In Proceedings of the International Conference on Expert Systems for Development, Bangkok, Thailand, March 29-31, 1994.

GRAPHICAL AND FORMAL KNOWLEDGE SPECIFICATION WITH KARL

Dieter Fensel
Institut AIFB, University of Karlsruhe,
76128 Karlsruhe, Germany
phone: 49-721-6084754, fax: 49-721-693717,
e-mail: fensel@aifb.uni-karlsruhe.de

Keywords: Expert system methodologies, knowledge acquisition, knowledge representation, formal specification languages, graphical specification languages

Abstract: The paper discusses an approach which allows the specification of a knowledge-based system (kbs) at several levels. The *Knowledge Acquisition and Representation Language KARL* combines a description of a kbs at the conceptual level supported by graphical modelling primitives with a description at a formal and executable level. Therefore, a KARL-specification can be used as a means for communication between expert and knowledge engineer as well as an intermediate representation, closing the conceptual gap between an informal specification and an implementation of a kbs. In the paper, KARL is mainly discussed as a *graphical modelling language*.

Introduction

There is a clear need for combining formal and executable specification of a kbs with descriptions at a high conceptual level supported by graphical representation primitives. In the paper, we discuss the Knowledge Acquisition and Representation Language KARL, which integrates formal specifications with a *graphical modelling language*. KARL uses a refined version of the KADS model of expertise [SWB93] as the conceptual model for a kbs description. These modelling primitives can be represented graphically. In addition, these primitives have a unique and precise semantics and can be used to define an executable prototype.[1] To some extent, KARL can be viewed as a high-level graphical programming language.

Section one of the paper introduces the main rationale of our approach. Section two introduces the description of a kbs at the conceptual level. Section three touches the description at the formal and executable level and section four discusses the graphical modelling primitives of KARL. A comparison with related work completes the paper.

1 Knowledge Level Modelling

Originally, expert systems or 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 [You89] or object-oriented analysis [CoY91] 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 [BHL90] 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 [Zav91] 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 (cf. section 5 of this paper) 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

1. „Graphical objects have to be more than nice pictures on the screen.“ [BFN91]

[FBA93]:

- 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.
- 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 *Knowledge Acquisition and Representation Language KARL* ([FAL91], [AFS94]) provides such an integrated specification approach for the domain of kbs. A kbs is described at three levels:

- *Conceptual level:* A refined KADS Model of Expertise is used to describe the kbs at a conceptual level. A specification is divided into several layers and a set of predefined modelling primitives is provided at each layer to describe the different knowledge types. Most of the primitives have a graphical representation which should support their understandability.
- *Formal level:* KARL combines perfect Herbrand model semantics of Horn logic [Prz88] with the modal semantics of dynamic logic [Koz90] to define a formal semantics for the specification of static and dynamic aspects of a kbs.
- *Operational level:* A fixpoint operator [Llo87] and a sequence operator are defined which compute the semantics of a KARL specification for a given input. Therefore, specifications in KARL can be evaluated by prototyping.

The development of the language KARL is part of the MIKE-project (Model-based and Incremental Knowledge Engineering) [AFL93], which aims at a developing method for kbs covering all steps from initial knowledge acquisition to design and implementation. Besides KARL, a hypertext-based tool has been implemented which allows to structure verbal protocols in a so-called semiformal Hyper model [NeM93]. This semiformal model can be used to built up a formal specification with KARL by refining the informal specification. An interpreter for KARL has been implemented which includes a debugger. Currently, work is done to extent KARL to cover design aspects [LaS94].

2 The Conceptual Level

We first introduce the KADS model of expertise before we show some refinements of it.

2.1 The KADS Model of Expertise

A very important part of the KADS methodology [SWB93] is the *model of expertise* which describes the different kinds of knowledge required to solve the given tasks. The model of expertise distinguishes different types of knowledge, defines primitives to express them, and organizes them into several layers. Precisely it distinguishes static knowledge and three types of control knowledge.[1] The goal of a model of expertise is to provide a model of the problem solving behaviour independently of a certain implementation.

Domain layer: It represents knowledge about the application domain of the system. An important property of the domain layer is that the knowledge should be represented as independently as possible from the way it will be used. It has two main purposes. First, it should define a conceptualization of the domain. Second, it should define a declarative theory of the domain providing all the domain knowledge required to solve the given tasks.

Inference layer: This second layer defines the first type of control knowledge. It specifies the inferences that constitute a problem-solving method and specifies how to *use* the knowledge from the domain layer in these inferences. This is done in two ways: the inference layer specifies

- the *inference steps* that can be made using the domain knowledge, and
- the *knowledge roles*, which model the premises and conclusions of the inferences.

The inference steps are assumed to be elementary in the sense that they are completely described by their names, an input/output specification and a reference to the domain knowledge that they use. The inference layer specifies the inference steps and knowledge roles as well as the data-dependencies between these steps and roles. These dependencies are specified in a network of inference actions and knowledge roles known as an *inference structure*. The inference layer *restricts* the use of the domain layer knowledge and *abstracts* from it. It restricts all possible inferences to the set of inferences which are defined by it. This is done to improve the efficiency of the problem-solving process. The inference layer abstracts from the domain layer by using task-specific names for inferences and roles. The domain-independent formulation of the inference layer should support its reuse, i.e. its application for similar tasks in different application domains. A *domain view* must specify the relationship between the generic terms used at the inference layer and the domain-specific knowledge specified at the domain layer. Mainly, roles have to be connected with domain concepts and inference actions have to be connected with knowledge required for them.

Task layer: A task represents a fixed strategy for achieving problem solving goals. The purpose of the task layer is to specify *control* over the execution of the basic inference steps specified at the inference layer. This is done by imposing an ordering on these steps in terms of execution sequences, iterations, conditional statements etc. The description of a task consists of three components: The goal which is fulfilled by the task; the control terms which correspond to knowledge role of the inference layer and which are used to specify conditions for the control flow; and the task structure which

1. Because there is still significant disagreement about the third type of control knowledge (i.e., the *strategic layer*) we have neither regarded it for the KARL model nor will we further discuss it in this paper.

hierarchically refines a given task to subtasks and elementary steps, i.e. inference actions.

2.2 The Refined Model of Expertise of KARL

Originally, KADS proposed KL-ONE as language for the domain layer. KL-ONE defines a very restricted set of language primitives which enables strong characterisations of decidability and efficiency of reasoning with it. Yet, for a specification language a broad syntactical variety of modelling primitives seems necessary to make the step from an informal to a formal description as smooth as possible. Therefore, KARL integrates concepts of object-oriented databases and logic for the domain layer, the inference layer, and their connections. KARL provides the sublanguage *Logical-KARL (L-KARL)* for this purpose. L-KARL is a derivate of Frame-logic (F-logic) [KLW93]. L-KARL distinguishes classes, objects, and values. L-KARL provides classes and an is-a hierarchy with multiple attribute inheritance to describe terminological knowledge. Attributes can be single-valued or set-valued, i.e. the value can be a set. Attributes can be used to describe objects as well as classes. Attributes have defined domain and range types.

Intensional and factual knowledge is described by logical relationships between classes, objects, and values. Objects are grouped into classes via an is-element-of relationship. The logical language to describe relationships between classes, objects, and values is Horn logic with equality and function symbols extended by (stratified) negation.

A domain layer is structured and hierarchically ordered by the is-a hierarchy between classes and a module hierarchy.

Besides its use at the domain layer, L-KARL is used to specify the logical relationship defined by an inference action at the inference layer. Extending KADS, L-KARL can be used to define a terminological structure of a knowledge role. In KADS, such roles are flat containers, whereas in KARL they can be used to define a *task-specific terminology* independently from the domain-specific terminology. The need for such a task-specific terminology is one of the most significant results of the role-limiting method approach ([Mar88], [Pup93]).

A second improvement compared to KADS at the inference layer is the introduction of *hierarchical refinement* similar to levelled dataflow diagrams [You89]. Therefore, in KARL large specifications are possible.

Furthermore, L-KARL is used to specify the mapping between domain layer and inference layer. Modified Horn logic can be used to define a view from the problem-solving method on the domain knowledge.

The sublanguage *Procedural-KARL (P-KARL)* is used to specify the control flow of a problem-solving method at the task layer. Sequence, branch, loops, and procedure calls are the means to specify control flow and the hierarchical task structure. Conditions can be specified via logical statements about the contents of knowledge roles. The goal of a task is only described informally.

3 The Formal and Executable Level

The logical language L-KARL used to describe the domain layer, the inference layer, and their connection has a Herbrand model semantics [Llo87]. KARL allows stratified negation under the closed-world assumption using the minimal (i.e., perfect) Herbrand model as semantics [Prz88]. Constraints check this model for correctness.

The procedural knowledge is represented by P-KARL. It is a variant of Dynamic Logic which has a modal semantics [Koz90]. The integration of the modal semantics of the task-layer and the Herbrand models of L-KARL is as follows: the models of L-KARL are used to define an interpretation for a P-KARL language, i.e. the perfect Herbrand model of a set of clauses is used to interpret a function symbol occurring in value assignments in P-KARL.

An operational semantics for KARL is defined in [Ang93]. In contrast to Prolog, the evaluation of these clauses is set-oriented [Ull88]: not one but all instantiations of a predicate are computed. This semantics was used to implement an interpreter and debugger in C which supports knowledge evaluation by prototyping. The main restriction for executable specifications is that the perfect Herbrand models have to be finite.

For more details see [Fen93] and [Ang93].

4 The Graphical Modelling Primitives of KARL

KARL provides graphical representations of most modelling primitives to improve their understandability. Every graphical symbol has a defined meaning given by the formal semantics of KARL. In particular, the following graphical representation formalisms are used:

- A variant of *Enhanced-Entity-Relationship (EER) diagrams* [ElN89] for the domain layer,
- a variant of *levelled dataflow diagrams* [You89] for the inference layer, and
- a variant of *programflow diagrams* [Din66001] for the task layer.

Specifications of real-world systems become unintelligible because of their complexity. Therefore all three graphical representations include *hierarchical refinement* to allow to represent the system on different levels of refinement.

4.1 The Three Layers

Figure 1 shows a model of expertise of a solution of the so-called *Sisyphus problem*. Sisyphus is a project that aims at comparing different approaches to aspects of knowledge engineering [Lin92]. An assignment problem is posed, in which employees are assigned to office places with several requirements to be met. Extracts of this example are used to illustrate the graphical primitives of KARL.

The domain terminology and the domain knowledge required by the problem-solving method is defined at

the domain layer. The inference layer contains the

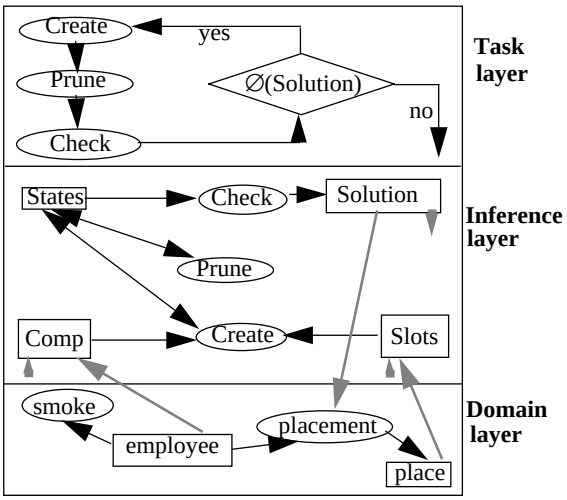

Fig. 1. An example for a model of expertise.

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.

4.2 The Domain Layer

Enhanced-Entity-Relationship (EER) diagrams [ElN89] are a well-known technique to represent static knowledge graphically. A similar representation is used here for the domain layer of KARL. Classes, domain and range restrictions of

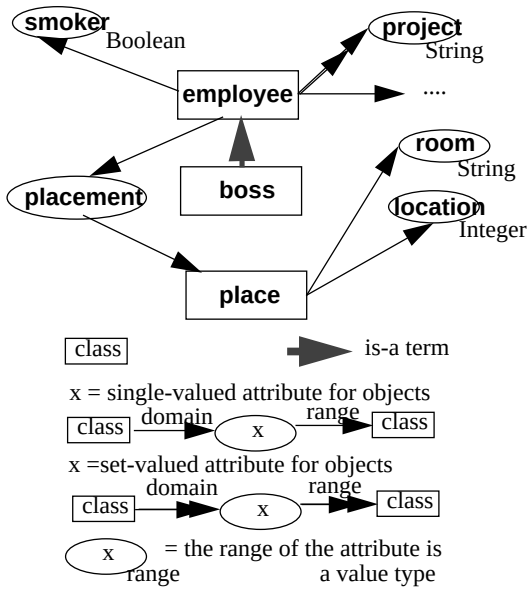

Fig. 2. Graphical representation at the domain layer.

attributes, and is-a terms are represented graphically. In our example a class of employees is which are described by their projects. It also indicate if the employees smoke or not. An additional class *boss* contains employees who have to be handled in a specific way. Furthermore, there is a list of rooms with a description of where they are situated.Figure 2 shows the graphical representation. Ground facts are represented by tables. Rules and constraints are not represented graphically. This would, however, be possible by applying the ideas given in [Cru92] that allow the visualization of literals and the introduction of user-defined icons.

class: boss ≤ {employee}

Object-ID	smoker	project
wl	FALSE	EULISP, RESPECT
...	...	...

4.3 The Inference Layer

The terminological knowledge which is defined in inference actions and knowledge roles can be represented by the same graphical primitives as introduced at the domain layer.

As, in the interpretation of KARL, the inference layer is similar to a dataflow diagram [You89], a similar kind of formalism is used for it. In difference to KADS, KARL distinguish three types of roles:

- Views ( ) which can be used to read knowledge from the domain layer.
- Terminators ( ) which can be used to write results of a problem-solving method at the domain layer.
- Stores ( ) which model the dataflow between inference actions.

Similarly to dataflow diagrams and in extension of the original proposal of KADS, the inference layer is extended by hierarchical refinement. The rules that apply for a consistent refinement are comparable to those applied to dataflow diagrams or Petri nets (see [Fen93] for more details). Additional properties of inference actions which can be represented graphically concern the following issues:

- If an inference action has several input roles, does it work if at least one input roles is not empty, i.e. not filled with {∅}, or does it work only if all input roles are not empty?
- If an inference action has several output roles does it always fill all of them or only some?

An inference action *merge* which should compute tuples of elements of two roles, only works if each of the two roles contains at least one element. An inference action like *join*, however, which should join

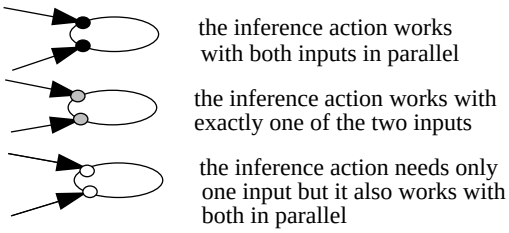

Fig. 3. Interface modelling of inference actions.

all elements of two roles into one set could also work

if only one role contains some elements. [Kun89] introduced modelling primitives answering these questions in the domain of dataflow diagrams. These modelling primitives permit to express whether the inputs of an inference action are connected by *and*, *or*, or *exclusive or* (see figuer 3). The same can be expressed for the outputs. This notation can be nested to connect several inputs or outputs.

4.4 The Task Layer

There are several alternatives for the graphical representation of the procedural knowledge at the task layer. Two well-known means are Nassi-Shneiderman diagrams [NaS73] and Program-flow diagrams [Din66001]. Nassi-Shneiderman diagrams enforce structured programming and are well-suited for hierarchical refinement. A graphical knowledge acquisition tool for them, however, is quite complicated to realize, and the revision of modelled knowledge is quite difficult for the knowledge engineer. Therefore, because of the cyclic manner of knowledge acquisition and the high amount of revision, a variant of program flow diagrams for the task layer is used here (see figure 1).

5 Related Work

The development of formal and informal specification techniques has a long tradition in software engineering. *Structured analysis* [You89] is a method which contains several informal instruments for software specifications like data dictionary, dataflow diagrams, and state-transition diagrams. The description techniques of structured analysis define powerful means to describe a system at a high conceptual level in an easily understandable way. There exist several approaches to formalize or operationalize these informal descriptions. Teamwork/ES [BlH88] is a prototype system that executes a real-time structured-analysis specification in an interactive and graphical manner. Other approaches recommend a combination with algebraic specification techniques [FrD89] or with functional programming languages [EMS89]. The Prototyping System Description Language PSDL [KrR89] is used together with modified dataflow diagrams as a graphical tool for rapid prototyping of system specifications. For a comparison of structured analysis techniques and KARL see [FAL93]. It is shown how KARL can be used to formalize and operationalize the informal techniques of structured analysis.

Petri nets are broadly used as a graphical and formal specification techniques for information system development [Mar93]. INCOME [LNO89] is a method for the development of information systems providing an executable specification language based on semantical data models similar to SHM and high-level Petri nets (i.e., predicate-transition nets). This proposal is similar in spirit to [Stu87] and [Kun89]. Petri nets are also used in software engineering to specify distributed systems. SEGRAS [Kra87] is a formal and semigraphical language combining Petri nets and abstract data types. SPECS [DGG87] uses high-level Petri nets combined with object-oriented extensions. For a comparison of INCOME and KARL see [AFL92]. The main difference between INCOME and KARL is the clear separation of dataflow and controlflow and the domain independent description of these two components in KARL.

The model of expertise of KADS defines an informal description of a kbs at a high conceptual level. Several language developers have used this or similar models as a starting point for defining a formal or executable specification language for kbs. Examples are DESIRE, FORKADS, $K_{BS}SF$, $(ML)^2$, Model, MODEL-K, MoMo, OMOS, and QIL. A comparison of KARL with most of these languages can be found in [FeH93] and [FeS92]. The main features of KARL compared with these languages are:

- KARL is the only language which has a complete declarative and operational semantics, i.e. aims successfully at formalization and operationalization of a specification.
- KARL tries to integrate results of software engineering and information system development into the knowledge engineering process. Therefore, KARL has very powerful means to model static knowledge and integrates prototyping as a means for the evaluation of specifications.

Meanwhile, KARL has been applied in a dozen of case studies and several tools supporting the specification process with KARL have been developed (see [Fen93]).

Acknowledgements

I thank Claudia Böttcher and Dieter Landes for very helpful comments on drafts of the paper.

References

[AFL92] 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-Verlag, Berlin, 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.), *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.

[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).

[BHL90] D. Bjørner, C. A. R. Hoare, and H. Langmaack (eds.): *VDM´90. VDM and Z - Formal Methods in Software Development*, Lecture Notes in Computer Science, no 428, Springer-Verlag, Berlin, 1990.

[BlH88] R. Blumofe and A. Hecht: Executing Real-Time Structured Analysis Specifications, *ACM SIGSOFT Software Engineering Notes*, vol 13, no 3, July 1988, pp. 32-39.

[BFN91] H.-D. Böcker, G. Fischer, and H. Nieper-Lemke: The Role of Visual Representations in

Understanding Software. In D. Patridge (ed.), *Artificial Intelligence & Software Engineering*, Ablex Publishing, Norwood, New Jersey, 1991.

[CoY91] P. Coad and E. Yourdon: *Object-Oriented Analysis*, 2nd ed., Yourdon Press, Englewood Cliffs, 1991.

[Cru92] I. F. Cruz: DOODLE: A Visual Language for Object-Oriented Databases. In *Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data*, San Diego, California, June 2-5, 1992.

[DGG87] J. Dähler, P. Gerber, H.-P. Gisiger, and A. Kündig: A Graphical Tool for Design and Prototyping of Distributed Systems, *ACM SIGSOFT Software Engineering Notes*, vol 12, no 3, 1987.

[DIN66001]German Industry Norm (DIN), no 66001.

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

[EMS89] S. Eisenbach, L. McLoughlin, and C. Sadler: Data-Flow Design as a Visual Programming Language. In *Proceedings of the 5th International Workshop on Software Specification and Design*, May 19-20, Pittsburg, Pennsylvania, 1989.

[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*, May 27-31, Avignon, 1991.

[FAL93] D. Fensel, J. Angele, D. Landes, and R. Studer: Giving Structured Analysis Techniques a Formal and Operational Semantics with KARL. In H. Züllighoven et al. (eds.), *Requirements Engineering ´93: Prototyping, German Chapter of the ACM Berichte*, no 41, Teubner Verlag, Stuttgart, 1993.

[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.

[FeH93] D. Fensel and F. van Harmelen: A Comparison of Languages which Operationalize and Formalize KADS Models of Expertise. Research report, University of KArlsruhe, no 280, 1993.

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

[FeS92] D. Fensel and R. Studer: An Analysis of Languages which Operationalize and Formalize KADS Models of Expertise. In *Proceedings of the 7th Knowledge Acquisition for Knowledge-Based Systems Workshop KAW´92*, Banff, Canada, October 11-16, 1992.

[FrD89] R. B. France and T. W. G. Docker: Formal Specifications Using Structured System Analysis. In *Proceedings of the 2nd European Software Engineering Conference ESEC´89*, Warwick, September 11-15, Lecture Notes in Computer Science, no 387, Springer-Verlag, Berlin, 1989.

[KLW93] 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.

[Kra87] B. Krämer: SEGRAS - A Formal and Semigraphical Language Combining Petri Nets and Abstract Data Types for the Specification of Distributed Systems. In *Proceedings of the 9th International Conference on Software Engineering*, March 30th - April 2th, Monterey, CA, 1987.

[KrR89] R. Krista and I. Rozman: A Computer Aided Prototyping Methodology, *ACM SIGSOFT Software Engineering Notes*, no 6, vol 14, October 1989.

[Kun89] C. H. Kung: Conceptual Modeling in the Context of Software Development. In *IEEE Transaction on Software Engineering*, vol 15, no 10, 1989.

[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, January 30 - February 5, 1994.

[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.

[LNO89] G. Lausen, T. Nemeth, A. Oberweis, F. Schönthaler, and W. Stucky: The INCOME Approach for Conceptual Modelling and Prototyping of Information Systems. In *Proceedings of the 1st Nordic Conference on Advanced Systems Engineering CASE'89*, Stockholm, Sweden, May 9-11, 1989.

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

[Mar93] M. A. Marsan (ed.): *Application and Theory of Petri Nets 1993, Proceedings of the 14th International Conference*, Chicago, June 21-25, Lecture Notes on Computer Science, no 691, Springer-Verlag, Berlin, 1993.

[NaS73] I. Nassi and B. Shneiderman: Flowchart Techniques for Structured Programming, *ACM SIGPLAN Notices*, vol 8, no 8, 1973.

[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)*, Mai 24-28, Avignon, 1993.

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

[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.

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

[Stu87] R. Studer: *Konzepte für eine verteilte wissensbasierte Softwareproduktionsumgebung* (Concepts for Distributed Knowledge-Based Software Development Environments). In Informatik-Fachberichte, no 132, Springer-Verlag, Berlin, 1987 (in German).

[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.

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

[Zav91] P. Zave: An Insider´s Evaluation of PAISLey. In *IEEE Transactions on Software Engineering*, vol 17, no 3, 1991.