Domain and Task Modeling in MIKE

J. Angele, D. Fensel, and R. Studer
Institute AIFB, University of Karlsruhe
76128 Karlsruhe, Germany,
tel. [049] (0)721 608 3923, fax [049] (0)721 693717
e-mail: {angele | fensel | studer}@aifb.uni-karlsruhe.de

Abstract

The paper describes the MIKE (*Model-based and Incremental Knowledge Engineering*) approach for the development of knowledge-based systems (kbs). It integrates semiformal specification techniques, formal specification techniques, and prototyping into a coherent framework. This allows the domain and task model of a kbs to be described on different formalization levels. All activities in the building process are embedded in a cyclic life cycle model. 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). Since KARL is executable the model of expertise can be developed and validated by prototyping. 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 functional and non-functional aspects of 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. Though the MIKE approach aims at supporting the building process of kbs, its principles and methods apply also to the development of non-knowledge-based software systems, e.g. information systems.

Keywords

Knowledge Engineering, Knowledge Acquisition, Domain Modeling, Task Modeling, Problem-Solving Method, MIKE, KARL

1 INTRODUCTION

In earlier days the development of a kbs has been seen as a transfer process of knowledge from the head of the expert into the computer system. As a consequence it was assumed (i) that interviewing the expert brings all necessary knowledge to day, (ii) that the knowledge in the kbs is best structured in the same way as the expert structures it, (iii) that the existing representation formalisms, which were formalisms for the realization of the final kbs, are adequate to capture this knowledge, and (iv) that the kbs might be developed using the rapid prototyping approach, i.e. that the acquired knowledge may be implemented immediately and that the running prototype may be used to validate the acquired knowledge.

In contrast to the expectations this paradigm failed in building large expert systems for commercial use in many cases, due to the following reasons:

- Typically kbs are developed to solve very complex problems or even problems which are not entirely understood (Shaw,Gaines,1992). So the functionality of the system is not completely available in advance. Instead such a specification must be developed iteratively in cooperation with expert(s) and the user(s) of the system.
- While the expert may consciously articulate some parts of his or her knowledge, he or she will not be aware of a significant part of this knowledge since it is hidden in his or her skills. This knowledge is not directly accessible by interviews but only by observing the expert and interpreting these observations.
- Even if the functionality of a kbs is entirely specifiable the complexity of every computational mechanism might be so high, that the problem in its entire generality may only be solved for small instances. Many AI-problems in their general formulation are at least NP-hard problems (Nebel,1995). Experts solve such problems by using a large amount of domain specific knowledge, which allows to restrict the problem, to approximate the problem or to reformulate the problem in order to solve a simpler problem efficiently or to use domain specific heuristics which reduce the average complexity (Nebel,1995). It is often exactly this knowledge which distinguishes an expert from a novice and moreover its exactly this knowledge which is only partially conscious to the expert.

Due to these reasons it is not sufficient to give a detailed functional specification of a kbs and to build a solution using 'normal' computer science know-how. Instead task and domain specific knowledge and task and domain specific heuristics are necessary in order to be able to build a solution which allows to solve larger instances of the problem in a reasonable time. Thus the process of building a kbs is nowadays seen as a *modeling activity*, i.e. as the construction of several models capturing different types of knowledge (Clancey,1989). Building a kbs means building a computer model with the aim of imitating an expert´s approach. It is not intended to create a cognitive adequate model, i.e. to simulate the cognitive processes of an expert in general, but to create a model which offers similar results in problem-solving for problems in the area of concern.

The models resulting from this modeling process must contain different kinds of knowledge necessary to solve the task at hand:

- Domain specific static knowledge about terminology, relationships and facts. This domain knowledge comprises all knowledge to solve the task in principle. These domain models may partially be reused for solving other problems in the same domain. Domain ontologies are reusable model components which provide a conceptualization of a specific domain and are shareable and reusable across different tasks (Gruber,1993).
- Due to the failure of the general problem solver approach in AI, it is not possible to model all relevant knowledge and to solve every problem in a given domain using a general deduction mechanism. Instead task specific *problem-solving methods (psm)* must be used in order to solve the given task. These problem-solving methods may be kept generic, i.e. independent of the concrete domain, but specific for the given task. Therefore these problem-solving methods constitute building blocks which may be reused for similar tasks in another domain (Breuker,Velde,1994).

• Task-specific heuristics formulated in domain-specific terms which allow to reduce the average complexity of the used problem solving method.

Thus it is even indispensable for building a kbs to perform a detailed domain analysis and domain modeling as well as an analysis and modeling of the task in an early stage of this building process. The distinction of symbol level and knowledge level (Newell,1982) gives rise to an abstract description of the *task* solved by the system and the *knowledge*, which is required to solve the task. This knowledge level description is built independently of the design and implementation activity. The separation of analysis and design/implementation is an analogue to the separation of analysis or requirement engineering on the one hand and design and implementation on the other hand in software engineering.

In this paper we present some aspects of the *MIKE approach* (*Model-based and Incremental Knowledge Engineering*) (Angele et al, 1993)*, which aims at a development method for kbs covering all steps from the initial specification to design and implementation. MIKE proposes the integration of *life cycle models*, *prototyping*, *semiformal*, and *formal specification techniques* into a coherent framework:

• *Informal and semiformal models* of the knowledge provide a high and informal level for description. Graphical means similar to entity-relationship diagrams, data flow diagrams, flow charts, and state-transition diagrams are used. This type of information is easy to understand and very useful as a mediating representation for the communication between the domain expert, the user and the system developer. These models also contain non-functional requirements which have to be met by the system.
• The *formal* specification of the *domain* and *problem-solving knowledge (*knowledge how to solve the task) in the language KARL allows to describe the functionality of a system in an unambiguous and precise way.
• Making this formal model *executable* adds the flavour of prototyping to the specification process. The model may 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.
• Additional representations document modelling decisions made during the various phases of the life-cycle (Landes,Studer,1995). This enables requirements traceability, i.e. it is recorded which parts of the implementation are addressing a particular requirement.
• All different activities of the building process itself are embedded in a cyclic life cycle model (Boehm, 1988).

This paper presents the models and the description formalisms used in MIKE, namely the structure model described by the semiformal hypermedia-based formalism *MEMO* (Neubert,1993) and the model of expertise formulated in the formal and executable *Knowledge Acquisition and Representation Language KARL* (Fensel et al,1995), (Fensel,1995a), (Angele,1993). An extension to KARL, namely *DesignKARL* (Landes,1994) has been developed to describe the design model. The description of the models at different abstraction and formalization levels and the strong conceptual model all models are based on enables a smooth transition from a semiformal to a formal specification and further on to the design of the system. This conceptual model enables a clear separation of domain specific knowledge and knowledge to describe the problem-solving method (task related knowledge). The clear separation of different knowledge types allows to describe the problem-solving method generically and thus to reuse it in another domain, and it allows to reuse parts of the domain model for another task. Due to the executability of all formal models (the formal specification and the design model) and due to the cycles described in the life cycle model the different models may be constructed and validated by prototyping.

Though the MIKE approach addresses the building process of kbs, these principles and

*. This paper presents an update to earlier descriptions of the MIKE approach.

methods apply to requirements analysis and specification for information systems and 'conventional' software systems as well. For instance in (Fensel et al, 1993) it is shown how KARL may be used to formalize the system model of Structured Analysis.

The paper is organized as follows. In section two, the semiformal models and their description formalisms are described. Then the model of expertise which is specified in the formal specification language KARL is discussed in section three. Section four addresses the embedding of the different activities in a cyclic life cycle model. Section five gives a short overview of some of the tools that support the knowledge acquisition process in MIKE and some applications of the MIKE approach. Finally, related work is described and a conclusion is given.

2 THE SEMIFORMAL MODELS

In this section we describe the models which describe the domain and task specific knowledge semiformally, i.e. in natural language already structured according to the underlying conceptual model.

Developing a formal specification directly from informal knowledge protocols gained from interviews with experts or by observing experts is rather difficult. Therefore, mediating representations are constructed in MIKE before starting the formalization process (Neubert,1993).

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 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) (Neubert,1993). 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 and is 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.

The *structure model*, which is developed based on the elicitation model gives a more structured description of the knowledge. In Figure 1 at the left the structure model is sketched for the *Sisyphus office assignment task.* The Sisyphus office assignment task is concerned with assigning a set of employees to a set of office places. Both, employees and places are described by a set of properties. A valid assignment must assign a place to each employee and must fulfil several constraints like 'smokers must not be placed with nonsmokers in the same room', 'the boss must have a central, large single-room', etc. This task was used to compare different knowledge engineering approaches (Linster,1994).

The structure model is composed of the following contexts:

- 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 embedded in a control flow description. These activity nodes are elements on *one* hierarchy level of the activity refinement hierarchy. In Figure 1 the ordering context describes a loop of the three activities.
- The *concept context* comprises domain specific concept nodes which describe static domain aspects. Moreover, concepts are related by *Generalization links*, *Aggregation links* and *Association. links*. Association links can be added by the user to describe an arbitrary relationship between concepts. The graphical notation has been adopted from OMT (Rumbaugh et al,1991). In Figure 1, one Generalization link and two Aggregation links are shown, that is, it is modelled that an *Employee* and a *Place* are part of a *Pair*.
- A *data flow context* is also a view on *one* hierarchy level of *activity nodes*. Here, activity nodes are related with concept nodes by so-called *data flow links*. A data flow context describes the flow of data during the problem-solving process. Figure 1 shows in the data flow context section that e.g. *Employee* and *Place* are input for the activity *Create Pairs*.
- A further context of the structure model which is constructed from the information provided by the elicitation model is the *NFR context* (Non-Functional Requirements context) which is used for describing the NFRs the system must fulfil. Node types like requirements category or evaluation criterion as well as link types like correlation or conflict solution are used to describe in a semi-formal way types of requirements as well as various relationships which may exist between them (see (Landes,Studer,1995) for details). The non-functional requirements modelled in the NFR context form the basis for the design process.

The structure model together with its nodes and links is constructed on the basis of the node contents of the elicitation model. Its parts are related to the corresponding parts of the elicitation model via *elicitation links*. The graphical representation of the control flow, the data flow and the static knowledge have been adopted from corresponding representations in software engineering (program flow diagrams, data flow diagrams, OMT notation). So large parts of this structuring process can be done by the expert himself, supported by the knowledge engineer. The resulting structure model is an adequate basis for the development of the formal models by the knowledge engineer. This model comprises the knowledge about the functional aspects as well as non-functional aspects. The domain knowledge is clearly separated from the knowledge about the problem-solving method (task related knowledge). All the different parts of the structure model are well-integrated and linked to the appropriate parts of the elicitation model.

3 THE FORMAL MODEL

3.1 The KARL Model of Expertise

The conceptual model underlying KARL is derived from the KADS *model of expertise* (Schreiber et al,1993). This model describes the functional specification of a kbs. It is structured into different layers each containing a different type of knowledge. The KARL model of expertise distinguishes three types of knowledge at three different layers. These types establish a static view, a functional view, and a dynamic view to the kbs.

Domain knowledge at the *domain layer* consists of static knowledge about the application domain. 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 task.

Inference knowledge at the *inference layer* specifies the *inferences* that can be made using the domain knowledge, and the *knowledge roles,* which model input and output of the inferences. Three types of knowledge roles are distinguished. 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 to the domain layer are called *terminators*. The inferences and roles together with their data flow dependencies constitute a description of the problem-solving method applied. The roles and the inference actions are specified in a *problem-solving-method-specific terminology* independently of the domain-specific terminology.

A *domain view* specifies the relationship between the knowledge used at the inference layer and the domain-specific knowledge. It performs a transformation of the domain-specific terminology of the domain layer to the problem-solving-method-specific terminology at the inference layer.

Dynamic control knowledge at the control layer is used to specify *control* over the execution of the inferences of the inference layer. It determines the sequence in which the inferences are activated.

Inference and control knowledge are domain independent, i.e. they describe the problem-solving process in a generic way using a problem-solving-method-specific terminology. Such a problem-solving method can be reused for different application problems. Using the MIKE approach it has been started to build up a library of formally specified problem-solving methods: Hill Climbing, Chronological Backtracking, Beam Search, Cover-and-Differentiate, Board-game Method, Propose-and-Exchange, and Propose-and-Revise. In (Fensel,1995b) limitations and assumptions of the psm Propose-and-Revise have been analysed which is a precondition to match features of a given task to appropriate psms.

Figure 1 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. For instance the employees are described by their properties like their name, whether they smoke etc. The inference layer contains the elementary inference steps and knowledge roles. Components and slots 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 control layer. The inference actions *Create*, *Prune*, and *Check* are repeatedly activated in this sequence until a solution is found, i.e. the role *Solution* at the inference layer is not empty. The inference layer is described in generic terms such as *Components* and *Slots*. These generic terms are mapped to the domain specific terms *Employees* and *Place* at the domain layer.

All parts of the model of expertise are linked to the corresponding parts of the structure model via *formalization links* and are thus in turn connected to the corresponding parts of the elicitation model.

3.2 Knowledge Acquisition and Representation Language (KARL)

A description of KARL may be found in (Fensel,1995a), and (Fensel et al,1995). The main characteristics of KARL is the combination of a *conceptual description* of a knowledge-based system with a *formal* and *executable specification*. In the following we give a

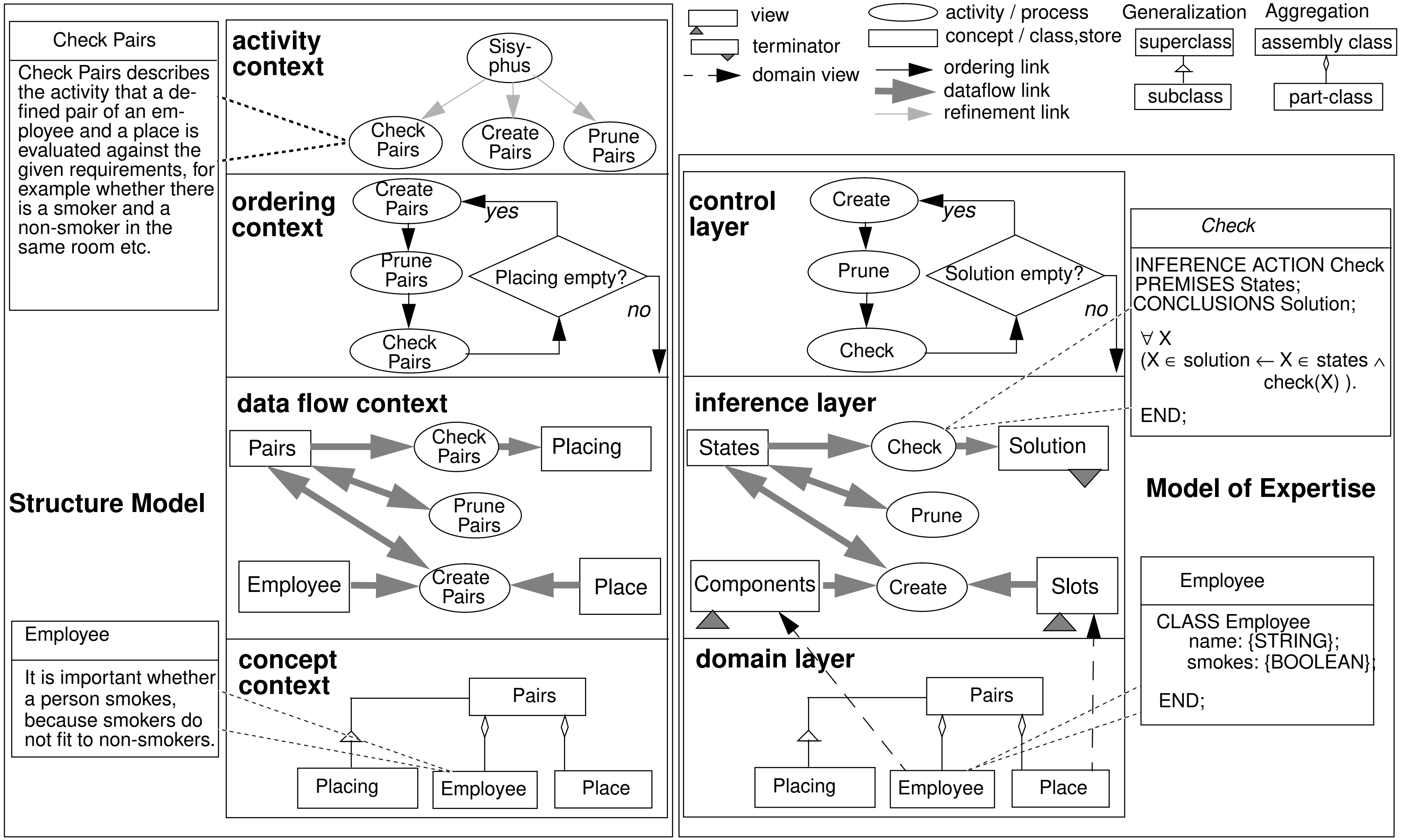

Figure 1 Structure model at the left (nodes have informal content expressed in domain specific terms) and of a model of expertise at the right (nodes with formal content expressed in generic terms) for the Sisyphus office assignment task.

brief overview of KARL and its use to describe the different types of knowledge at the different layers of the KARL model of expertise.

Logical-KARL (L-KARL

L-KARL enriches the modelling primitives of first-order logic by epistemological primitives but preserves its model-theoretical semantics. These additional primitives allow to describe static aspects more adequately than pure first-order logic. By 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. Classes which are arranged in an is-a hierarchy with multiple attribute inheritance are used to describe terminological knowledge. Intentional and factual knowledge is described by logical formulae over classes, instances of classes, and values. L-KARL is used to describe the following parts of the model of expertise:

- The domain knowledge at the domain layer is defined by a class hierarchy with attribute inheritance and by additional logical formulae describing sufficient and necessary conditions. This knowledge is described in a domain-specific terminology.
- The static generic knowledge of the roles at the inference layer, i.e. generic concepts together with their attributes are described in L-KARL. This knowledge is described in a problem-solving-method-specific terminology.
- The input-/ output specification of an elementary inference action is specified by L-KARL formulae.
- The mapping of the domain knowledge to the generic inference knowledge is described in the views using L-KARL formulae.

The logical language to describe conditions between classes, objects, and values and to specify the views and elementary inference actions is restricted to Horn logic with equality extended by stratified negation. Using a logical language enriched by additional primitives allows to describe the respective parts of the model on an adequate abstraction level and the restriction to Horn logic allows to execute a KARL model of expertise and thus supports the construction process by prototyping.

Procedural-KARL (P-KARL)

In KARL knowledge about the control flow is explicitly described by the logical language P-KARL. The control flow at the control layer, i.e. the sequence of the activation of inference actions, is specified by the modeling primitives *sequence*, *loop*, and *alternative* which are similar to the control flow statements of procedural programming languages.

KARL as a Formal And Executable Specification Language

The KARL model of expertise contains the description of domain knowledge and knowledge about the problem-solving method (inference and control layer). The gist of the matter of the *formal semantics* of KARL is therefore the integration of static and procedural knowledge. For this purpose, two different types of logic have been used and 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. Both types of languages have been combined to specify the semantics of KARL (see (Fensel,1995a)). Based on this semantics, a constructive semantics and an optimised evaluation strategy have been developed which establish the foundation of an interpreter and debugger for KARL (Angele, 1993). By this way it becomes possible to validate a formal specification by executing it and to create this specification by a prototyping approach.

4 MODEL DEVELOPMENT

In this section we will describe the knowledge engineering *process* in which the models of MIKE are developed and related to each other. Moreover, we will describe the *product* of this knowledge engineering process, the entire model environment.

4.1 The Process of Model Development

The phases and subphases of kbs development according to the MIKE approach are shown in Figure 2. These steps are performed in a cyclic fashion (Angele et al, 1993), (Angele, 1993) guided by a *spiral model* (Boehm, 1988) as process model (Figure 3). The steps which are of particular interest in the context of this paper are printed in *italics* in Figure 2.

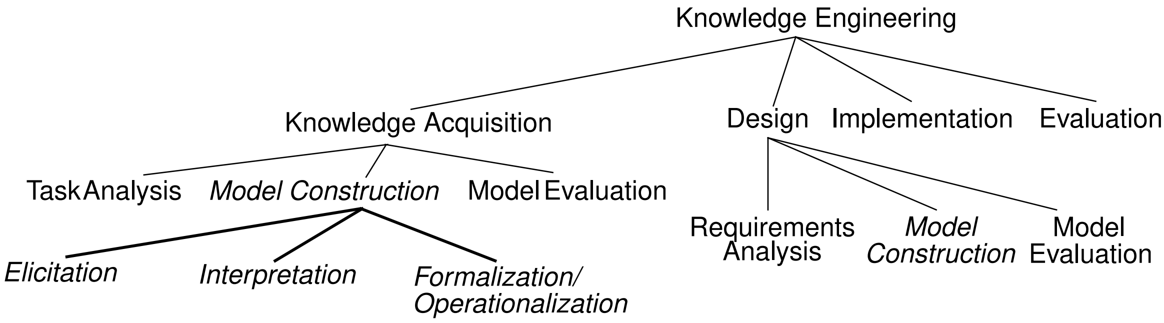

Figure 2 Phases in the MIKE life-cycle.

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. The resulting knowledge protocols are stored in protocol nodes of the elicitation model.

Knowledge structures which may be identified in the knowledge protocols are modelled by the corresponding contexts of the structure model. The semiformal structure model is the result of the *Interpretation* phase. It provides a first structured description of the emerging knowledge structures and is thus a first valuable result on its own.

The structure model is the foundation for the *Formalization/Operationalization* process which results in the model of expertise described in KARL. The construction of the domain layer of a model of expertise is in addition supported by a collection of formal, lexical, and graphical methods for reusing commonsense ontologies (Pirlein,Studer,1994).

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

The *Design* phase is performed on the basis of the model of expertise after it has been evaluated with respect to the required functionality. It results in the *design model*. Design decisions are motivated by non-functional requirements and the constraints imposed by potential software and hardware target environments.

In Figure 3 the process model of the entire knowledge engineering process in MIKE is shown. The phases Knowledge Acquisition, Design, Implementation and Evaluation are performed cyclically until the desired system has been constructed (large spiral). During

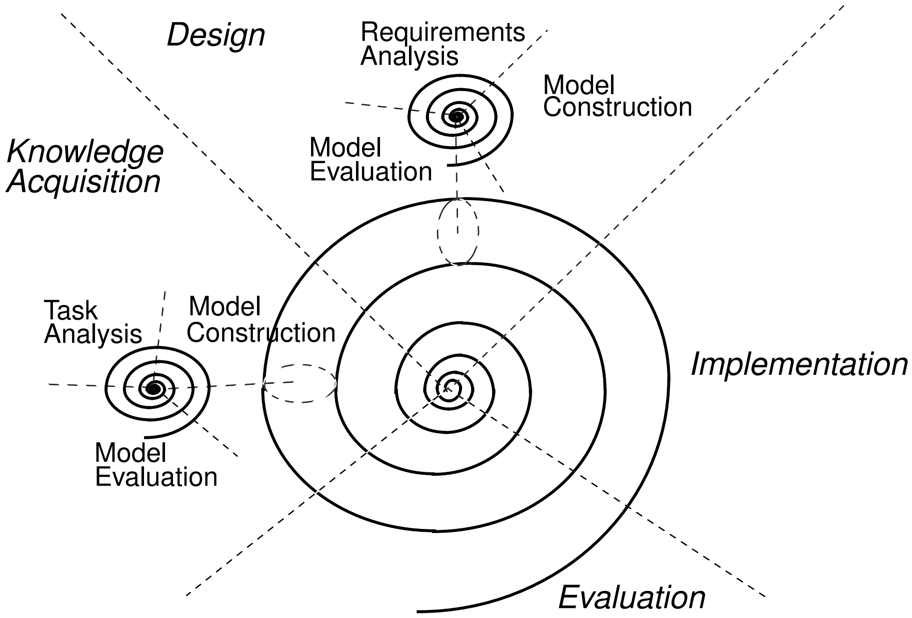

Figure 3 The cyclic process model.

the Knowledge Acquisition phase its subphases are performed in a cycle until the desired functionality of the model of expertise is achieved. The subphases of the Design phase are also performed in a cyclic manner until the desired non-functional requirements are met. The evaluation of the constructed models by prototyping within each main phase allows to separate these phases. In each phase the knowledge engineer may focus on different aspects (functional, non-functional, implementation) without considering other aspects: This divide-and-conquer strategy reduces the complexity of the entire development process considerably.

4.2 The Modelling Result

The result of the different phases described in section 4.1 is a complex model environment including several interrelated models. Figure 1 shows a small section of the model environment developed for the Sisyphus example during knowledge acquisition. The knowledge within the structure model is related to the elicitation model via *elicitation links*. Concept nodes (e.g. *Pair*) and activity nodes (e.g. *Check Pairs*) are related to protocol nodes, in which they have been described using natural language. 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 to the knowledge element *Employee* of the concept context, where an informal explanation is given. In the same way inference steps are related to activity nodes of the structure model. The design model refines the model of expertise by refining inferences into algorithms and by introducing additional data structures. These parts of the design model are linked to the corresponding inferences of the model of expertise and are thus in turn linked to the structure model and to the elicitation model (Landes,1994).

5 TOOL SUPPORT AND APPLICATIONS

The hypermedia based MIKE-tool (Neubert,1993) supports the graphical construction of the elicitation model, the structure model and the formal model of expertise by different editors. Additionally it supports the management of a library of problem-solving methods. After the elicitation, the knowledge engineer enters the knowledge protocols into the elicitation model using the elicitation model editor. 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 automatically by the editors.

For executing the operational specification, the model of expertise, an interpreter for KARL exists (Angele, 1993). With the help of a graphical debugger the intermediate and final results can be evaluated in a comfortable environment.

During the last year the MIKE approach has been evaluated by conducting several case studies in cooperation with external partners. In one case study, expert knowledge for supporting the recycling of building rubble has been modeled (Fichtner et al,1995). It resulted in a prototypical kbs providing support in selecting the most appropriate technique for dismantling a building. This case study exemplified among others that (i) the structure model is well suited for a first structuring of expert knowledge in a domain and that (ii) the structure model can be built up and evaluated in close cooperation with the domain expert.

Another case study has been carried out in the area of online helpdesks. Objective of this case study was the development of a prototypical kbs supporting an online helpdesk in diagnosing troubles in communication networks. Special emphasis has been put on exploiting the executability of a KARL model of expertise for an iterative and incremental development process. It has been shown that an incremental and intertwined construction of the structure model and the model of expertise can be supported by the MIKE tool rather well.

As a second problem for comparing different knowledge engineering approaches an elevator configuration task has been posed (Yost,1992). For configuring an elevator values have to be assigned to a predefined set of parameters. The entire configuration has to fulfil various constraints. This complex task especially showed the adequacy of the language KARL for representing the knowledge on an abstract level. For validating the resulted complex model of expertise it was even indispensable to have a running prototype.

On the other hand, all these case studies resulted in some modifications of the structure model and the model of expertise. A deficiency of the structure model was the very small collection of offered modeling primitives. Therefore, additional modeling primitives have been included in the structure model, e.g. an alternative construct for the ordering context and additional pre-defined relationships in the concept context. Furthermore, several experts complained that the structure model and the KARL model of expertise used different graphical primitives. As a consequence, MIKE now offers as far as possible the same graphical primitives for both models. In addition, the OMT notation has been adopted. Especially the elevator configuration task gave us various insights into some deficiencies of KARL which shall be eliminated in a future version.

6 RELATED WORK

Since the MIKE approach took KADS-I (Schreiber et al,1993) 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. Major differences between both approaches are: (i) The semi-formal representation within the structure model in MIKE. There exist some similarities between the structure model and several CommonKADS models as e.g. the task model. (ii) The emphasis we put on the formalization of the model of expertise. This formal model provides a profound basis for the further development process. (iii) The smooth transition between informal and formal descriptions using the same conceptual model for all descriptions in MIKE. (iv) The inherent integration of prototyping into a life cycle oriented approach in MIKE.

When compared e.g. to ARIES (Johnson et al,1992) a distinctive feature of MIKE is the semi-formal MEMO formalism supporting a mediating representation between informal natural language description of requirements and the completely formal KARL specifications. A further distinction is the clear separation of generic knowledge from domain / task specific knowledge in the KARL model of expertise. Thus the reuse of domain models is supported as proposed in (Johnson et al,1992). In addition, the generic problem solving knowledge may be reused as well. In contrast to ARIES which puts emphasis on a highly expressive knowledge representation language the KARL language has a rather restrictive expressive power in order to be able to provide an adequate KARL prototyping environment.

Within Structured Analysis (Yourdon,1989) or within the object oriented method OMT (Rumbaugh et al,1991) the system is considered from different points of view. The function oriented point of view considers the data flows between the processes. This point of view strongly resembles the data flow context in the structure model. The dynamic point of view resembles the ordering context. The object model in OMT corresponds to the concept context. The description in SA or OMT is semi-formally only. In contrast to MIKE no formal description of the model is provided in addition. Furthermore there is no distinction between generic and domain specific 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 (Oberweis et al,1994) 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.

The construction and reuse of models as part of requirements engineering has gained a lot of interests in recent years (see e.g.(Jarke et al,1993)). In (Sutcliff,Maiden,1994) the notion of object system models is introduced for describing types of applications like e.g. object composition (which can be instantiated to manufacturing systems) or agent-object control (which can be instantiated to command and control applications). Such object system models are embedded into a specialization hierarchy: each object type may be refined to further specialized object system models by using additional knowledge types for discrimination. These object system types provide a framework for reuse during requirements engineering since they can be used as initial generic descriptions of application types. When compared to the notion of reusable problem-solving methods object system models are used rather for describing problem spaces whereas problem-solving methods are used for specifying solution spaces (compare (Sutcliff,Maiden,1994)). It should be furthermore noticed that the notion of domain model as used in (Sutcliff,Maiden,1994) has a much broader meaning than the notion of domain model as used in the KADS or MIKE framework, since there behavioural aspects are not included in the domain model, rather these aspects are modeled at the inference and control layer of the model of expertise.

7 CONCLUSION

MIKE integrates semiformal and formal description formalisms in an incremental development process. The semiformal specification of the structure model is not only used to facilitate the formalization process, but is also seen as an important result itself. It structures the domain knowledge and the knowledge about the problem-solving process (task related knowledge) and due to its semi-formal description it 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 to validate the model of expertise. The clear separation of knowledge about the problem-solving method and domain knowledge allows the reuse of these parts. 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 all the models.

In that way MIKE addresses one of the main topics which have been put on the research agenda for requirements engineering in software engineering and information systems engineering: combination of different representations (Pohl et al,1995) based on a strong conceptual model involving aspects like smooth coupling of different representations, traceability and consistency.

For constructing the models and their relationships the MIKE tool environment provides different integrated graphical editors and a debugging tool which comprises the interpreter for the formal and executable specification language KARL.

Current work addresses among others the reuse of problem-solving methods and domain models during the knowledge acquisition phase. This includes a detailed analysis of the characteristics of problem-solving methods and their mutual dependencies with domain models (see (Fensel,1995b), (Fensel,1995c)).

Acknowledgements

We thank Susanne Neubert and Dieter Landes who provided valuable contributions to many of the ideas addressed in this paper.

8 REFERENCES

J. Angele, D. Fensel, D. Landes, S. Neubert, and R. Studer (1993): 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.

J. Angele (1993): Operationalisierung des Modells der Expertise mit KARL (Operationalization of the Model of Expertise with KARL), Ph.D. Theses in Artificial Intelligence, No. 53 , infix, St. Augustin, 1993 (in German).

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

J.A. Breuker and W. Van de Velde (eds.) (1994). *The CommonKADS Library for Expertise Modeling*. IOS Press, Amsterdam, 1994.

W.J. Clancey (1989): The Knowledge Level Reinterpreted: Modeling How Systems Interact. In: *Machine Learning 4*, 1989, 285-291.

D. Fensel, J. Angele, D. Landes, and R. Studer (1993): 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.

D. Fensel, J. Angele, R. Studer (1995): The Knowledge Acquisition and Representation Language KARL. Research report, no 316, Institute AIFB, University of Karlsruhe, Mai 1995.

D. Fensel (1995a): *The Knowledge Acquisition and Representation Language KARL*. Kluwer Academic Publisher, Boston, 1995.

D. Fensel (1995b): A Case Study on Assumptions and Limitations of a Problem Solving Method. In: *Proceedings of the 9th Banff Knowledge Acquisition for Knowledge-Based System Workshop (KAW´95), Banff, Canada,* 1995.

D. Fensel (1995c): The Mincer Metaphor: A New View on Problem-Solving Methods for Knowledge-Based Systems? Department SWI, University of Amsterdam, 1995.

W. Fichtner, D. Landes, Th. Spengler, M. Ruch, O. Rentz, and R. Studer (1995): Der MIKE Ansatz zur Modellierung von Expertenwissen im Umweltbereich - dargestellt am Beispiel des Bauschuttrecyclings. In: H. Kremers et al. (eds.): *Space and Time in Environmental Information Systems, Proc. 9th Int. Symposium on Computer Science for Environmental Protection*, Berlin, September 1995, Metropolis Verlag (in German).

T.R. Gruber (1993): A Translation Approach to Portable Ontology Specifications. In: *Knowledge Acquisition 5*, 2, 1993, 199-221.

M. Jarke, J. Bubenko, C. Rolland, A. Sutcliffe, and Y. Vassiliou (1993): Theories Underlying Requirements Engineering: An Overview of NATURE at Genesis. In: *Proc. IEEE Symposium on Requirements Engineering*, San Diego, 1993.

D. W.L. Johnson, M.S. Feather, and D.R. Harris (1992): Representation and Presentation of Requirements Knowledge. In *IEEE Trans. of Software Engineering 18*, 10 (October 1992), 853-869.

D. Landes (1994): 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.

D. Landes and R. Studer (1995): The Treatment of Non-Functional Requirements in MIKE. In: *Proc. of the 5th European Software Engineering Conference (ESEC´95)*, Sitges, 1995, Springer LNCS, Vol. 989, 1995.

M. Linster (ed.) (1994): Sisyphus: Models of Problem Solving. In: *Int. Journal of Human-Computer Studies 40, 2* , special issue, February 1994.

B. Nebel (1995): Artificial Intelligence: A Computational Perspective. To appear in: G. Brewka (ed.): *Essentials in Knowledge Representation*.

S. Neubert (1993): 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.

A. Newell (1982): The Knowledge Level, *Artificial Intelligence*, vol 18, 1982.

A. Oberweis, G. Scherrer, and W. Stucky (1994): INCOME/STAR: Methodology and Tools for the Development of Distributed Information Systems. In: *Information Systems 19*, 8, 1994, 643-660.

T. Pirlein and R. Studer (1994): KARO: An Integrated Environment for Reusing Ontologies. In: Steels et al. (eds): *A Future of Knowledge Acquisition, Proc. 8th European Knowledge Acquisition Workshop (EKAW´94)*, Hoegaarden, LNCS 867, Springer, 1994.

K. Pohl, G. Starke, and P. Peters (1995): Workshop Summary First Int. Workshop on Requirements Engineering: Foundation of Software Quality (REFSQ´94). In: *ACM SIGSOFT 20*, 1, pp. 39-45, January 1995.
J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy und W. Lorensen (1991): *Object-Oriented Modeling and Design*. Prentice Hall, 1991
A.G. Sutcliff and N.A.M. Maiden (1994): Domain Modeling for Reuse. In: *Proc. 3rd Int. Conf. on Software Reuse*, Rio de Janeiro, 1994.
G. Schreiber, B. Wielinga, and J. Breuker (eds.) (1993): *KADS - A Principled Approach to Knowledge-Based System Development*, Academic Press, London, 1993.
A.Th. Schreiber, B.J. Wielinga, R. de Hoog, H. Akkermans, and W. van de Velde (1994): CommonKADS: A Comprehensive Methodology for KBS Development. In: *IEEE Expert*, December 1994, 28-37.
M.L.G. Shaw and B.R. Gaines (1992): The Synthesis of Knowledge Engineering and Software Engineering. In: P. Loucopoulos (ed.): *Advanced Information Systems Engineering*, LNCS 593, 1992, 208-220.
G.R. Yost (1992): *Configuring Elevator Systems.* Technical report, Digital Equipment Co., Marlboro, Massachusetts, 1992.
E. Yourdon (1989): *Modern Structured Analysis*, Prentice-Hall, Englewood Cliffs, 1989.

9 BIOGRAPHY

Juergen Angele received the diploma degree in computer science in 1985 from the University of Karlsruhe, Karlsruhe, Germany. From 1985 to 1989 he worked for the companies AEG in Konstanz, Germany and SEMA GROUP in Ulm, Germany. From 1989 to 1994 he was a Research and Teaching Assistant at the University of Karlsruhe. He received the Ph.D. from the University of Karlsruhe in 1993 on the subject of the operationalization of the language KARL. In 1994 he became Associate Professor of Applied Computer Science at the Fachhochschule of Braunschweig, Germany. His current research interests include knowledge engineering, software engineering, formal specification techniques, and problem-solving methods.

Dieter Fensel received the diploma degree in sociology and the diploma degree in computer science in 1989 from the Technical University of Berlin, Germany. From 1989 to 1994 he was a Research and Teaching Assistant at the University of Karlsruhe. He received the Ph.D. from the University of Karlsruhe in 1993 on the subject of the specification language KARL. At the moment he is a guest researcher at the University of Amsterdam, department SWI. His current research interests include knowledge engineering, formal specification languages, problem-solving methods, and machine learning techniques.

Rudi Studer received the diploma degree in computer science and the doctoral degree, in 1975 and 1982, respectively, from the University of Stuttgart, Stuttgart, Germany. He was as a Research Assistant with the Institute of Computer Science of the University of Stuttgart from 1975 to 1985. In 1985 he joined the Scientific Center of IBM Germany as a Project Leader and Manager, respectively. In 1989 he became Full Professor of Applied Computer Science at the University of Karlsruhe. His research interests include knowledge engineering, formal specification techniques, and knowledge discovery in databases.