate shell is not available. Then the knowledge engineer and the expert have to customize the best fitting shell for the specific needs of the expert. It may even happen during knowledge maintenance that the shell must be adapted to allow the formulation of other knowledge types. This double spiral is shown in the figure 2.

2 Relationship of Domain Knowledge and Problem-Solving Methods

An important common feature of both approaches is *the separation of domain knowledge from generic (i.e., domain-independent) problem-solving knowledge.* The problem-solving knowledge controls the use of the domain knowledge for problem-solving. Yet, the two approaches differ significantly in their way of separating and combining both parts of an expert system.

CRLM clearly separates the problem-solving method from domain knowledge. The problem-solving method, its terminology (i.e., the object and relation types it uses) and the knowledge acquisition environment are an implemented and *fixed* building block which is called an *expert-system shell*. The domain knowledge, which consists of instantiations of the generic object and relation types of the selected problem-solving method, is acquired during the knowledge-acquisition step. The domain knowledge is called the *knowledge base* of the expert system. *The terminology is problem-solving-method-specific, but not domain-specific.* The predefined terminology allows the development of reusable graphical knowledge-acquisition tools for the domain knowledge ([GPS93], [PoG93]). This enables domain experts to enter their knowledge without assistance, that is, without a translation step by a knowledge engineer, and to maintain the knowledge base. The reuse of domain knowledge for different problem-solving methods requires additional effort. Even a method specific knowledge base consists of problem specification and problem-solving knowledge. The problem specification can be reused for other methods, for example in classification the same set of observables and diagnoses may be used for heuristic, case-based and set-covering classification, but the terminology must eventually be mapped. The problem-solving knowledge, e.g. set-covering relations, can only be reused when explicit knowledge transformation procedures are applied as for example in [BGG+93]. These mappings or transformations are significantly eased by the explicit knowledge representation of the role limiting methods, where each knowledge entry is used in exactly in way that is at least informally specified. MYCIN rules in contrast are nearly impossible to reuse since they contain both diagnosis rating and dialogue guidance knowledge in the same place.

The model of expertise as the result of the analysis phase in MIKE clearly separates

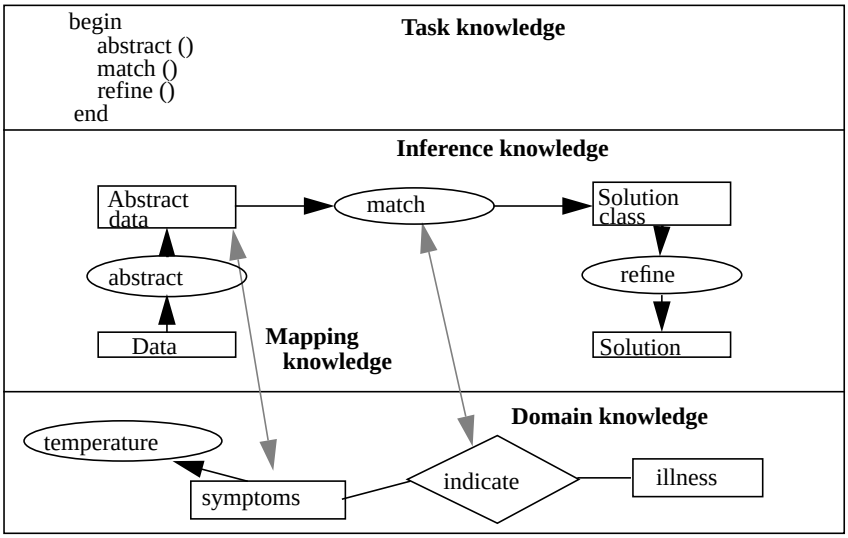

Fig. 3. The model of expertise of heuristic classification in KARL.

three types of knowledge and represents them at three different layers (see figure 2). At the *domain layer*, the domain knowledge is represented independently of its application for the problem-solving process. It consists of the terminology of a domain, a set of rules and constraints which model regularities in this domain, and a set of facts which represent factual knowledge. The *inference layer* is used to represent the inference steps of a problem-solving method and their data dependencies, but also defines a problem-solving-method-specific or task-specific terminology. Domain knowledge and domain terminology are mapped onto the inference layer and its task-specific terminology via view definitions. The *task layer* represents the control flow between the inference steps, that is, it can be used to define sequences, choices, and iterations of inferences. *None of these layers is fixed.* Due to the flexible mapping, a domain layer can in principle be reused for different problem-solving methods [PiS94].[4]

Comparing both approaches with respect to the relationship of domain knowledge and problem-solving method, three main distinctions can be identified (see figure 3):

- CRLM views the problem-solving method as a fixed entity which is not a part of the knowledge base. MIKE views domain knowledge and the generic control knowledge of the inference layer and task layer as part of the knowledge base.
- CRLM only offers a terminology that is specific to a problem-solving method. A model of expertise in MIKE contains two terminologies, that is a domain-specific terminology at the domain layer and a problem-solving-method-specific terminology at the inference layer.
- The relationship of the domain knowledge and the problem-solving method is expressed by the instantiation of a generic terminology in CRLM and by defining

4. The different knowledge types can be used to refine the process model of MIKE as their elicitation and modelling define different activities of the development process. The inference and task knowledge can be used as a guidance for modelling domain knowledge, for example (see for more details [NeS92]).

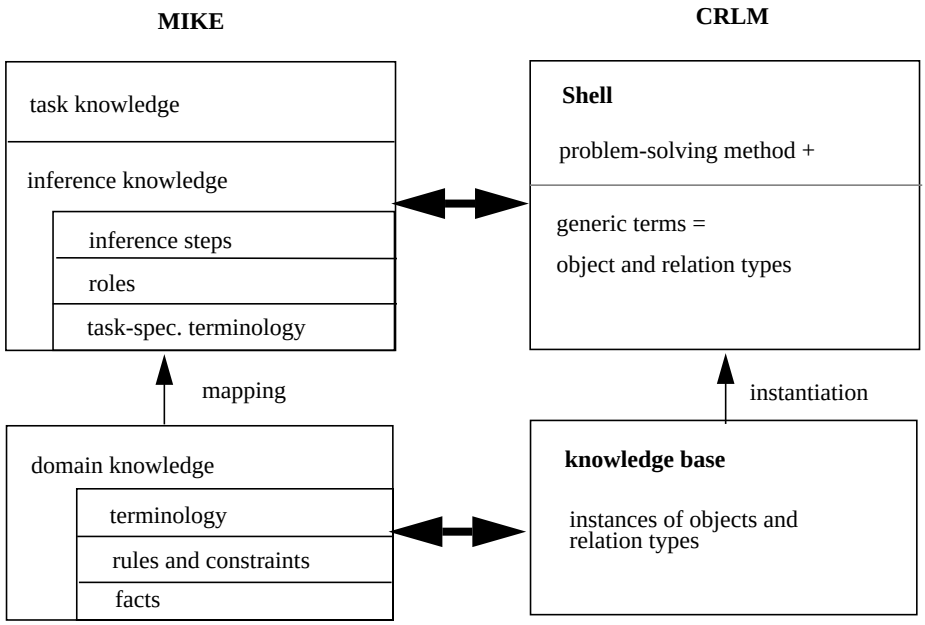

Fig. 4. Relationship of domain knowledge and problem-solving methods in MIKE and CRLM.

a mapping via Horn clauses extended by stratified negation (cf. [Llo87], [Prz88]) in MIKE.

CRLM and MIKE are both methods of developing *expert systems*.[5] Therefore, it is not without interest to ask how both methods view the difference between expert systems and ordinary software programs. The question of how an approach characterizes an expert system is not only of philosophical interest but it also brings up the problem of the scope of its applicability. Both methods aim at supporting the development process of *expert* systems, and not of *arbitrary* systems. Therefore, if one wants to apply one of these methods it must be made sure that one wants to build an expert system in the sense of the selected method.

From the CRLM point of view, the following features are essential for an expert system (despite the fact that the performed task is meant for an expert): transparency, flexibility, user friendliness, and competence (cf. [Pup93]). To that end, the separation of an inference component and domain knowledge is inevitable. The inference component is not viewed as a part of the knowledge base. Domain knowledge (i.e., the knowledge base) can be changed without affecting the inference component. The inference component, that is, the dynamic or control knowledge, is viewed as fixed.

In MIKE, the problem-solving method is regarded as a part of the knowledge which consists of the three parts already mentioned: domain knowledge, inference knowledge,

5. To some extent, this is not true for MIKE. In [FAL+93] it is shown how KARL can be used to formalize and operationalize techniques of structured analysis, which is a commonly used approach in software engineering. Therefore, the specification of knowledge-based components can naturally be embedded into the specification of an entire system containing also conventional parts.

and task knowledge. None of these parts is regarded as fixed, and the dynamic or control knowledge which is represented at the inference layer and task layer is part of the knowledge base. Yet, every knowledge type is clearly separated and can be manipulated independently of the others. An expert system is mainly characterized by the kind of task it solves. An expert system is a computer program which solves a task requiring a high amount of knowledge and intellectual capability when a human solves the task, *and* this knowledge is necessary for solving the task, that is, the problem cannot be efficiently solved by a simple complete search through the variety of possible solutions.

While in CRLM an expert system is mainly characterized by the features of the system and its realization, in MIKE it is mainly characterized by the features of the task it solves. This is also reflected in the different degrees of emphasis on the individual life-cycle phases (specification versus design/implementation) in the different approaches.

3 Degree of Mechanization

The view of the *knowledge engineer´s* the probably defines one of the most significant differences between both approaches.

In MIKE, the knowledge engineer is the essential medium whose task is to bridge the assumed deep and wide gap between the human expertise and the expert system. The knowledge engineer has to create the model of expertise in cooperation with the expert. The expertise which exists as skills or hidden and implicit knowledge must be transformed into an explicit and formal model. As neither this model exists before the knowledge-acquisition process, nor are the experts very apt at retrospectively developing a model of their own expertise, the development of the model of expertise is mainly the task of the knowledge engineer.

Role-limiting methods provide shells which should enable experts to write expert systems on their own. Therefore, it is assumed that a knowledge engineer is only required during the initial problem analysis when he or she must decide whether there is an appropriate shell for the given task and, if so, which one should be chosen. In CRLM, the role of the knowledge engineer is enlarged to some extent. As an application may not perfectly fit a given shell, or a complex task may require the combination of several shells, the knowledge engineer must configure the appropriate shell. Therefore, he or she must elicit knowledge about the task and the appropriate problem-solving method. The appropriate method is therefore no longer regarded as a fixed precondition for the knowledge acquisition process, but, to some extent it also constitutes a part of its result.

A benefit of the CRLM approach is the *tool support* it offers for knowledge acquisition and knowledge evaluation. As the problem-solving method and its terminology are fixed, the expert only has to instantiate given generic structures. Tools with graphical interfaces can be defined and implemented in order to support this powerful means of knowledge acquisition. Evaluation is supported by the running prototype of the system and by additional tools that use the fixed structure and semantics of the knowledge base imposed by the problem-solving method.

The main means of MIKE are the hypermedia tool MeMo-Kit [Neu93] and the formal and operational language KARL. MeMo-Kit supports the process of building a semiformal model by different customized editors and a library of informally described reusable problem-solving methods. KARL allows a precise and unique description of the knowledge and its evaluation by prototyping. For this purpose, an interpreter and

debugger for KARL was developed which is integrated into the MeMo-Kit environment.

4 Reuse By Use of Specification Languages

Both approaches aim at the *reuse* of knowledge and software. [Kru92] compares eight different types of software reuse. One of them is based on *very-high level languages (VHLL)* like KARL which is used by MIKE. VHLL define a higher conceptual level than high-level programming languages like C or Modula by abstracting from efficiency and other implementation details, while having the same or even more expressive power. In comparison to assembler even in a high-level language a considerable reuse rate can be achieved by grouping a set of low-level statements into a single high-level statement. This kind of reuse is extended by VHLLs where it is possible to quickly write powerful (though inefficient) programs. Due to their generality, they can be applied to a broad range of tasks. KARL is neither task-specific nor domain-specific, that is, in principle it should be possible to specify arbitrary tasks and domains in KARL. A significant restriction is the fact that KARL cannot be used to specify real-time problems.

MIKE and CRLM both use *specification languages* which abstract from the implementation, but differ significantly in the level of abstraction of these languages and by the way their semantics is defined. CRLM uses *problem-solving-method-specific representation languages*. Every problem-solving method is combined with a specific representation language that allows the expert to specify knowledge in terms of his or her task without referring to a general-purpose representation formalism. As already mentioned, given a complete graphical interface, this allows non-programmers to program (i.e., to enable them to develop and maintain the knowledge base without assistance). In contrast to KARL, these languages do not have a declarative semantics. Their semantics is only defined by their interpreter, that is, by the shell using the language. An implementation of a problem-solving method together with its specific representation formalism and knowledge acquisition tools can be applied for problems in different domains. [Kru92] classifies this approach as *application generators*, where an application-specific (task-specific or better problem-solving-method-specific in our terms) languages is provided that allow users to think in terms of their application (i.e., task). Comparing both concepts of software reuse, two comments can be made:

- On the one hand, task-specific languages are easier to learn because of their limited range of applicability and there higher conceptual level than that of general-purpose languages. Users can think in terms of their task instead of learning a very general language.
- On the other hand, the limited range of task-specific languages restricts the expressiveness of the language constructs. Different tasks or the combination of several problem solvers require the effort of developing and learning different languages.

Conclusion: Contradiction or Complement?

Finally, we discuss some experiences concerning the combination of MIKE with CRLM. To some extent, both approaches are complementary and supplement each other very well. [FEM+93] already report on a fruitful combination of a role-limiting method approach and MIKE. In its current stage, MIKE offers significant tool support for the early phases in knowledge engineering. The hyper model can be used to semiformally

describe a model of expertise and this description can be further refined and operationalized by the specification language KARL. Language and tool support for the design phase is under way. Currently, there is no support for the implementation of a final system. The inverse holds for the CRLM approach which only supplies powerful shells which eliminate or drastically reduce the implementation effort, but provides less support for the early knowledge acquisition phases. By combining both approaches, a description of a system and of the used knowledge at different complementary levels can be achieved: First, the knowledge is described at the *conceptual level* in a semiformal manner by the different layers and primitives of a model of expertise. Second, the knowledge is described at the *formal level* to define a precise and unique meaning. This formal description makes it possible to exactly define the knowledge-based systems without referring to implementational aspects. Third, the knowledge is described at the *implementational level* by a running system. The domain knowledge can comfortably be acquired and efficiently be executed by the shell. Since the implemented problem solver operates directly on a representation corresponding to the problem solving method explanation and maintainability capabilities are significantly improved.

Speaking as a cynic, both approaches fit together and supplement each other because of the incompleteness of their current development state. On the other hand, both approaches seem to converge regarding the kind of *reuse* they aim to support in the near future. The original RLM reused monolithic task-specific shells. An implementation of a problem-solving method together with its task specific representation formalism and knowledge acquisition tools could be applied for problems in different domains. CRLM extend this as also smaller parts of problem solving methods (mechanisms) can be reused, and new methods can be configured from the mechanisms. The knowledge acquisition tools can be generated from a declarative description of the knowledge representation and the views acquired in the knowledge editors. The library of CRLM currently contain mechanisms for heuristic ([PuG92], [GPS93]) case-based [PuG91] and set-covering classification [Pup93], methods for assignment (propose-and-exchange [PoP92]) and for simple configuration (propose-and-revise [PFL+94]).

In MIKE, reuse is enabled by the very-high level language KARL which makes it possible to quickly write powerful (though inefficient) specifications of problem-solving methods and domain knowledge. A further kind of reuse in MIKE is subject of ongoing work. Different problem-solving methods like the *board-game method* [FEM+93], *cover-and-differentiate* [Ang93], *propose-and-exchange* [LFA93] together with their generic terminology have been specified in KARL. These collection of problem-solving methods will be continually extended, and the formal semantics of these models will be investigated in order to support the selection, modification, and combination of these methods based on formal goal descriptions of a given task (cf. [Fen93b]). A library of semiformally specified problem-solving methods which supplements these formal descriptions is described in [Neu94].

The library of semiformal, formal and operational problem-solving methods indicates that MIKE and CRLM are convergent. The main difference concerns the representation of the reusable blocks, e.g. the problem-solving methods, and their descriptions. CRLM provides an informal textual description and an efficient implementation of such a block in a programming language. MIKE provides a semiformal description and a formal (and operational) description, but no implemented building blocks. The formal semantics of it can be used to derive properties of it (e.g., pre and post conditions which can be used to guide the selection process). There is a clear need in the CRLM approach to use some kind of precise and unique description formalisms for its implemented mechanisms in

order to support their selection and combination.

A clear distinction between the both approaches arises by the different ways to view domain knowledge: KARL (MIKE) uses a domain-specific and a problem-solving-method-specific terminology. The user can work in terms of his domain or of the applied problem-solving method. In CRLM, only a problem-solving-method-specific terminology is provided and the implemented system does not use the domain-specific terminology. Domain knowledge is assumed to be only factual knowledge. This restriction is necessary to define powerful generic knowledge acquisition tools. A first step to enrich the role-limiting method approach was done by *PROTÉGÉ* [Mus89] which allows the derivation of domain-specific acquisition tools, but requires structural equivalence of domain and task knowledge. Work on the *PROTÉGÉ-II framework* [PET+92] tries to define more flexible mappings between problem-solving methods and domain ontologies which again narrows the gap between both approaches.

Acknowledgement

We thank Jürgen Angele and Dieter Landes who participated on our Sisyphus-II case study, Gabi Rudnick for partial correction of our manuscript, and two anonymous referees for helpful comments.

References

[AFL+93] J. Angele, D. Fensel, D. Landes, S. Neubert, and R. Studer: Model-Based and Incremental Knowledge Engineering: The MIKE Approach. In J. Cuena (ed.), *Knowledge Oriented Software Design*, IFIP Transactions A-27, Elsevier, Amsterdam, 1993.

[AFS90] J. Angele, D. Fensel, and R. Studer: Applying Software Engineering Methods and Techniques to Knowledge Engineering. In D. Ehrenberg et al. (eds.), *Wissensbasierte Systeme in der Betriebswirtschaft, Reihe betriebliche Informations- und Kommunikationssysteme, no 15*, Erich Schmidt Verlag, Berlin, 1990.

[AFS94] J. Angele, D. Fensel, and R. Studer: The Model of Expertise in KARL. In *Proceedings of the 2nd World Congress on Expert Systems*, Lisbon/Estoril, Portugal, January 10-14, 1994.

[Ang93] J. Angele: *Operationalisierung des Modells der Expertise mit KARL (Operationalization of a Model of Expertise with KARL)*, Ph. D. thesis, University of Karlsruhe, 1993 (in German).

[BGG+93] S. Bamberger, U. Gappa, K. Goos, and K. Poeck: Teilautomatische Wissenstransformation zur Unterstützung der Wissensakquisition (Semiautomatic Knowledge Transformation Supporting Knowledge Acquisition). In *Proceedings of the 2. Deutsche Tagung Expertensysteme (XPS-93)*, Hamburg, February 17-19, 1993 (in German).

[Boe88] B.W. Boehm: A Spiral Model of Software Development and Enhancement, *IEEE Computer*, May 1988.

[Boo88] J. H. Boose: A Research Framework For Knowledge Acquisition Techniques and Tools. In *Proceedings of the 2nd European Knowledge Acquisition for Knowledge-Based Systems Workshop (EKAW-88)*, St. Augustin/Bonn, 1988.

[FAL91] D. Fensel, J. Angele, and D. Landes: KARL: A Knowledge Acquisition and Representation Language. In *Proceedings of the 11th International Conference on Expert Systems and their Applications, vol.1, General Conference Tools, Techniques and Methods*, 27-31 May, Avignon, 1991.

[FAL+93] D. Fensel, J. Angele, D. Landes, and R. Studer: Giving Structured Analysis Techniques a Formal and Operational Semantics with KARL. In *Proceedings of Requirements Engineering ´93 - Prototyping -*, Bonn, April 25 - 27, 1993, H. Züllighoven (ed.), Teubner Verlag, Stuttgart, 1993.

[FeH94] D. Fensel and F. van Harmelen: A Comparison of Languages which Operationalize and Formalize KADS Models of Expertise. In *The Knowledge Engineering Review*, vol 9, no 2, June 1994.

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

[Fen92] D. Fensel: Knowledge Acquisition and the Interpretative Paradigm. In F. Schmalhofer et al. (eds.), *Contemporary Knowledge Egineering and Cognition, First Joint Workshop*, Kaiserslautern, Germany, February 21-22, 1991, Lecture Notes in Artificial Intelligence, no 622, Springer-Verlag, Berlin, Juli 1992.

[Fen93a] D. Fensel: *The Knowledge Acquisition and Representation Language KARL*, Ph D. thesis, University of Karlsruhe, 1993.

[Fen93b] D. Fensel: Reuse of Problem-Solving methods in Knowledge Engineering. In *Proceedings of the 6th Annual Workshop on Software Reuse (WISR´6)*, Owego, New York, November 1-4, 1993.

[FGS93] D. Fensel, U. Gappa, and S. Schewe: Applying a Machine Learning Algorithm in a Knowledge Acquisition Scenario. In *Proceedings of the IJCAI´93 Workshop Knowledge Acquisition and Machine Learning*, Chambery, France, August 28th - September 3rd, 1993.

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

[GaP92] U. Gappa and K. Poeck: Common Ground and Differences of the KADS and Strong-Problem-Solving-Shell Approach. In *Proceedings of the 6th European Knowledge Acquisition for Knowledge-Based Systems Workshop (EKAW-92)*, May 18-22, Heidelberg/Kaiserslautern, 1992, Lecture Notes in Artificial Intelligence, no 599, Springer-Verlag, Berlin, 1992.

[GaP94] U. Gappa and K. Poeck: An Architecture for Reusing Role-limiting Mechanisms and Knowledge Acquisition Modules, submitted, 1994.

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

[KLV90] W. Karbach, M. Linster, and A. Voss: Models, Methods, Roles, and Tasks: Many Labels—One Idea. In *Knowledge Acquisition*, vol 2, no 4, 1990.

[KLW93] M. Kifer, G. Lausen, and J. Wu: Logical Foundations of Object-Oriented and Frame-Based Languages. In Technical Report 93/06, Department of Computer Science, SUNY at Stony Brook, NY, April 1993. To appear in *Journal of ACM*.

[Koz90] D. Kozen: Logics of Programs. In J. v. Leeuwen (ed.), *Handbook of Theoretical Computer Science*, Elsevier, Amsterdam, 1990.

[Kru92] C. W. Krueger: Software Reuse, *ACM Computing Surveys*, vol 24, no 2, June 1992.

[LaS94] D. Landes and R. Studer: The Design Process in MIKE. In *Proceedings of the 8th Banff Knowledge Acquisition for Knowledge-Based System Workshop (KAW´94)*, Banff, Canada, Januar 30th - February 4th, 1994.

[LFA93] D. Landes, D. Fensel, and J. Angele: Formalizing and Operationalizing a Design Task with KARL. In J. Treur and T. Wetter (eds.), *Formal Specification of Complex Reasoning Systems*, Ellis Horwood, Chichester, 1993.

[Lin93] M. Linster: A review of Sisyphis 91 & 92: Models of Problem-Solving Knowledge. In *Proceedings of the 7th European Knowledge Acquisition Workshop (EKAW´93)*, Toulouse, France, September 6-10, Lecture Notes in Artificial Intelligenc, no 723, Springer-Verlag, Berlin, 1993.

[Llo87] J.W. Lloyd: *Foundations of Logic Programming, 2nd Editon*, Springer-Verlag, Berlin, 1987.

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

[McD88] J. McDermott: Premilary Steps Towards a Taxonomy of Problem Solving Methods.

In [Mar88].

[Mus89] M. A. Musen: *Automated Generation of Model-Based Knowledge-Acquisition Tools*, Morgan Kaufmann Publisher, San Mateo, CA, 1989.

[NeS92] S. Neubert and R. Studer: The KEEP Model. In *Proceedings of the 6th European Knowledge Acquisition for Knowledge-Based Systems Workshop (EKAW-92)*, May 18-22, Heidelberg/Kaiserslautern, 1992, T. Wetter et al. (eds.), *Current Developments in Knowledge Acquisition*, Lecture Notes in Artificial Intelligence, no 599, Springer-Verlag, Berlin, 1992, pp. 230-249.

[Neu93] S. Neubert: Model Construction in MIKE (Model-Based and Incremental Knowledge Engineering). In *Knowledge Acquisition for Knowledge Based Systems, Proceedings of the 7th European Knowledge Acquisition Workshop (EKAW´93)*, Toulouse, France, September 6-10, Lecture Notes in Artificial Intelligenc, no 723, Springer-Verlag, Berlin, 1993.

[Neu94] S, Neubert: Modellkonstruktion in MIKE - Methoden und Werkzeuge (Model Construction in MIKE - Methods and Tools), Ph.D thesis, University of Karlsruhe, 1994 (in German).

[NPB+91] H. S. Nwana, R. C. Paton, T. J. M. Bench-Capon, and M. J. R. Shave: Facilitating the Development of Knowledge Based Systems. In *AI Communication (AICOM)*, vol 4, no 2/3, 1991.

[Nwa93] H. S. Nwana: Using KADS and Generic Tasks to Model a Timetabling Problem. In *Complement Proceedings of the 7th European Knowledge Acquisition Workshop (EKAW´93)*, Toulouse, France, September 6-10, 1993.

[PFL+94] Karsten Poeck, Dieter Fensel, Dieter Landes, and Jürgen Angele: Combining KARL and Configurable Role Limiting Methods for Configuring Elevator Systems. In *Proceedings of the 8th Banff Knowledge Acquisition for Knowledge-Based System Workshop (KAW´94)*, vol 3, Banff, Canada, Januar 30th - February 4th, 1994.

[PET+92] A. R. Puerta, J. W. Egar, S. W. Tu, and M. A. Musen: A Multiple-Method Knowledge-Acquisition Shell For The Automatic Generation Of Knowledge-Acquisition Tools, *Knowledge Acquisition*, vol 4, no 2, 1992.

[PiS94] T. Pirlein and R. Studer: An Environment for Reusing Ontologies within a Knowledge Engineering Approach. In N. Guarino et al. (eds.), *Formal Ontology in Conceptual Analysis and Knowledge Representation*, Kluwer, Boston, to appear 1994.

[PoG93] K. Poeck and U. Gappa: Making Role-limiting Shells More Flexible. In *Knowledge Acquisition for Knowledge Based Systems, Proceedings of the 7th European Knowledge Acquisition Workshop (EKAW´93)*, Toulouse, France, September 6-10, Lecture Notes in Artificial Intelligence, Springer, 1993.

[PoP92] K. Poeck and F. Puppe: COKE: Efficient solving of complex assignment problems with the propose-and-exchange method. In *Proceedings of the 5th International Conference on Tools with Artificial Intelligence*, Arlington, Virginia, USA, November 10-13, 1992.

[Prz88] T. C. Przymusinski: On the Declarative Semantics of Deductive Databases and Logic Programs. In J. Minker (ed.), *Foundations of Deductive Databases and Logic Programming*, Morgan Kaufmann Publisher, Los Altos, CA, 1988.

[PuG91] F. Puppe and K. Goos: Improving Case-based Classification with Expert Knowledge. In *Proceedings of the 15th German Workshop on Artificial Intelligence (GWAI-91)*, Bonn, September 16-20, 1991.

[PuG92] F. Puppe and U. Gappa: Towards Knowledge Acquisition by Experts. In *Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, Proceedings of the 5th International Conference IEA/AIE-92*, Paderborn, June 9-12, 1992.

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

[Sch92] G. Schreiber: *Pragmatics of the Knowledge Level*, Ph D. Thesis, University of Amsterdam, 1992.

[ScS93] S. Schewe and M. A. Schreiber: Stepwise development of a clinical expert system

in rheumatology, In *The Clinical Investigator*, 1993.

[Ste93] L. Steels: The Componential Framework and its Role in Reusability. In J.-M. David et als (eds.), *Second generation expert systems*, Springer, Berlin, 1993

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

[ThD90] R. H. Thayer and M. Dorfman (eds.): *System and Software Requierements Engineering*, IEEE Computer Society Press, Los Alamitos, California, 1990.