In Proceedings of the 2nd World Congress on Expert Systems, Lisbon/Estoril, Portugal, January 10-14, 1994.

The Model of Expertise in KARL

J. Angele, D. Fensel, and R. Studer
Institut für Angewandte Informatik und Formale Beschreibungsverfahren
University of Karlsruhe, Englerstr. 11, 76128 Karlsruhe, Germany
e-mail: {angele | fensel | studer}@aifb.uni-karlsruhe.de

Abstract. In the paper we present the modelling primitives of KARL. KARL is a formal and operational language to represent models of expertise of knowledge-based systems. KARL allows to describe the model of expertise on a high level of abstraction. The formal semantics of KARL provides means to describe the expertise unambiguously and the operational semantics of KARL allows to directly execute models of expertise in order to provide feedback for the expert, the knowledge engineer and the user of the final knowledge-based system. KARL is based on object oriented logic and dynamic logic.

Keywords: Knowledge Acquisition, Model of Expertise, Knowledge Representation, Software Engineering, Prototyping

1 Introduction

During the last years there has been a paradigm shift in the development process of expert systems. The building process of a knowledge-based system is no longer seen as a transfer process which assumes that the knowledge in the heads of the experts is structured in a similar way as in the system. Instead this process is now seen as a modelling activity (cf. [FBA93]):

- Only part of the knowledge of an expert is conscious to the expert and may be articulated by the expert directly. A large part of his knowledge and especially that knowledge which allows him to solve those specific problems for which he is an expert is not conscious to him. It is the goal of the development process to create an explicit model of the knowledge (model of expertise) which is necessary to solve a given task.
- Such a model of expertise abstracts from all details which are related to design and implementation of the final system. This separation is based on Newells (cf. [New82]) distinction of symbol and knowledge level and corresponds to a similar distinction in conventional software engineering. During the analysis phase a model of the system functionality is built which abstracts from all issues that are concerned with design and implementation aspects.

The KADS four layer model of expertise (cf. [WSB92]) defines a general framework which can be used to express such a model. It defines four different types of knowledge and several modelling primitives for every knowledge type allowing to describe the modelled expertise in an informal or semi-formal way. Though these descriptions provide a good basis for the communication between the expert and the knowledge engineer they also bring about several disadvantages:

- Informal representations include a lot of vagueness and ambiguities which allow to interpret these descriptions in many different ways.
- An informal description which does not have a clear formal semantics cannot be analyzed by techniques such as symbolic evaluation, consistency checking, or redundancy checking.
- A description which does not have an operational semantics cannot be executed in order to provide feedback to the expert or the knowledge engineer by a prototype.

One of the key ideas in software engineering in order to overcome the shortcomings of informal or semi-formal descriptions is the development of formal and operational specification languages. These languages allow a formal and executable description of the system functionality without considering implementation details which are investigated during later steps of the software development process.

With the “Knowledge Acquisition and Representation Language (KARL)”[1] we subsequently introduce a language which helps to overcome the shortcomings of such informal descriptions of a model of expertise. KARL is part of the MIKE approach (Model Based and Incremental Knowledge Engineering) (cf. [AFL93]) which aims at integrating the advantages of life cycle models, prototyping approaches and formal specification techniques into a coherent framework for the knowledge engineering process. In this paper we provide an overview of KARL which is used to formalize and operationalize models of expertise.

The main features of KARL may be summarized as follows:

- KARL offers adequate modelling primitives which allow the explicit representation of different kinds of knowledge. For this purpose we used the model of expertise as it is defined in the KADS-I project (cf. [WSB92]) as a framework and extended it by ***hierarchical refinement*** and ***structuring primitives*** (specifications of real life systems, soon become rather complex). These topics are discussed in section two to six.
- Normally, formal specifications are difficult to understand. Therefore, ***graphical representations*** with defined semantics are included in KARL. KARL offers graphical representations of most modelling primitives to improve understandability. Every graphical symbol has a defined meaning given by the semantics of the corresponding language primitive: Enhanced-Entitity-Relationship (EER) diagrams (cf. [ElN89]) for the domain layer, levelled data flow diagrams (cf. [You89]) for the inference layer, and structured control flow diagrams for the task layer.
- KARL allows a ***formal specification*** of the model of expertise to avoid ambiguities and to reason about properties of the modelled expertise. This specification is automatically mapped to an operational one allowing to validate the specified expertise by testing. Thus the model of expertise may be developed by ***explorative prototyping*** (cf. [Flo84]). The model theoretic semantics of KARL and its operationalization are sketched in section seven.

Chapter eight lists some applications of KARL and chapter nine discusses related work.

Due to the limited space of the paper, we focus our attention to the modelling primitives of KARL. The formal semantics of KARL and its operationalization are sketched only.

2 A model of expertise in KARL

A model of expertise according to the KADS-I project (cf. [WSB92]) separates different kinds of knowledge and provides different epistemological primitives for each of these types of knowledge. The result of the knowledge acquisition phase consists of a four layer model of expertise, each layer containing a special type of knowledge. We restrict ourselves to three of them only because there is no general agreement about the content of the fourth layer the so-called strategic layer[2].

- The ***domain layer*** contains domain specific knowledge about concepts, their features, their elements, and their relationships. This knowledge is task-neutral in the sense that there are no means to control its use. At this layer the objects and the terminology of the domain are described which the knowledge-based system should use.
- The ***inference layer*** contains knowledge about the used problem-solving method. This layer indicates which inferences are necessary within the problem-solving method. This knowledge describes the

1. An early version was reported in [FAL91]. The new version allows the reasoning about classes, i.e. reasoning about predicates, integrates functional and set-valued attributes, equality-reasoning, extends the semantics of inference actions and defines a model theoretic semantics for all layers of KARL including the task layer.

2. The content of his layer has even been described in a very vagious manner by the KADS group.

application of the domain knowledge for inferencing. It describes the logical inferences and their dependencies. No causal dependencies or causal orderings between these logical implications are given.

- The ***task layer*** contains knowledge about the control flow of a problem-solving method in order to solve a specific task. This layer specifies when inferences are made.

3 The Domain Layer

In KARL we adopt the idea of object-oriented data modelling. An object denotation is a reference to a real-world object. This object can be described by attribute values and there is a functional dependency from the object´s identity to its attribute values. Objects are grouped into different classes. The class definition describes class attributes which refer to the class as such and attributes for the objects which are elements of the class. The attributes are described by their name and their range. Classes are arranged in a specialization/generalization hierarchy. Examples for class definitions are shown in the following:

```
CLASS patients
   ELEMENT_ATT
      name : {STRING};
      health_insurance: {agencies, registered};
      ...
END;
   CLASS_ATT
      health_insurance_companies:: {agencies};
END;
```

```
CLASS released_patients
   ISA patients;
   ELEMENT_ATT
      release_date: {date};
END;
```

The first class definition describes the class “patients”. For the elements of this class the element attributes “name”, and “health_insurance” are given. For each attribute the range is described by a set of classes, e.g. {agencies,registered}, which means that a value of the attribute is an element of all these classes, e.g. an element of the class “agencies” and and element of the class “registered”. The attribute “health_insurance_companies” is used to describe the class itself. It is a set-valued attribute, i.e. it may contain the set of all health-insurance companies of the patients. The class “released_patients” is a subclass of the class “patients”. This means that every element of the class “released_patients” is also an element of the class “patients”. In addition, all attributes and their range restrictions which have been supplied for the class “patients” are also available for the class “released_patients”.

KARL provides a logical language in order to be able to formulate intensional relations and facts. Together with the class definitions it is based on F-Logic (cf. [KLW90]) for which a declarative semantics is described for inheritance, typing, and logical expressions about objects and classes.

Elements are denoted by ***element-id-term***s, consisting of variables, functions, or element-constants, similar to terms in first-order logic. By means of functions it is possible to generate new object identifiers in logical expressions. This way to generate new objects is based on O- and F-Logic (cf. [KLW90, KiW93]). Classes are denoted by ***class-id-terms***, which consist of variables or class constants. The class constants are the class names of the class definitions. A ***value-id term*** denotes a value, i.e. an integer or a string.

The basic ingredients of logical expressions in KARL are F-terms:

- “$e \varepsilon c$” is an element-term, where “e” is an element-id-term, “c” is a class-id-term. An element-term describes that an object “e” is an element of class “c”.
- “$c \leq d$” is an is-a-term, where “c” and “d” are class-id-terms. An is-a-term expresses that a class “c” is a subclass of class “d”. Using variables within “c” or “d” it is possible to browse the class hierarchy using this term.
- “o[..., a:T,..., s::{S_1,..,S_n},...]” is a data-term, where “o” is either an element-id-term or a class-id-term, “T”,”S_i” are data-terms. “a” is an attribute name of a single-valued attribute, “s” of a set-valued

attribute. A data-term defines attribute values for the object which is referred to by the id-term "o".

- "$e \cong d$" denotes an equality-term, where "e" and "d" are id-terms. This means that "e" and "d" denote the same element, class, or value.

In addition, P-terms "$p(a_1:T_1, ..., a_n:T_n)$" allow to express relationships between data-terms "T_i" in a similar way as in predicate logic. The arguments of a predicate are named by the "a_i"´s in order to improve readability.

Logical formulae are built from F-terms in the usual way using logical connectors $\wedge$ (and), $\vee$ (or), $\neg$ (not), $\longleftarrow$ (implication), brackets, and variable quantification.[1]

For the description of sufficient conditions Horn rules extended by stratified negation are used (cf. [Llo87]). Necessary conditions are conditions which are not used to derive new facts, but which must hold for all facts which are extensionally or intensionally (by rules) defined. These conditions are described by arbitrary logical formulae.

Using the class definitions above the following examples denote rules or facts:

- rst[name:"Rudi Studer", age: 42].
  The object "rst" has the name "Rudi Studer" and the age "42".
- rst ε patients.
  The object "rst" is an element of class "patients".
- patients[health_insurance_companies:: {X}] $\longleftarrow$ Y[health_insurance: X] ε patients[2]
  The class "patients" has all the health-insurances companies of its elements as attribute value for the set-valued attribute "health_insurance_companies".

The class definitions together with their attribute definitions and is-a relationships are used to define several well-typing conditions for domain and range of an attribute.

4 The Inference Layer

At the inference layer generic inferences are represented which use domain specific knowledge. These two kinds of knowledge must be separated clearly, i.e. for both kinds of knowledge, specific language primitives with clearly separated functions must be offered. By this way reusability may be supported in two ways. On the one hand the knowledge about the problem-solving method is domain independent, i.e. it may be used in several domains. On the other hand the same domain knowledge may be used for different problem-solving methods and different, though related tasks. Therefore domain knowledge may be reused as well. In the following the modelling primitives used at the inference layer are described.
Roles at the inference layer have two different tasks. First, they establish the connection of the domain layer and the inference layer. The generic inference actions can use the domain knowledge by means of a role. Roles which are connected to the domain layer constitute a ***view*** of the problem-solving method to the domain knowledge and case data (***upward mapping***). In addition, a role can be used to map results of the inference layer, which are expressed in generic terms, back to domain layer expressions. The result of a problem-solving process, which is described by generic terms of the inference layer are rephrased in domain-specific terms by means of a ***downward mapping*** of a role. Second, roles contain intermediate data of the problem-solving process. Such roles collect the output of an inference action and provide input for other inference actions.

Therefore, in KARL three types of roles are distinguished. A ***View*** (input role of an inference action) reads knowledge and data from the domain layer, which serve as input to an inference action. ***Terminators*** write results of the problem-solving process to the output data part of the domain layer. ***Stores*** contain

1. Variables which are not explicitly quantified are all-quantified.

2. "o[...] ε c" is a short-cut for "o[...] $\wedge$ o ε c".

intermediate data at the inference layer.

The definition of a role consists of a set of ***class-definitions*** which define the content of a role, an ***upward mapping*** which defines a connection from the domain layer to the role if the role is a view, a ***downward mapping*** which defines a connection from the role to the domain layer if the role is a terminator.

An example for a simple store definition is shown in the following:

```
STORE set_of_possible_hypotheses
    CLASS set_of_possible_hypotheses
        ELEMENT_ATT:
            evidence :{REAL};
    END
```

The store “set_of_possible_hypotheses” contains a set of objects which are described by the attribute “evidence”.

A knowledge source or elementary inference action has one or more premise roles and conclusion roles. For every elementary inference action the names of the premise and conclusion roles and a declarative description of the relationship between input and output has to be given. The latter is expressed using Horn rules with stratified negation as described above. A very simple inference action "diagnose", for instance, can be described as follows:

```
INFERENCE ACTION diagnose
    PREMISES     abstract_symptoms, class_recognition_relation;
    CONCLUSIONS set_of_possible_hypotheses;
    RULES
                I[evidence : Y] e set_of_possible_hypotheses ¨
                        C e abstract_symptoms Ÿ
                        class_recognition_relation(symptom: C, illness: I, evidence: Y).
END
```

This specifies that an inference action with name “diagnose” is supplied with input by the premise store “abstract_symptoms”, transforms these data using the view “class_recognition_relation”, and writes the results to the conclusion store “set_of_possible_hypotheses”.

A store may be a conclusion store for one inference action and a premise store for another inference action. Thus inference actions and stores are related in a graph structure, the so called ***inference structure***. It should be noticed, that no control knowledge concerning the ordering of the inference actions is represented at the inference layer.

A subgraph of an inference structure can be combined to one ***composed inference action*** and, vice versa, an inference action can be refined to a network of inference actions and stores. Therefore, the inference processes can be described at several levels of granularity. The main advantages of ***hierarchical refinement at the inference layer*** are:

- Modelling complex systems in one inference structure would result in an unreadable inference layer.
- It is possible to define several views to a system or a problem-solving method. Each of the views shows the method at a different level of granularity.
- If a system requires the cooperation of several predefined problem-solving methods, any of these methods can be represented by a composed inference action in one inference structure. The representation of the interaction of the methods is not mixed with their internal structure.

The semantics of a composed inference action is defined by its constituents, i.e. the elementary inference actions it contains and the control flow between those constituents (which is described at the task layer).

5 The Connection between Inference and Domain Layer

Because the description of the problem-solving steps and their interaction should be generic, an appropriate view to the domain knowledge - similar to views in the area of information systems (cf. [ElN89]) - must be defined. It defines how domain specific knowledge and case-specific data are used for the inferences. This requires that for views and terminators a connection to the domain layer must be established. These mappings from the domain layer to roles is described by a set of stratified Horn rules in KARL.

An example for such a connection is shown in the following:

class_recognition_relation(symptoms: X, illness: Y, evidence: Z) $\leftarrow$
    X[temp_intervall: T, age_intervall: A, sex: S] ***e*** abstracted_description $\wedge$
    indicated_by(illness: Y, temp_intervall: T, age_intervall: A, sex: S, evidence: Z).

This expression states that the view “class_recognition_relation” which an inference action uses corresponds to the conjunction of the domain specific predicate “indicated_by” and elements of the domain class “abstracted_description” having the corresponding values for the attributes “temp_intervall, “age_intervall”, and “sex”.

Terminators containing the output of the problem solving process are connected in the same way “downward” to the domain layer. By this way the generic knowledge at the inference layer is rephrased in domain specific terms.

6 Task Layer

Experiences in XCON (cf. [SBJ87]) show that great problems arise if control flow is only specified implicitly, e.g. in the rules. In KARL knowledge about control flow is clearly separated from other kinds of knowledge by describing it at a separate layer of the model of expertise, namely the task layer. So the ***task layer*** describes the sequence in which inferences are performed.

In KARL control flow is specified similar to procedural programming languages. One design goal of KARL has been to give it a declarative character, i.e. its model-theoretic semantics. We achieved this by applying ***Dynamic Logic*** (cf. [Koz90]) for specifying the semantics of the KARL primitives for the task layer.

At the task layer a number of functions $F = \{f_1, f_2, ..., f_r\}$, which correspond to the inference actions at the inference layer and a number of role variables $R = \{r_1, ..., r_n\}$, which refer to the contents of the roles are available.

The elementary boolean formula $\varnothing(s_i, c)$ evaluates to true if the class c within the store s_i contains no element. Arbitrary boolean formulae may be combined from elementary boolean formulae using the usual connectors $\neg$, $\vee$ and $\wedge$.

A primitive program is a ***call of a function***

$(r_{k1}, ..., r_{kh}) := f_i(r_{j1}, ..., r_{jl})$

which corresponds to an elementary inference action “f_i”. The “r_{ks}“ denote conclusion roles and the “r_{js}“ denote premise roles of it.

Composed programs may be built up from primitive programs and boolean formulae as conditions using ***sequence***, ***branch*** and ***loop*** statements.

Programs may be combined into named ***subtasks*** which correspond to composed inference actions of the inference layer. So hierarchical refinement is supported at the task layer too.

In the following a small example for a program at the task layer is given:

```
abstract_symptoms := abstract(symptoms);
set_of_possible_illnesses := diagnose_illness (abstract_symptoms);
illness := refine(set_of_possible_illnesses)
```

7 Semantics and Operationalization

The semantics of an elementary inference action is the ***perfect Herbrand model*** (cf. [KLW90]) of the facts in the premise roles, the rules of the inference action and if the inference action is connected to the domain layer the mapping rules of the views and terminators and the rules and facts of the domain layer. If this perfect Herbrand model is not well-typed an error is indicated. The content of a conclusion role (an inference action can have several conclusion roles) is determined as that subset of the Herbrand model which contains only expressions over the terminology (i.e., the according class definitions) of the conclusion store.

To define a declarative semantics for the task layer we use ***dynamic logic*** (cf. [Koz90]). Every inference action defines a mapping from the facts of the premise roles to the facts of the conclusion roles. These mappings are used to interpret the function symbols of the task layer. The actual content of the roles define the actual value of the role variables at the task layer. A ***state*** is characterized by the content of the roles, i.e. the values of the role variables.

An elementary program is interpreted as a binary relation between two succeeding states, where the latter one results from the previous one by the new variable assignments. Composed programs are interpreted as composition of the relations which interpret the elementary programs from which the composed program is built up. A logical formulae is interpreted by a subset of all possible states for which the formulae is regarded as true.

For KARL an operational semantics has been defined which is the basis for the implementation of an interpreter for KARL. The semantics of a set of rules and facts is defined by the ***fixed point*** of an ***immediate consequence operator***. This immediate consequence operator also has to consider the equalities which are defined by the equality-terms. The fixed point has been proven to be equivalent to the perfect Herbrand model. The semantics of the execution of an inference action is defined by the change of the state caused by the change of the conclusion stores of the inference action.

The operational semantics of a whole KARL-program is defined as the semantics of a ***deterministic while-program*** where the activations of inference actions are primitive programs.

Based on the fixed point semantics an algorithm for the evaluation of rules and facts has been developped which extends the ***dynamic filtering algorithm*** (cf. [KiL86]) by ***equality-reasoning***. This algorithm is complete and due to the set-oriented evaluation very efficient if all the solutions of a posed query has to be computed which is the case in KARL.

Based on the operational semantics and on the optimized evaluation strategy a debugger has been developed which integrates the interpreter for KARL. This tool has been proven to be very useful in order to validate the specification by testing.

8 Applications

KARL has already been applied to several problems. The Sisyphus sample problem has been posed in order to allow for the comparison of different modelling approaches and languages. The problem in Sisyphus consists of finding a consistent assignment of a set of employees to appropriate office rooms. When assigning employees to offices, certain requirements have to be taken into account (cf. [AFL92a]). In [LHS92], a KARL model for a problem in the context of air pollution control is described. This application is concerned with the configuration of an optimal emission control measure, i.e. a combination of individual emission control technologies, for a given plant. The selection of scheduling algorithms for project management has also been modelled in KARL ([KFG92]). Starting from a description of the context where the scheduling algorithm should be applied, all algorithms appropriate for this context are selected. The performance component of MOLE (cf. [Esh88]) which has been identified as a general problem-solving method for diagnosis tasks has been remodelled in KARL (cf. [Ang92]). Similar to the Sisyphus example a design task has been posed in order to compare different modelling approaches and languages. A variant of the problem-solving method propose and revise (cf. [Mar88]) has been modelled for scheduling a set of activities to a set of time periods with respect so some given requirements (cf. [LFA93]).

9 Comparison with Related Work

FORKADS (cf. [Wet90]), OMOS (cf. [Lin92]), and Model-K (cf. [KaV92]) are languages which also aim at operationalizing models of expertise. However no declarative semantics for these languages are given. OMOS has only a very restricted expressive power, since it is not Turing complete. Model-K offers the full expressive power of Babylon, a hybrid knowledge representation shell, but enforces a programming step in order to operationalize the elementary inference actions. FORKADS avoids this by a set of predefined primitive inference actions. Until now, it is not clear whether this set is complete, i.e. how large a complete set of elementary inference actions must be.
$(ML)^2$ (cf. [HaB92]), which has been developed as part of the KADS project, is a language which mainly aims at formalizing models of expertise. $(ML)^2$ uses order-sorted first-order logic to model the domain layer, meta-logic to model the inference layer, and dynamic logic to model the task layer. Besides all technical details the main difference lies in the fact that $(ML)^2$ does not stress the operationalization of a model of expertise, i.e. it is not meant as a tool to support explorative prototyping. Prototyping implies not only executability in principle but executability under time restrictions. First-order logic, as proposed for the domain layer requires a theorem prover in order “to compute” results, an approach which is not yet very efficient. In addition, full first-order logic is only semi-decidable, i.e. the possible operationalization by a theorem prover is incomplete.

10 Conclusion

We have presented KARL, a formal language for the representation of models of expertise. The modelling primitives KARL and their graphical representation allow to describe the model of expertise at a high level of abstraction which is a precondition for understandability, maintainability and reuse. These primitives provide means for communicating the contents of the model to the expert. The formal semantics of KARL allows to describe the model of expertise in an unambiguous manner and thus avoids the interpretation of this model in different ways. The operational semantics of KARL allows to execute the model of expertise and thus provides means for validating it in cooperation with the expert and the knowledge engineer.

References

[Ang92] J. Angele: Cover and Differentiate remodelled in KARL. In *Proceedings of the 2nd KADS User Meeting*, Munich, February 17-18th, 1992, C. Bauer and W. Karbach (eds.), GMD Studie no 212, St. Augustin, 1992.

[AFL92a] J. Angele, D. Fensel, and D. Landes: An executable model at the knowledge level for the office-assignment task. In M. Linster (ed.): Sisyphus ´92: Models of Problem Solving, Arbeitspapiere der GMD, no 663, St. Augustin, July 1992.

[AFL93] J. Angele, D. Fensel, D. Landes, S. Neubert, and R. Studer: Model-based and Incremental Knowledge Engineering: The MIKE Approach. In J. Cuena (ed.), *Proceedings of the IFIP TC12 Workshop on Artificial Intelligence from the Information Processing Perspective - AIFIPP´92*, Madrid, Spain, 14-15 September, 1992, Elsevier Science Publisher B.V., Amsterdam, 1993.

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

[ElN89] R. Elmasri and S.B. Navathe: *Fundamentals of Database Systems*, Benjamin/Cummings, Houston, 1989.

[Esh88] L. Eshelman: MOLE: A Knowledge-Acquisition Tool for Cover-and-Differentiate Systems. In: S. Marcus (ed.): *Automating Knowledge Acquisition for Expert Systems*, Kluwer Academic Publishers, Boston, 1988.

[FAL91] D. Fensel, J. Angele, and D. Landes: KARL: A Knowledge Acquisition and Representation Language. In *Proceedings of expert systems and their applications, 11th International Workshop, Conference "Tools, Techniques & Methods"*, 27-31 Mai, Avignon, 1991.

[FBA93] K. M. Ford, J. M. Bradshaw, J. R. Adams-Webber, and N. M. Agnew: Knowledge Acquisition as a Constructive Modeling Activity. In *International Journal of Intelligent Systems, Special Issue Knowledge Acqisition as Modeling*, part I, no 1, vol 8, 1993, pp. 9-32.

[Flo84] C. Floyd: A systematic look at prototyping. In R. Budee et al. (eds.), *Approaches to Prototyping*, Springer-Verlag, Berlin, 1984.

[HaB92] F. v. Harmelen and J. Balder: $(ML)^2$: A formal language for KADS conceptual models. In *Knowledge Acquisition 4(1)*, 1992.

[KaV92] W. Karbach and A. Voß: Reflecting about expert systems in MODEL-K. In *Proceedings of expert systems and their applications, 12th International Workshop, Conference "Tools, Techniques & Methods"*, May, Avignon, 1992.

[KiW93] M. Kifer and J. Wu: A Logic for Programming with Complex Objects. To appear in *Journal of Computer and Systems Science*, 1993.

[KFG92] R. Köppen, D. Fensel, J. Geidel: Modelling the Selection of Scheduling Algorithm with KARL. In *Proceedings of the 2nd KADS User Meeting*, Munich, February 17-18th, 1992, C. Bauer and W. Karbach (eds.), GMD Studie no 212, St. Augustin, 1992.

[KiL86] M. Kifer, E. Lozinskii: A Framework for an Efficient Implementation of Deductive Databases. In *Proceedings of the 6th Advanced Database Symposium*, Tokyo, 29.-30. August 1986.

[KLW90] M. Kifer, G. Lausen, and J. Wu: Logical Foundations of Object-Oriented and Frame-Based Languages. In Technical Report 90/14 (2nd revision), Department of Computer Science, SUNY at Stony Brook, NY, August 1990.

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

[LFA93] D. Landes, D. Fensel, and J. Angele: Formalizing and Operationalizing a Design Task with KARL. In J. Treur et al. (eds.), *Formal Specification of Complex Reasoning Systems*, Proceedings of the ECAI-92 Workshop on Formal Specification Methods for Complex Reasoning Systems, European Conference on Artificial Intelligence (ECAI-92), Wien, Austria, August 3-7, 1992, Ellis Horwood, Chichester, 1993.

[LHS92] D. Landes, D. Hackenberg, and T. Schweier: An Inference Structure for a Configuration Problem. In *Proceedings of the 2nd KADS User Meeting*, Munich, February 17-18th, 1992, C. Bauer and W. Karbach (eds.), GMD Studie no 212, St. Augustin, 1992.

[Lin92] M. Linster: Knowledge Acquisition Based on Explicit Methods of Problem Solving, PhD thesis, University of Kaiserslautern, 1992.

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

[Mar88] S. Marcus: SALT: A Knowledge-Acquisition Tool for Propose-and-Revise Systems. In: S. Marcus (ed.): *Automating Knowledge Acquisition for Expert Systems*, Kluwer Academic Publishers, Boston, 1988.

[New82] A. Newell: The knowledge level, *Artificial Intelligence*, vol 18, 1982.

[SBJ87] E. Soloway, J. Bachant, and K. Jensen: Assessing the Maintainability of XCON-in-RIME: Coping with the Problems of a VERY large Rule-Base. In *Proceedings of 6th National Conference on AI (AAAI'87)*, AAAI, Seattle, Washington, July 13-17, 1987.

[Wet90] T. Wetter: First order logic foundation of the KADS conceptual model. In B. Wielinga et. al. (eds.), *Current Trends in Knowledge Acquisition*, IOS Press, Amsterdam, 1990.

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

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