View metadata, citation and similar papers at core.ac.uk brought to you by CORE
provided by KITopen

In B. Wolfinger (ed.), Innovation bei Rechen- und Kommunikationssystemen, Informatik Aktuell, Springer-Verlag, Berlin, 1994.

Integration Of Semiformal And Formal Methods For Specifying Knowledge-Based Systems

Dieter Fensel and Susanne Neubert
Institute AIFB, University of Karlsruhe, 76128 Karlsruhe, Germany
e-mail: {fensel | neubert} @aifb.uni-karlsruhe.de

Abstract. The paper describes a specification approach for knowledge-based systems (kbs) combining semiformal and formal specification techniques. The semiformal knowledge representation uses a hypermedia-based formalism which serves as a communication basis between expert and knowledge engineer. This representation is also the basis for the formalization process resulting in the formal and executable model of expertise written in KARL. A smooth transition from the semiformal to the formal specification is enabled as both description techniques use the same conceptual model to describe the system.

1 Introduction

Originally, expert systems or knowledge-based systems (kbs) were developed using the rapid prototyping approach. The acquired knowledge was immediately implemented and the running prototype was used as a guide for the further knowledge acquisition process. The distinction of symbol level and knowledge level [New82] created the conceptual framework for a different process models for the development of kbs. A knowledge level description of the *task* solved by the system and the *knowledge*, which is required to solve the task, is constructed during a modelling activity. This knowledge level description is built independently of the design and implementation activity. The separation of analysis and design/implementation resembles a lesson learnt in software engineering. In response to the so-called software crisis in the late sixties, methodologies, process models, methods, and tools have been developed to maintain the software development process and its results. A significant result was the separation of the description what a system should do from how this can be achieved by a specific implementation, i.e. the separation of analysis or requirement engineering at the one hand and design and implementation at the other hand. As a result, several description techniques have been developed to describe the specification as it emerges from the analysis step. Mainly, these specification techniques follow three lines:

- *Informal specification techniques* like structured analysis or object-oriented analysis allow the description at a high and informal level. These approaches broadly use graphical means like entity-relationship diagrams, dataflow diagrams, flow charts, and state-transition diagrams. The specifications are easy to understand and very useful as a mediating representation for the communication between user and system developer.
- *Formal specification techniques* like Z or VDM allow a unique and detailed specification of the functionality of a system. In the case of Z, a software system is specified as a partial mathematical function by applying the theory of finite sets. Specifications can be checked via formal methods.
- *Executable specification techniques* like PAISLey add the flavour of prototyping to the specification process. The results can be evaluated by a running prototype. Often, this is

nearly the only way to end up with realistic descriptions of the desired functionality of the systems.

Several authors argue for the combination of these description techniques so as to overcome the disadvantages when used stand-alone. Informal specifications contain ambiguity and contradictions and lack precision. Conversely, formal descriptions and their formal semantics are hard to understand and it is very difficult to extract an intuition about the functionality of a system given the huge amount of details of a formal specification only. The need for the combination becomes obvious regarding the two different purposes of specifications [FBA+93]. First, it should serve as a *mediating representation* supporting the communication between the user and the system developers. In the case of kbs, it should mediate the communication between user and expert at the one hand and the knowledge engineer at the other hand. Second, it should serve as an *intermediate representation* closing the gap between an intuition about the functionality of a systems and its actual design and implementation.

The integrated development of semiformal and formal specification techniques as discussed in this paper is part of the *MIKE-approach* (*Model-based and Incremental Knowledge Engineering*) [AFL+93], which aims at a development method for kbs covering all steps from initial specification (knowledge acquisition) to design and implementation. In fact, we present the semiformal hypermedia-based formalism *MEMO* [Neu93], the formal and executable *Knowledge Acquisition and Representation Language KARL* ([FAL91], [AFS94]) and their relationships.

The contents of the paper are organized as follows. In section two, the semiformal models are described. Then, the formal specification language KARL and the model of expertise are discussed in sections three. Section four shows how both specification are related.

2 The Semiformal Models Of A Knowledge-Based System

Problems in directly developing a formal specification from the knowledge protocols lead in MIKE to construct mediating representations before starting the formalization process [HoN92]. Our mediating representations describe protocols, concepts and activities, hierarchies of modelling primitives, data flow and control flow of activities etc. The development of semiformal mediating representations provides different advantages. The expert can be integrated in the knowledge engineering process of structuring the complex knowledge so that the knowledge engineer is able to interpret and formalise it. Thus, the cooperation between expert and knowledge engineer is improved. Moreover, the formalization process is simplified. In addition, a mediating representation is also a basis for documentation and the explanation facility.

For our mediating representations we developed a semiformal, hypermedia-based formalism ([Neu93], [NeO92]). In fact, two semiformal models (the *elicitation model* and the *structure model*) have been developed which are sets of special node and link types. A *node* is a hypermedia document with a content using text, graphics, audio or video to describe an state/ process/concept. A *link* describes a relationship between two nodes. A link is directed. Links are defined by a source node, a destination node, a link name, a link type, and an explanation field. *Contexts* establishes a specific view on a set of nodes and links.

The first model, the *elicitation model*, documents the elicitation process. Thus, it includes knowledge protocols which are stored in *protocol nodes*. Additionally, *date links* between protocol nodes are included to describe the elicitation ordering.

The *structure model* which is developed on the basis of this first collection of protocols contains a more structured description of knowledge. It is built up by the following description elements:

- The *activity context* includes all *activity nodes* which describe a step of the problem-solving process. Additionally, *refinement links* are integrated. This context enables a view on the complete hierarchy of activity nodes and their subactivity nodes. Every activity node has to be a refinement of another activity node except for the global activity node which characterizes the whole problem-solving process.
- An *ordering context* provides a view on *activity nodes* which are related by *ordering links*. These activity nodes lie on *one* hierarchy level. One activity node can be the source-node or the destination-node of different ordering links. This means that different activity nodes are alternative options to solve the problem.
- The *concept context* encompasses all concept nodes which serve as descriptions of the static objects. Moreover, all links between two concept nodes, so-called *is_a links* and self-defined *relationship links*, are included. Relationship links can be added by the user to describe an arbitrary relationship between two concepts.
- A *structure context* is also a view on *one* hierarchy level of *activity nodes*. Here, activity nodes are related with concept nodes by so-called *dataflow links*. A structure context gives the flow of data produced during the problem solving process.

An example for the two models is sketched at the left side of Figure 1. The *Sisyphus problem*[1] is an assignment problem in which employees are assigned to office places with several requirements to be met [Lin92]. The whole problem is divided into three subactivities, to create pairs of employees and places, to prune faulty pairs and to check whether a solution has been found (i.e., whether a placement is complete and correct).

3 The Formal Model Of A Knowledge-Based System

3.1 The KARL Model of Expertise

The conceptual model underlying KARL is derived from the KADS *model of expertise* [SWB93] and distinguishes four types of knowledge. Three of them define static knowledge, whereas the task layer is used to define the dynamics of the problem-solving process.

Domain knowledge consists of static knowledge about the application domain of the system. The domain knowledge should define a conceptualization of the domain as well as a declarative theory providing all the knowledge required to solve the given tasks. KARL integrates frames and logic for the domain layer by providing the sublanguage *Logical-KARL (L-KARL)* for this purpose. Terminological knowledge can be described by a taxonomy of concepts. For each concept, attributes can be defined and are inherited according to the taxonomy. Further knowledge can be described with logical formulae.

Inference knowledge specifies the *inferences* that can be made using the domain knowledge, and the *knowledge roles,* which model input and output of the inferences. KARL distinguishes three types of knowledge roles. Roles which deliver domain knowledge to an inference action are called *views*, roles which model the dataflow dependencies between inference actions are called *stores*, and roles which are used to write final results back onto the domain layer are called *terminators*. The inferences and roles together with their dataflow dependencies

1. Sisyphus is a project that aims at comparing different approaches of knowledge engineering.

constitute a description of the problem-solving method applied. In addition to its use at the domain layer, L-KARL is used to specify the logical relationship defined by an inference action at the inference layer and to specify a *task-specific terminology* independently from the domain-specific terminology by means of concept definitions in roles.

A *Domain view* specifies the relationship between the generic terms used at the inference layer and the domain-specific knowledge. Again, L-KARL is used to specify the mapping between domain and inference knowledge.

Dynamic control knowledge: The purpose of the task layer is to specify *control* over the execution of the inferences of the inference layer. The sublanguage *Procedural-KARL (P-KARL)* is used to specify this dynamic knowledge via sequences, branches, loops, and procedure calls. Conditions for the controlflow can be specified via logical statements about the contents of stores.

Inference and control knowledge are domain independent, i.e. they describe the problem solving process in a generic way. Thus, such a so-called *problem-solving method* can be reused for different application problems. MIKE provides a library, where these generic problem solving methods are stored which are described formally and informally.

The right side of Figure 1 sketches a model of expertise of the *Sisyphus problem*. The domain terminology and the domain knowledge required by the problem solving method is defined at the domain layer. The inference layer contains the elementary inference steps and knowledge roles of it. Components (employees) and slots (places) are combined by the inference action *create*. *Prune* eliminates illegal states, and *check* searches for valid solutions. The control flow between these inferences is defined at the task layer.

3.2 The Knowledge Acquisition and Representation Language (KARL)[2]

Logical-KARL (L-KARL)

L-KARL is a customization of Frame-logic (F-logic) [KiL93]. F-logic and L-KARL enrich the modelling primitives of first-order logic by syntactic modifications but preserve the model-theoretical semantics of it. In this way, ideas of semantical and object-oriented data models are integrated into a logical framework enabling the declarative description of terminological as well as assertional knowledge. L-KARL distinguishes classes, objects, and values. It provides classes and an is-a hierarchy with multiple attribute inheritance to describe terminological knowledge. Intentional and factual knowledge is described by logical relationships between classes, objects, and values.

A *class* or *concept definition* which corresponds to a frame 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, their domain, and their range. Classes are arranged in a specialization/generalization hierarchy with multiple attribute inheritance. Attributes can be single-valued or set-valued. Attributes can be used to describe objects as well as classes. They have defined domain and range types.

The literals of logical expressions in L-KARL are *is-element-of literals* which describe that objects are elements of classes; *is-a literals* which describe subset relationships between classes; *equality literals* which describe equality of objects, classes, and values; and finally *data literals* which define attribute values for objects and classes. Logical formulae are built from these literals using logical connectors $\wedge$, $\vee$, $\neg$, $\leftarrow$ and variable quantification. The

2. A complete description of KARL can be found in [Fen93]. A short description of the modelling primitives of KARL is given in [AFS94], some of the applications of KARL can be found in [AFL92b], [LFA93], and [PFL+94].

logical language to describe relationships between classes, objects, and values is Horn logic with equality and function symbols extended by stratified negation [Ull88].

Procedural-KARL (P-KARL)

In KARL knowledge about controlflow is explicitly described by the logical language P-KARL. The control flow is specified similar to procedural programming languages. For a P-KARL program, a number of functions $F = \{f_1, f_2, ..., f_r\}$ and a number of variables $\{X_1, ..., X_n\}$ are available. The function symbols correspond to names of inference actions. The variables address their stores. The actual parameters of a function are the input stores of the corresponding inference action and the results of the function are mapped to its output stores. A primitive program is an *assignment*

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

f_i corresponds to an inference action and the X_{ks} denote its output stores and the X_{js} its input stores. A composed program is defined as *sequence*, *loop*, or *alternative* of programs.

KARL As A Formal And Executable Specification Language

The KARL model of expertise contains the description of domain knowledge, inference knowledge, and task knowledge (i.e., procedural control knowledge). The gist of the matter of the semantics of KARL is therefore the requirement to include the specification of static and procedural knowledge. For this purpose, two different types of logic have been integrated. The sublanguage L-KARL, which is based on object-oriented logics, combines frames and logic to define terminological as well as assertional knowledge. The sublanguage P-KARL, which is a variant of dynamic logic, is used to express knowledge about the control flow of a problem-solving method in a procedural manner. The representation of the interaction of both types of knowledge is reached by combining both types of languages. For more details see [Fen93]. Based on this semantics an operationalization and an optimized evaluation strategy were developed providing an interpreter and debugger for KARL.

KARL As A Graphical Modelling Language

KARL provides graphical representations of most modelling primitives to improve their intelligibility: A variant of *Enhanced-Entity-Relationship (EER) diagrams* describes the domain layer, a variant of *levelled dataflow diagrams* is provided for the inference layer, and a variant of *programflow diagrams* describes the task layer. All three graphical representations include *hierarchical refinement* to allow to represent the system on different levels of refinement. Figure 1 shows the graphical representation a model of expertise of a solution of the so-called *Sisyphus problem*.

4 Integration Of Semiformal And Formal Specification

The integration of semiformal and formal specification techniques can be discussed in two dimensions. First, we will sketch their integration during the specification *process*. Then, we will sketch their integration in the specification *product*.

4.1 Integration During the Process of Model Development

The knowledge acquisition process consists of three activities: eliciting knowledge, interpreting knowledge, and formalizing knowledge. These activities are done in a cyclic manner. The process starts with selecting a partially specified structure model from a library of predefined models. These models (socalled problem-solving methods) are developed for specific problem types like classification, diagnosis, assignment, configuration, planning etc.

(see [Neu94] for more details). Then, the selected structure model guides the elicitation process, i.e. it is used as a guide for asking and observing the expert. It defines a network of activities and knowledge items which are used as a form for the elicitation process. The resulting knowledge protocols are stored in the node-content of protocol nodes of the elicitation model. The structures described in the knowledge protocols are represented by contexts of the structure model. The semiformal structure model is a first result of specification which clarifies complex knowledge structures. At the one side, it guides the elicitation process. At the other side, it becomes modified and refined as a result of the elicitation process. Moreover, the structure model is the foundation for the formalization process where the model of expertise is developed. The formalization is achieved in a refinement step which

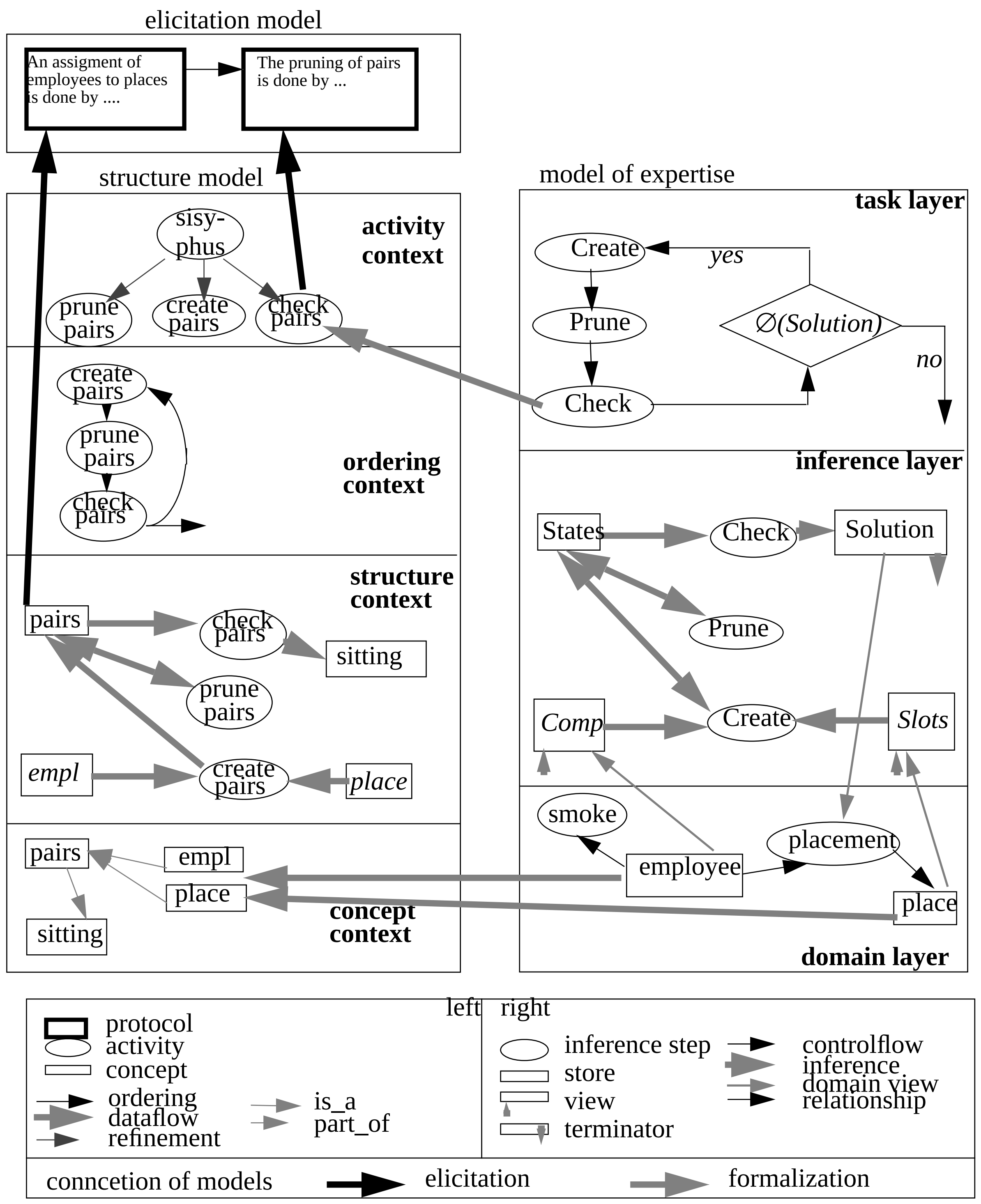

Fig. 1. Parts of an elicitation model and a structure model at the left (nodes have informal content) and of a model of expertise at the right (nodes with formal content).

supplements informal descriptions of elementary activities and knowledge entities by formal definitions. Again, this can lead to revision of the whole structure of the model of expertise and can stimulate new elicitation tasks. The main features of our approach are therefore the integration of bottom-up and top-down modelling by applying reusable components (problem-solving methods) and the smooth transition from informal to semiformal to formal specification techniques. A large amount of the formal specification can be done graphically using the same modelling primitives as in the semiformal model. The underlying conceptual model of the system (i.e., the model of expertise) relates both description techniques.

4.2 Model connection

A further central point of MIKE is to relate the three models with each other. First, the elicitation model is related with the structure model. More precisely: the activity and concept nodes are related with the protocol nodes in which they have been described during elicitation. So, a connection is existing to the originally given information from the expert. A so-called *elicitation link* exists to describe these interrelation. During structuring, these links can be easily integrated. Second, the structure model is related with the model of expertise[3]. Socalled *formalization links* relate a formally described node of the model of expertise with an informally described node of the structure model. Corresponding nodes, including on the one hand an informal description and on the other hand a formalization, are linked during the formalization/operationalization process.

The model connection has different advantages: First, the informal information integrated in the elicitation or structure model function as a documentation of a formal description. Large parts of documentation can be directly done during knowledge acquisition. The explanation facility can use the informal models during the usage of the system which are also helpful for the maintenance of the system. Figure 1 shows some links relating the elicitation and the structure model or the structure model and the model of expertise.

5 Conclusion

MIKE is an approach integrating semiformal and formal specification techniques used during an incremental development process. The semiformal specification is not only used to simplify the formalization process but is also seen as an important result itself. It structures the complex problem solving process and with its informal descriptions of facts it can be used for documentation. The formal specification in the early specification phase describes precisely the functionality of the system without implementation details. Our formal specification is also operational representing a first prototype of the system. For developing our models and the relationships between models during the specification phase we provide a tool called MeMo-Kit (see [NeM93] and [Neu94]) providing a graphical editor environment. For evaluating the model of expertise an interpreter exists (see [Ang94]). Part of the current work of MIKE is the enhancement of the model of expertise to a design model which also considers non-functional requirements ([LaS94]). A comparison of MIKE work done in information system development and software engineering can be found in [AFL92a] and [FAL93].

Acknowledgements

We thank Rudi Studer who encouraged us in integrating semiformal and formal specification techniques.

3. How the model of expertise is described in the hypermedia environment as a network of nodes with formal contents and links is not described here. See [Neu94] for more details.

Bibliography

[AFL92a] J. Angele, D. Fensel, and D. Landes: Two Languages to Do the Same? In *Proceedings of the 2nd Workshop Informationssysteme und Künstliche Intelligenz*, February 24-26, 1992, Ulm, R. Studer (ed.), Informatik- Fachberichte, no 303, Springer

[AFL92b] J. Angele, D. Fensel und 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, July 1992.

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

[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.

[Ang94] J. Angele: Operationalisierung des Modells der Expertise mit KARL, Infix, St. Augustin, 1993 (in German).

[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"*, May 27-31, Avignon, 1991, pp. 513-528.

[FAL93] D. Fensel, J. Angele, D. Landes, and R. Studer: Giving Structured Analysis Techniques a Formal and Operational Semantics with KARL, *Proceedings of Requirements Engineering ´93 - Prototyping* -, Bonn, April 25 - 27, 1993, Teubner Verlag, Stuttgart, to appear 1993.

[FBA+93] 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.

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

[Fen94] D. Fensel: Graphical And Formal Knowledge Specification With KARL. In *Proceedings of the International Conference on Expert Systems for Development*, Bangkok, Thailand, March 29-31, 1994.

[HoN92] U. Hoppe and S. Neubert: Using Hypermedia for Integrating Mediating Representations in the Model-based Knowledge Engineering. In Proceedings of the AAAI'92 Workshop Knowledge Representation Aspects of Knowledge Acquisition, San José, California, July, pp. 55-62, 1992.

[KiL93] M. Kifer, G. Lausen, and J. Wu: Logical Foundations of Object-Oriented and Frame-Based Languages, technical report 93/06, Department of Computer Science, SUNY at Stony Brook, NY, April 1993. To appear in *Journal of the ACM*.

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

[LaS94] D. Landes and R. Studer: The Design Process in MIKE. In *Proceedings of the 8th Knowledge Acquisition for Knowledge-Based Systems Workshop KAW´94*, Banff, Canada, Jan. 30 - Feb. 5, 1994.

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

[Lin92] M. Linster (ed.): Sisyphus ´92: Models of Problem Solving, Arbeitspapiere der GMD, no 663, July 1992.

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

[NeM93] S. Neubert and F. Maurer: A Tool for Model Based Knowledge Engineering. In Proceedings of the 13th International Conference AI, Expert Systems, Natural Language (Avignon´93), 24-28 Mai, Avignon, 1993.

[NeO92] S. Neubert and A. Oberweis: Einsatzmöglichkeiten von Hypertext beim Software Engineering und Knowledge Engineering. In Proceedings Hypertext & Hypermedia '92, München, September 1992

[Neu93] S. Neubert: Model Construction in MIKE (Model Based and Incremental Knowledge Engineering) In: *Current Trends in Knowledge Acquisition - EKAW'93, 7th European Knowledge Acquisition Workshop*, Toulouse, France, September 6th - 10th, 1993, Lecture Notes in Artificial Intelligence, Springer Verlag, Berlin, Heidelberg

[Neu94] S. Neubert: Modellkonstruktion in MIKE (Modellbasiertes und Inkrementelles Knowledge Engineering) - Methoden und Werkzeuge. PhD thesis, Universität Karlsruhe, 1994 (in German).

[New82] A. Newell: The Knowledge Level, *Artificial Intelligence*, vol 18, 1982.

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

[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.

[Ull88] J. D. Ullman: *Principles of Database and Knowledge-Base Systems, vol I*, Computer Sciences Press, Rockville, Maryland, 1988.