Integrating Semiformal and Formal Methods in Knowledge-Based Systems Development

Dieter Fensel, Dieter Landes, Susanne Neubert and Rudi Studer

Institut AIFB, University of Karlsruhe, 76128 Karlsruhe, Germany
e-mail: {fensel | landes | neubert | studer}@aifb.uni-karlsruhe.de

Abstract

The paper describes a development approach for knowledge-based systems (KBS) combining semiformal and formal techniques for specification and design. For the semiformal representation we use a hypermedia-based formalism which serves as a communication basis between expert and knowledge engineer during knowledge acquisition. The semiformal knowledge representation is also the basis for formalization, resulting in a formal and executable model of expertise specified in the Knowledge Acquisition and Representation Language (KARL). In the design phase, this representation is enriched by focussing on issues of realization and facilitating the mapping of the model of expertise to the implementation environment. A smooth transition from a semiformal to a formal specification and further on to design is achieved as all the description techniques rely on the same conceptual model to describe the system. Thus, the system is thoroughly documented at different description levels, each of which focuses on a distinct aspect of the entire development effort. Traceability of requirements is supported by linking the different models to each other.

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 to guide the further knowledge acquisition process. The distinction of symbol level and knowledge level [New82] created the conceptual framework for a different kind of process models for KBS development. A knowledge level description of the *task* solved by the system and the *knowledge*, which is required to solve the task, is constructed by means of a modelling activity. This knowledge level description is built independently of the design and implementation activity. The separation of analysis and design/implementation is an analogue to experiences 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 on the one hand and design and implementation on 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 a high and informal level of description. These approaches broadly use graphical means like entity-relationship diagrams, dataflow diagrams, flow charts, and state-transition diagrams. This type of specification is 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.
- *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 arrive at realistic descriptions of the desired functionality of the systems.

Several authors argue to combine these description techniques in order to overcome their individual shortcomings when used stand-alone. Informal specifications are prone to 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 intuitive understanding of the functionality the system must provide, given only the huge amount of details of a formal specification. The need for combination is obvious if one considers the two different purposes of specifications [FBA+93]:

- A specification 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 on the one hand and the knowledge engineer on the other hand.
- A specification should serve as an *intermediate representation* closing the gap between an intuition about the functionality of a system and its actual design and implementation.

Additionally, intermediate representations document modelling decisions made during the various phases of the life-cycle [PoB88]. Thus, they are helpful with respect to requirements traceability, i.e. the problem of making sure that requirements that are posed are actually implemented in the final system as well as clarifying which parts of the implementation are responsible for a particular requirement.

The integrated development on the basis of semiformal and formal 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. MIKE proposes the integration of *life cycle models*, *prototyping*, *semiformal*, and *formal specification techniques* into a coherent framework.[1]

This paper presents the formalisms used in MIKE, namely the semiformal hypermedia-based formalism *MEMO* [Neu93], the formal and executable *Knowledge Acquisition and Representation Language KARL* ([FAL91], [AFS94]) and its extension, *DesignKARL* [Lan94a], which serves as the formalism to express the system design. Moreover, we describe how the models that are described with these languages are interrelated, which is important for documentation and maintenance purposes.

The paper is organized as follows. In section two, the formalism to represent the semiformal models is described. Then, the formal specification language KARL and the model of expertise are discussed in section three. Section four focuses on the design model and DesignKARL as the formalism for its description. Section five addresses the linkage of the various models that are developed during knowledge acquisition and design. Section six gives a short overview of some of the tools that support the knowledge acquisition process in MIKE. Finally, related work is described and a conclusion is given.

---

1. One of the anonymous reviewers of our paper argued that our paper could also submitted to a software engineering conference. This is true and we are happy about this!

2 The Semiformal Models of a Knowledge-Based System

Developing a formal specification directly from informal knowledge protocols is rather difficult. Therefore, mediating representations are constructed in MIKE before starting the formalization process ([HoN92], [Neu93]). Our mediating representations enable to describe protocols, concepts and activities, data flow and control flow of activities etc. with multimedia facilities.

The development of mediating representations provides different advantages: *Semiformal* representations can be used as a communication level between the knowledge engineer and the expert. The expert can be integrated in the knowledge engineering process of structuring the complex knowledge such that the knowledge engineer is able to interpret and formalize it more easily. Thus, the cooperation between expert and knowledge engineer is improved and the formalization process is simplified. An early evaluation process is possible in which the expert himself is integrated. In addition, a mediating representation is a basis for documentation and explanation. The maintenance of the system is also simplified.

For our mediating representations we propose a semiformal, hypermedia-based formalism called *MEMO* (MEdiating Model Organization) [Neu93], [NeO92]. This formalism enables to describe two semiformal models (the *elicitation model* and the *structure model*) which are defined as sets of special node and link types grouped into so-called contexts. A *node* is a hypermedia document with a content using text, graphics, audio or video to describe the meaning of the node. 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* establish a specific view on a set of nodes and links. A *model* is defined as a set of nodes, links, and contexts.

The first model, the *elicitation model*, documents the elicitation process. Thus, it includes knowledge protocols which are stored in so-called *protocol nodes*. Additionally, *date links* between protocol nodes are included to describe the elicitation ordering. In Figure 2 at the top a short example of two related protocol nodes is sketched for the *Sisyphus office assignment problem*[2], which is concerned with assigning employees to office places in such a way that several requirements will be met (see [Lin92]).

The *structure model*, which is developed based on the elicitation model gives a more structured description of the knowledge. It contains the following description elements (an example is described in Figure 1):

- The *activity context* includes all *activity nodes* each describing a step of the problem-solving process. Additionally, *refinement links* are integrated. This context enables a view on the complete activity hierarchy. 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. Looking at the example of figure 1, the global Sisyphus activity node 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). Each activity node is informally described in the node content.
- An *ordering context* provides a view on *activity nodes* which are related by *ordering links*. These activity nodes lie on *one* hierarchy level (i.e. include activity nodes with the same mother node). 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. In figure 1 the ordering context describes a cycle of the three activities.

2. The Sisyphus problem was used to compare different knowledge engineering approaches.

- 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. In figure 1, one is-a link and two part_of links are shown, that is, it is modelled that an *empl* and a *place* are part of a *pair.*
- 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. Figure 1 shows in the structure context section that *empl* and *place* are input for the activity *create pairs*.

The structure model together with its nodes and links is produced on the basis of the node contents of the elicitation model. The structuring process can be mainly done by the expert itself, supported by the knowledge engineer. Then, the structure model is an adequate basis for the development of the formal models done by the knowledge engineer. These are described in the next section.

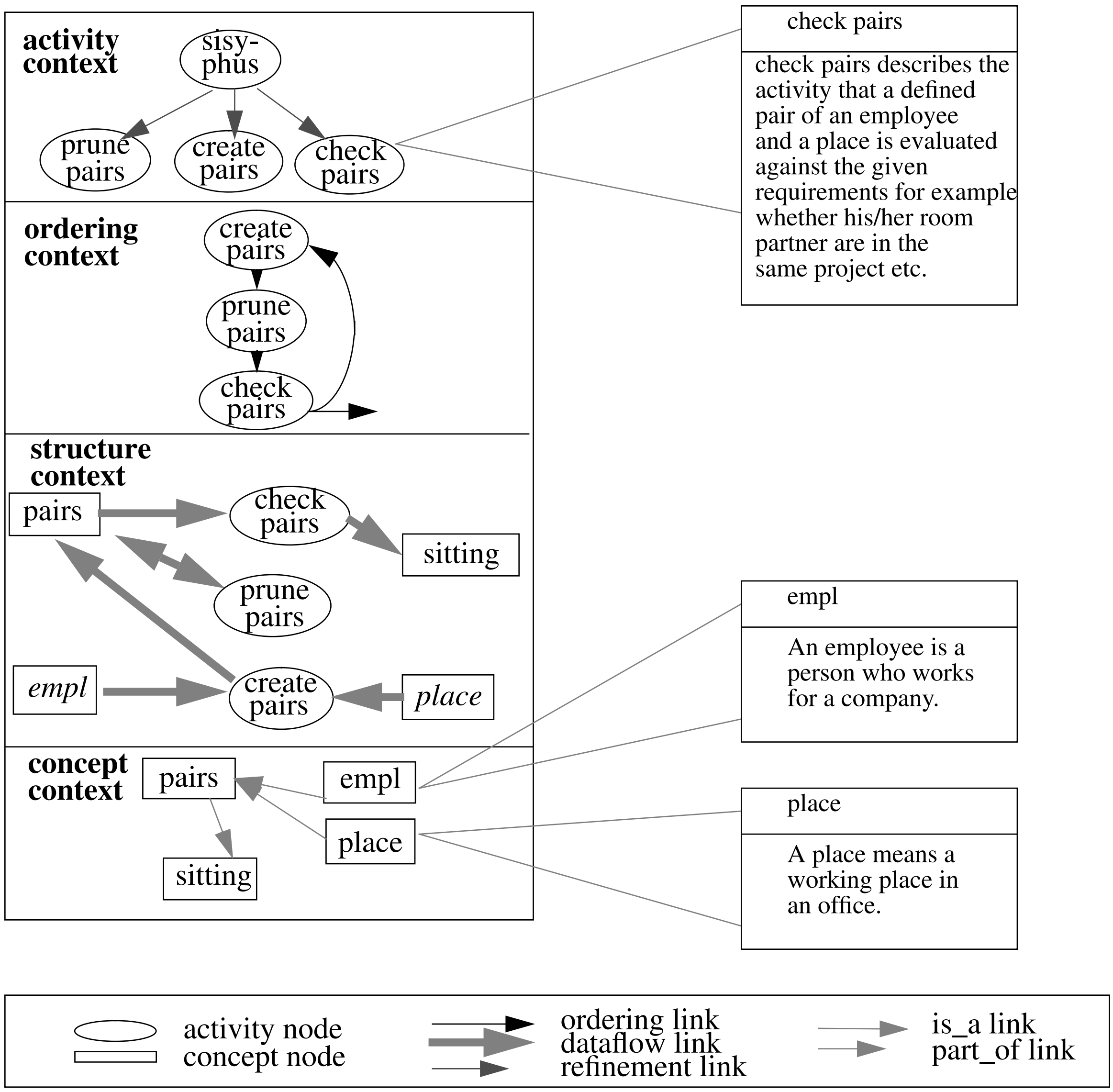

Figure 1 Example of a partial structure model

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 conceptualisation 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 supply domain knowledge to an inference action are called *views*, roles which model the data flow 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 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 *problem-solving-method-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 control flow 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.

Figure 2 shows a model of expertise of a solution 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. 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)

A complete description of KARL can be found in [Fen93]. A short description of the modelling primitives of KARL is given in [AFS94] and [FAS94]. Details on some of the applications of KARL can be found in [AFL92b], [LFA93], and [PFL+94]. A comparison of KARL with other languages can be found in [FeH94]. The main characteristics of KARL is the combination of a *conceptual description* of a knowledge-based system together with its *formal definitions* and its evaluation by *executing* the specification.)

Logical-KARL (L-KARL

L-KARL is a customization of Frame-logic (F-logic) [KLW93]. F-logic and L-KARL extend the modelling primitives of first-order logic by syntactic modifications but preserve its model-

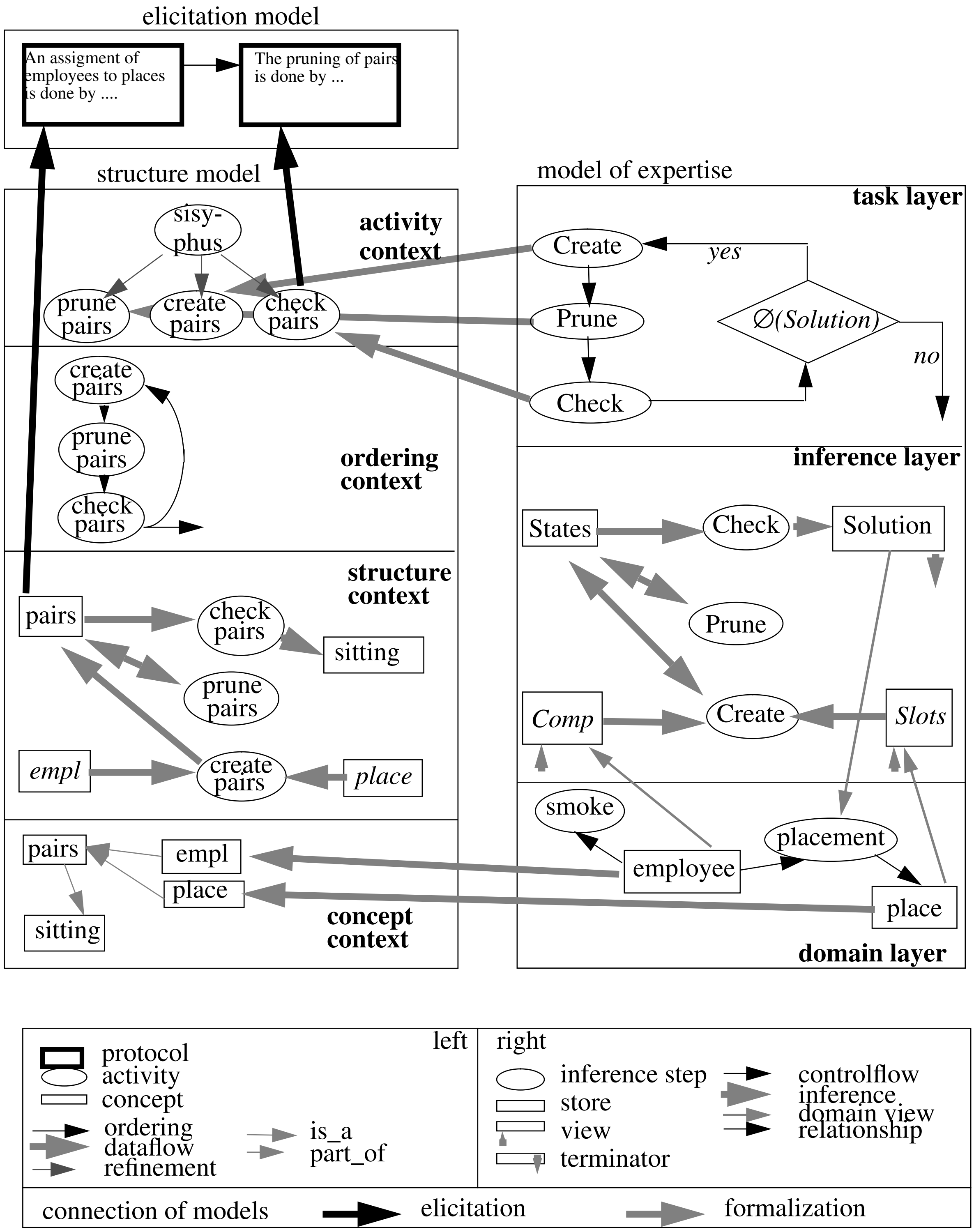

Figure 2 Parts of an elicitation model and a structure model at the left (nodes have informal content) and of a model of expertise at the right (nodes with formal content).

theoretical semantics. 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 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 roles. 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 *formal 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 optimised evaluation strategy were developed which establish the foundation of an interpreter and debugger for KARL [Ang93]. The *operationalisation* consists of two parts. The logical description of an inference action together with the domain layer and view definitions using L-KARL define a relationship. For given input, this logical descriptions are evaluated by a fixpoint operator which computes based on the perfect Herbrand model semantics the valid output. A sequence operator is used to interpret the control flow between inference action as it is defined in P-KARL. By this it becomes possible to evaluate a formal specification by executing it.

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 *program flow diagrams* describes the task layer. All three graphical representations include *hierarchical refinement* to allow to represent the system on different levels of refinement. Figure 2 shows the graphical representation a model of expertise of a solution of the so-called *Sisyphus problem*.

4 The Design Model

4.1 The Scope of the Design Model

The structure model and the model of expertise aim at a description of the knowledge-based system at a conceptual level. Consequently, they specify what functionality the KBS must provide to solve the given problem, but do not address issues which are related to the realization of this functionality. Thus, the gap between the model of expertise and the final implementation of the KBS is still fairly wide. In order to bridge this gap, a distinct design phase is required to facilitate the transition from knowledge acquisition to implementation. The design phase results in an additional model, the *design model*, which is concerned with mapping of the knowledge contained in the model of expertise into potential target environments, but which is still at a level of abstraction above the actual implementation.

During the design phase, particular emphasis has to be put on non-functional requirements such as, e.g., (symbol-level) efficiency, maintainability, understandability, reliability etc. as well as the constraints imposed by the intended software and hardware environment for the final implementation of the KBS [Lan94b]. Since the model of expertise already constitutes a functional specification of the required behaviour, only limited attention has to be devoted to functional requirements in the design phase. The goal of the design phase thus consists in enriching the knowledge which is already contained in the model of expertise by additional information which indicates how the realization of the KBS should be accomplished. The design model finally comprises the specification of how both, functional and non-functional, requirements can be met.

Two aspects are particularly important for the satisfaction of non-functional requirements. On the one hand, suitable data structures and algorithms have to be introduced in the design phase in order to realize knowledge and inference steps using that knowledge in an appropriate fashion. On the other hand, the structure of the system must be defined properly, e.g., by decomposing the system into a collection of smaller units such as modules which interact during problem-solving.

Design decisions which refine parts of the model of expertise with respect to these two aspects are constrained by the goal to preserve the structure of the model of expertise during design [Sch93]. Besides facilitating the transition between the formalisms used during knowledge acquisition and design, this conveys various benefits with respect to maintainability and explainability. In particular, structure-preserving design supports the traceability of both functional and non-functional requirements.

4.2 DesignKARL

In order to be able to describe those issues which are particularly addressed in the design phase, KARL as the description formalism for the model of expertise has to be extended by additional language primitives. This extension, *DesignKARL*, firstly allows to use more sophisticated data structures. KARL supplies only a very restricted collection of data types (namely, object classes and individual objects, predicates, sets and basic value types such as integer numbers, strings etc.), whereas DesignKARL provides data types which allow to exploit inherent relationships between data items (by means of data types such as, e.g., sequences, stacks, queues, trees) as well as to efficiently retrieve data from large data sets (by means of, e.g., index structures and hash tables). Each of these additional data types is associated with a predefined collection of operations which may be performed for the manipulation of data items of the respective type. Furthermore, DesignKARL allows to associate user-defined operations to object classes and predicates.

DesignKARL allows to refer to such data structures in enriched inference actions and algorithms to refine inference actions given in the model of expertise. Algorithms often express additional control knowledge which is irrelevant for a single inference step from a conceptual point of view, but which is important for its efficient realization.

During design, the system will usually be decomposed into smaller and largely autonomous units in order to reduce the overall complexity. To that end, DesignKARL supplies two additional structuring primitives, namely *domain modules* and *processing modules* [LaS94b]. Domain modules allow to collect related domain knowledge in a single place, while processing modules correspond to composed inference actions and their corresponding control specifications. Both types of modules communicate via interfaces which allow to hide irrelevant details to the outside.

In addition to these language primitives which aim to express the results of design decisions, DesignKARL also encompasses several language primitives for describing what the *design decisions* actually are, e.g. which data structures were introduced in a particular development step, and why these decisions were made. With these language primitives, sections in the design model can be linked to parts of the model of expertise from which they are derived through design decisions. Furthermore, design decisions are linked to those requirements that constitute the motivation for them being applied, thus providing valuable support for the traceability of requirements.

A more detailed account of DesignKARL can be found in [Lan94a].

5 Model Development and Connection

One central point of our approach is the connection of models developed during the knowledge engineering process. This model connection conveys various benefits: first, the informal information integrated in the elicitation or structure model serves as a documentation of a formal description. Moreover, 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. These models are also helpful for the maintenance of the system.

In this section we will describe the knowledge engineering *process* in which the models of MIKE in the different formalisms are developed and related (section 5.1). Moreover, we will describe the *product* of knowledge engineering, the entire model environment (section 5.2).

5.1 The Process of Model Development and Connection

The phases and subphases of KBS development according to the MIKE approach are shown in Figure 3. These steps are performed in a cyclic fashion [AFL+93] guided by a *spiral model* [Boe88] as process model. It is possible to switch between different activities like interpretation or model connection and there is no kind of sequence enforced as in traditional waterfall models. The steps which are of particular interest in the context of this paper are printed in *italics* in Figure 3.

After the task analysis which will not be treated here, the knowledge acquisition process starts with *elicitation*, i.e. trying to get hold of the experts' knowledge. During the interview, the knowledge engineer may use a suitable predefined problem-solving method from the library as a guideline[3]. The resulting knowledge protocols are stored in protocol nodes of the elicitation model.

Structures described in the knowledge protocols are modelled by contexts of the structure model. The semiformal structure model is a first result of *interpretation* which clarifies complex knowledge structures.

Moreover, the structure model is the foundation for the *formalization/operationalization* process which results in the model of expertise described in KARL.

During *model connection,* the models are related by special links. 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 to the information which originally was provided by the expert is established. A so-called *elicitation link* exists to describe these interrelation. During structuring, these links can be easily introduced. Second, the structure model is related with the model of expertise (see section 3.1). So-called *formalization links* relate a formally described node of the model of expertise with an informally described node of the structure model. Corresponding nodes, including an informal description on the one hand and a formalization on the other hand, are linked. It should be clear that most of these links can be automatically generated during the model construction process.

Model evaluation is concerned with evaluating the operational model of expertise with respect to functional requirements by means of test cases.

The *design* phase [LaS94a] is performed on the basis of the model of expertise after it has been evaluated with respect to the required functionality. Design decisions that are made in the model construction step are primarily motivated by non-functional requirements and the constraints imposed by potential software and hardware target environments. Instead of describing only the results of design decisions, model construction in the design phase also comprises establishing a link between the model states before and after the application of design decisions. Thus, parts of the model of expertise are (transitively) linked to corresponding sections of the design model, which in turn may be linked to corresponding parts of the final implementation. In combination with the goal to preserve the structure of the model of expertise during design, the links between model of expertise, design model and implementation ensure traceability of

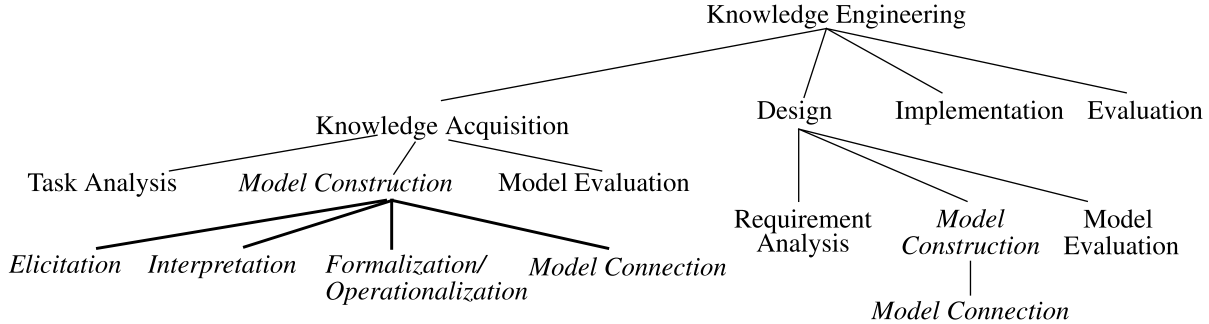

Figure 3 Phases in the MIKE life-cycle.

3. This principle is not described here.

functional requirements. Traceability with respect to non-functional requirements is established by links between requirements and the design decisions taken on their behalf during model construction.

5.2 The Modelling Result

The result of the different phases described in section 5.1 is a complex model environment including several interrelated models. Figure 2 shows a small sections of the model environment developed for the Sisyphus example during knowledge acquisition. Links relating the elicitation model with the structure model and the structure model with the model of expertise are shown. Concept nodes (e.g. *pairs*) and activity nodes (e.g. *check pairs*) are related with protocol nodes, in which they have been described by the expert, by so-called elicitation links. Between formally described knowledge elements (nodes of the model of expertise) and nodes of the structure model so-called formalization links are set. For example the knowledge element *employee* of the domain layer, which includes a formal specification in KARL, is related with the knowledge element *empl* of the concept context, where an informal explanation is given. In the same sense inference steps are related with activity nodes of the structure model. Design decisions may, e.g., focus on collecting inference knowledge and the corresponding control knowledge in a processing module or related domain items in a domain module. Furthermore, roles may be associated with appropriate data structures. Introducing data structures may imply an adaptation of the inference steps using the data structures. Inference steps can also be refined into algorithms by introducing additional control knowledge which is not required from a conceptual point of view.

6 Tool Support

MeMo-Kit[4] (Mediating Model Construction Kit) (cf. [NeM93] and [Neu94]) supports the graphical construction of the elicitation model, the structure model and the formal model of expertise by different editors. Furthermore, an editor for developing a library of predefined problem-solving methods (PSMs) (not described here) exists. One component helps selecting and adapting these PSMs. After the elicitation (which is not supported by MeMo-Kit), the knowledge engineer enters the knowledge protocols into the elicitation model using the elicitation model editor. Since MeMo-Kit is a hypermedia based tool, it is also possible to store tapes, graphics, etc. The protocol nodes are the basis for constructing the structure model using the structure model editor. Formalization, i.e. constructing the model of expertise, is done in the KARL editor. The editors cooperate with each other and links between models are constructed partly automatically by the editors.

For executing the operational specification, the model of expertise, an interpreter[5] for KARL exists [Ang93]. With the help of a graphical debugger the results and intermediate results can be evaluated in a comfortable environment.

Currently, work is in progress to support design activities. First, this includes additional editors for constructing the design model, e.g., by refining parts of the model of expertise, as well as tools that allow to link design decisions to the requirements that motivate them. Second, the operationalization of the design model is supported by mapping parts of the design model to code of a specific software target environment (C++), which also establishes the basis for evaluating the design model by running it as a hybrid prototype which consists of parts that are still

4. MeMo-Kit is a variant of CoMo-Kit which has been implemented in cooperation with Frank Maurer (cf. [NeM93]) in Smalltalk-80.

5. The interpreter of KARL is implemented in C.

executed using the the KARL interpreter while other parts are already operational in the C++ target environment.

7 Related Work

Since the MIKE approach took KADS-I [WSB92] as a starting point it is not surprising that both approaches have a lot of common features. This holds especially for the structure of the model of expertise. A major difference between the two approaches is the emphasis we put in MIKE on the formalization of the model of expertise and the inherent integration of prototyping into a life cycle oriented approach. From the requirement of providing a formally specified model of expertise the idea of introducing a mediating representation evolved rather naturally - a concept which is not that stressed in CommonKADS [WVS+93] although there exist some similarities between the structure model and several CommonKADS models as e.g. the task model. Most of the CommonKADS models, however, are oriented towards capturing a specific aspect like task modelling or agent modelling, whereas in MIKE the main emphasis is put on describing the same type of knowledge, i.e. expertise knowledge, on different formalization levels.

The Components of Expertise approach (CoE) as well as the corresponding COMMET tool [Ste93] are mainly oriented towards the configuration of an executable application system by analysing it from three perspectives: tasks, information sources, and methods. Task structures, model dependency diagrams, and control diagrams which are used to represent these perspectives provide similar structuring primitives as the structure model in MIKE. A distinctive feature of MIKE when compared to CoE is the formalization aspect since in MIKE the model of expertise provides a completely formal description of the domain and problem solving specific knowledge, whereas in CoE this knowledge is described only on a semi-formal and program code level, respectively. That is, CoE does not provide a description of the expertise on different levels which are related to each other by precisely defined and partially automatically generated links.

A basic distinction between MIKE and PROTEGE [TEG+94] is the fact that PROTEGE puts strong emphasis on the generation of domain specific knowledge acquisition tools from corresponding ontologies - an aspect which is not at all addressed in MIKE. Otherwise the PROTEGE notions of method ontology and mapping relations resemble the MIKE notions of a problem solving method specific terminology as well as of domain views. That is, although the PROTEGE approach does not use the concept of a model of expertise, similar types of knowledge are captured by the various ontologies and the corresponding mappings. PROTEGE does not include the notion of different description levels, e.g. there is no equivalent to the elicitation model or the structure model of MIKE. All the knowledge is either described using the frame language MODEL or the implementation language CLIPS.

Despite of all these differences among the various approaches there is also a strong agreement on some basic concepts like the distinction between knowledge and symbol level descriptions, the separation of domain-specific knowledge and problem solving methods, or the reuse of (generic) building blocks for constructing the different models.

Various approaches in software engineering and information systems engineering also provide means for describing application systems on different formalization levels. For example, the INCOME approach ([LNO+89], [NOS+92]) for developing information systems uses glossaries and object flow diagrams for the semi-formal description of static and dynamic aspects of an information system application. During the conceptual modelling phase this semi-formal description is formalized using an integration of high-level Petri nets and semantic data model concepts resulting in a conceptual schema description. When compared to the different models of MIKE, an object flow diagram corresponds to the structure model, a conceptual schema to the KARL model of expertise. However, a major difference between both approaches is the notion of generic layers as well as the clear separation of data and control flow aspects in the KARL model of expertise. A detailed comparison of MIKE with work done in information system development and software engineering can be found in [AFL92a] and [FAL+93].

8 Conclusion

MIKE integrates semiformal and formal techniques and formalisms in an incremental development process. The semiformal specification is not only used to facilitate the formalization process but is also seen as an important result itself. It structures the complex problem solving process and its informal description of knowledge can be used for documentation. The formal specification describes the functionality of the system precisely, yet abstracting from implementation details. Since the formal specification is operational, it is used as a prototype of the system in order to evaluate the model of expertise. During design, the formal specification is extended with respect to aspects related to the realization of the system, taking non-functional requirements into particular account.

Due to the common underlying conceptual model, the different representations can easily be linked to each other and there is a smooth transition from one representation to the other. By linking the models, we gain the advantage of using, e.g., the semiformal model as an additional documentation of the formal specification. Furthermore, requirements traceability is supported by interrelating our models.

For developing our models and the relationships between models during the specification phase, a tool environment provides different graphical editors and debugging tool which comprises the interpreter for the formal specification language KARL. Current work addresses the enhancement of the tool environment for supporting the design phase, e.g., by offering means to execute the design model as a hybrid prototype for evaluating the effects of design decisions.

Acknowledgements

We thank Jürgen Angele who provided valuable contributions to many of the ideas addressed in this paper.

References

[AFL92a] J. Angele, D. Fensel, and D. Landes: Two Languages to Do the Same? In R. Studer (ed.), *Proceedings of the 2nd Workshop Informationssysteme und Künstliche Intelligenz*, February 24-26, 1992, Ulm, Informatik- Fachberichte 303, Springer-Verlag, 1992.

[AFL92b] J. Angele, D. Fensel und D. Landes: An Executable Model at the Knowledge Level for the Office-Assignment Task. In [Lin92].

[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.

[Ang93] J. Angele: *Operationalisierung des Modells der Expertise mit KARL*, Infix, St. Augustin, 1993.

[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.

[Boe88] B.W. Boehm: A Spiral Model of Software Development and Enhancement. In *IEEE Computer,* May 1988, pp. 61-72.

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

[DKS93] J.-M. David, J.-P. Krivine, and R. Simmons (eds.): *Second Generation Expert Systems*. Springer-Verlag, Berlin, 1993.

[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, 513-528.

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

[FAS94] D. Fensel, J. Angele, and R. Studer: The Specification Language KARL and its Declarative Semantics. Research report, no 307, Institut AIFB, University of Karlsruhe, 1994.

[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.

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

[Fen93] D. Fensel: *The Knowledge Acquisition and Representation Language KARL*, Ph.D. thesis, University of Karlsruhe, 1993. Tp appear Kluwer Academic Publisher, Boston, 1995.

[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, 1992, 55-62.

[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, Amsterdam, 1990.

[Lan94a] D. Landes: DesignKARL - A Language for the Design of Knowledge-Based Systems. In *Proceedings of the 6th International Conference on Software Engineering and Knowledge Engineering SEKE'94*, Jurmala, Latvia, June 20-23, 1994.

[Lan94b] D. Landes: Addressing Non-Functional Requirements in the Development of Knowledge-Based Systems. In *Proceedings of the 1st International Workshop on Requirements Engineering: Foundation of Software Quality REFSQ'94*, Utrecht, The Netherlands, June 6-7, 1994.

[LaS94a] 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.

[LaS94b] D. Landes and R. Studer: Mechanisms for Structuring Knowledge-Based Systems. In *Proceedings of the 5th International Conference on Database and Expert Systems Applications DEXA'94*, Athens, Greece, September 7-9, 1994.

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

[LNO+89] G. Lausen, T. Németh, 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, 1989.

[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)*, May 24-28, 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

Knowledge Acquisition for Knowledge-Based Systems, Proceedings of the 7th European Workshop EKAW´93, Toulouse, France, September 6-10, Lecture Notes in AI, no 723, Springer-Verlag, Berlin, 1993.

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

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

[NOS+92] T. Németh, A. Oberweis, F. Schönthaler, and W. Stucky: INCOME: *Arbeitsplatz für den Programmentwurf interaktiver betrieblicher Informationssysteme*. Research report, no 251, Institut AIFB, University of Karlsruhe, 1992.

[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)*, vol III, Banff, Canada, Januay 30 - February 4, 1994.

[PoB88] C. Potts and G. Bruns: Recording the Reasons for Design Decisions. In *Proceedings of the 10th International Conference on Software Engineering*, Singapore, April 11-15, 1988, 418-427.

[Sch93] G. Schreiber: Operationalizing Models of Expertise. In [SWB93], 119-149.

[Ste93] L. Steels: The Componential Framework and its Role in Reusability. In [DKS93], 273-298.

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

[TEG+94] S.W. Tu, H. Eriksson, J. Gennari, Y. Shahar, and M.A. Musen: Ontology-Based Configuration of Problem Solving Methods and Generation of Knowledge-Acquisition Tools: Application of PROTEGE-II to Protocol-Based Decision Support. To appear in *Artificial Intelligence in Medicine*, 1994.

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

[WSB92] B. Wielinga, A. Schreiber, and J. Breuker: KADS: A Modelling Approach to Knowledge Engineering. In: *Knowledge Acquisition 4 (1)*, 1992, 5-53.

[WVS+93] B. Wielinga, W. Van de Velde, G. Schreiber, and H. Akkermans: Towards a Unification of Knowledge Modeling Approaches. In [DKS93], 299-335.

[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.