G - KARL:
A graphical, formal, and executable specification language

Juergen Angele and Rudi Studer
Institute AIFB
University of Karlsruhe (TH)
D-76128 Karlsruhe, Germany
e-mail: {angele, studer}@aifb.uni-karlsruhe.de

Abstract

In this paper we present the language G-KARL. G-KARL allows to describe the static, the functional and the dynamic view to a knowledge based system (kbs) graphically. This graphical representation makes the communication between the expert and the knowledge engineer easier. The underlying conceptual model for G-KARL is derived from the KADS model of expertise. Every primitive of G-KARL may be mapped to a language primitive of the language KARL (Knowledge Acquisition and Representation Language). KARL is a formal language, so every primitive of G-KARL has a defined formal semantics. KARL is an executable language, so G-KARL is also executable which supports the validation of the model of expertise by testing and thus allows the model of expertise to be built by prototyping. G-KARL allows to visualize the execution of this model which additionally supports the validation process. While G-KARL contains well established graphical means for the different views like the OMT-notation for the static view, data flow diagrams for the functional view and program flow diagrams for the dynamic view, G-KARL also offers new graphical primitives for specifying elementary inference actions within data flow diagrams and for specifying sufficient and necessary conditions within the static view. In KARL these model elements are described by L-KARL, a logical language enriched by additional modeling primitives.

1 Introduction

In contrast to the earlier transfer paradigm 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 [Cla89]. 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.

G-KARL and KARL *(Knowledge Acquisition and Representation Language)* [FAS97], [Fen95] have been developed within the *MIKE approach* (*Model-based and Incremental Knowledge Engineering*) [AFS96] 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.

The central model within the development process of MIKE is the *model of expertise* (cf. [SWB93]) which captures the entire functionality of the system and establishes a static, functional, and dynamic view to the model. The languages KARL and G-KARL have been developed in MIKE to specify this model.

KARL and G-KARL provide the following advantages:

- Both provide modelling primitives on a high level of abstraction independent of implementation aspects. KARL and G-KARL distinguish several types of knowledge and define different language primitives for them.
- KARL is a *formal* specification language. Thus the language primitives get a defined meaning which allows a precise and unequivocal description. The semantics of KARL has been defined based on the semantics of predicate logic and dynamic logic which have been combined to define a declarative semantics for the entire language. Because a graphical representation in G-KARL may be mapped to KARL, G-KARL is a formal language, too.
- KARL is an *operational* knowledge specification language. Based on a constructive semantics which corresponds to the declarative semantics an interpreter has been implemented. Due to the mapping of G-KARL to KARL, G-KARL is executable, too.

KARL is only partially representable by graphical means. Especially the functional view to the model of expertise is only in parts graphically representable. Other parts of this functional view must be expressed in a logical language. In various applications of MIKE it has been shown that graphical representations are preferred by humans to linear notations and are thus better suited for the communication with domain specialists and users.

Especially for those parts which must be described up to now in the linear notation of KARL this paper presents a new language. Thus using G-KARL the entire model of expertise is representable by graphical means which allows to improve the integration of domain specialists and users into the modeling process of requirements engineering and specification. G-KARL allows to animate the running model which supports the validation of the model beyond a pure execution of the model.

The paper is structured as follows. In section 2 we give a brief overview of the KARL model of expertise. In section 3 we describe the different views established by this model and their graphical representations. Section 4 describes the graphical language and its semantics used to specify sufficient and necessary conditions and elementary inference actions. It also shows how this graphical representation may be animated to support the validation process. Section 5 briefly outlines the current state of and our future plans for the implementation. Related work is discussed in section 6. Finally we give a conclusion.

2 The KARL Model of Expertise

G-KARL and KARL are based on a strong conceptual model: the KARL model of expertise [FAS97], [Fen95]. This model distinguishes three types of knowledge at three different layers which also reflects the distinction of knowledge necessary for the problem-solving method and knowledge necessary for the domain at hand:

- *Domain knowledge* at the *domain layer* consists of static knowledge about the application domain. It consists of domain specific static knowledge about terminology, relationships and facts. Additionally it contains domain specific heuristics which are used by the problem-solving process in order to reduce the search effort. This layer is one part of the static view to the model of expertise.
- *Inference knowledge* at the *inference layer* specifies the *inferences* that can be made using the domain knowledge, and the *knowledge roles*. They model the input and output of the inference actions. 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 and thus describes the functional view to the model. The roles and the inference actions are specified in a *problem-solving-method-specific terminology* which is independent of the domain-specific terminology. The *views* and *terminators* specify the relationship between the knowledge used at the inference layer and the domain-specific knowledge. Views perform a transformation of the domain-specific terminology at the domain layer to the problem-solving-method-specific terminology at the inference layer. Terminators reformulate the solutions of the problem-solving method in domain-specific terms.
- *Dynamic control knowledge* at the *control layer* is used to specify *control* over the execution of the inference actions of the inference layer. It determines the sequence in which inference actions are activated. This layer establishes the dynamic view to the model.

Inference and control knowledge are specific for the task which has to be solved, but they are independent of the concrete domain. As a consequence they may be represented in a generic way using a problem-solving-method-specific terminology. This enables two different kinds of reuse: the domain knowledge may partially be reused for another task, and the problem-solving method may be reused within another domain. Using the MIKE-approach the construction of a library of problem-solving methods [AFS96] has been started.

3 The Graphical Representation of the model of expertise

In the following we describe how the KARL model of expertise may be described graphically.

In figure 1 the model of expertise is sketched in its graphical representation in G-KARL and partially in its linear notation in KARL 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 non-smokers in the same room“, “the boss must have a central, large single-room“, etc. This task was used to compare different knowledge engineering approaches [Lin94].

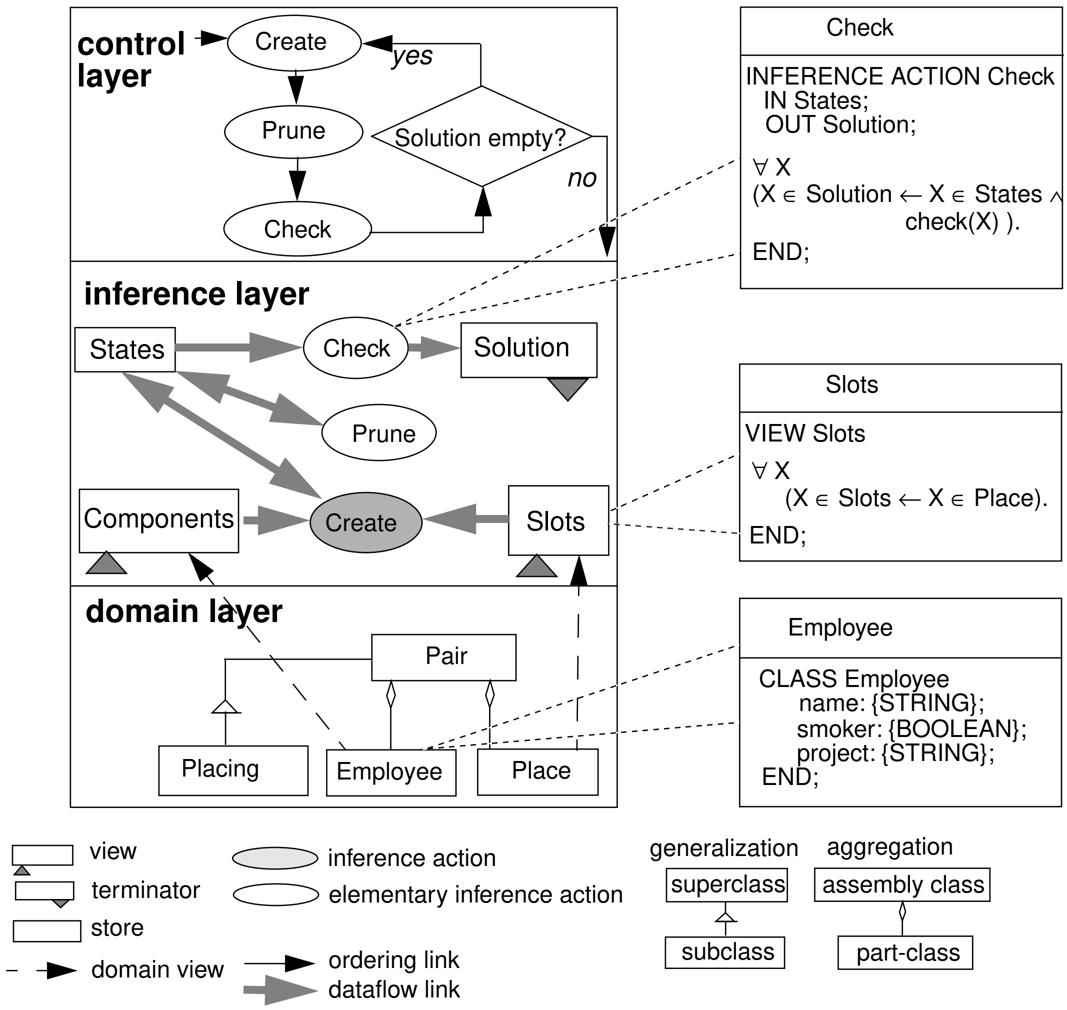


Figure 1. The model of expertise for the Sisyphus office assignment task.

3.1 Static View

Static knowledge at the domain layer and at the inference layer consists of a hierarchy of classes with element and class attributes. Element attributes describe properties of the elements (instances) of the class. Class attributes describe properties of the class itself. Attributes may be single-valued, i.e. their value is one element of the pre-defined value classes INTEGER, REAL, STRING, BOOLEAN. Attributes may also be set-valued or list-valued, i.e. the value of an attribute may be a set of values or an ordered list of values. Classes are related to other classes by the generalization relation. The generalization relation includes inheritance of element attributes from the superclass to the subclass. Multiple inheritance is supported in KARL. Every instance of a class is also an instance of the superclass of this class. Instances of classes have individual values for the element attributes of those classes they are an element of. Elements may be related to other elements by two kinds of relationships: aggregation and association. Aggregation means that elements are parts of other elements. Associations model arbitrary relationships. Associations may have attributes. The graphical notation in G-KARL for static knowledge has been adopted from the OMT notation [RBP+91]. In figure 1, a pair (an instance of class *Pair*) is an aggregation of an employee and a place, i.e. an employee and a place are part of a pair. The class *Placing* is a specialization of the class *Pair*.

Elements of classes are denoted either by instance diagrams or by tables. Instances of relations are denoted by tables. Each relation is given by the identifiers of the related objects filled in the role fields and the values of the attributes of the relation.

In addition to the description of static knowledge KARL allows to declaratively describe necessary and sufficient conditions at the domain layer using the logical language L-KARL which is a sublanguage of KARL. For instance the sufficient condition that two employees (X is an element of class *Employee* and Y is an element of class *Employee*) fit together (*fit_together*(*emp1*: X, *emp2*: Y)) if both are smoker or both are non-smoker (the attribute *smoker* has the same value for X and Y) and both do not work in the same project may be expressed in L-KARL as follows:

$$\begin{aligned}&fit\_together(emp1{:}\ X,\ emp2{:}\ Y) \leftarrow\\ &\quad X \in Employee \wedge X[smoker{:}\ Z,\ project{:}\ P] \wedge\\ &\quad Y \in Employee \wedge Y[smoker{:}\ Z,\ project{:}\ Q] \wedge P \neq Q.\end{aligned}$$

Up to now no graphical representation has been provided in MIKE for such conditions. Though L-KARL provides a high level of abstraction experience has shown that exactly these specifications are hard to understand and are therefore often modeled faulty.

3.2 Functional View

The inference layer is very similar to the functional view to a system in Structured Analysis [You89]. Stores are passive containers which contain input data for inference actions or collect the output of inference actions. So data flow diagrams are used in G-KARL to represent the functional view. The structure of the data within the stores is described using the graphical means mentioned above for static data. Figure 1 shows at the inference layer that e.g. *Employee* and *Place* are input for the inference action *Create*. In order to support the modeling of complex models the data flow diagrams may be refined hierarchically, i.e. an inference action may be refined to an entire data flow diagram etc.

In extension to data flow diagrams views and terminators which transform domain-specific data into problem-solving-method-specific data and vice versa have been introduced in the KARL model of expertise. Views and terminators are specified using L-KARL formulae.

The input-/output-behaviour of elementary inference actions as well as the views and terminators are described declaratively using the logical language L-KARL. For instance in figure 1 the inference action *Check* is represented by one rule which expresses that whenever an element of class *States* does not violate the given constraints (predicate *check*) then it is an element of the class *Solution*. So the functional view to the KARL model of expertise is described entirely formally.

3.3 Dynamic View

At the control layer the sequence is specified in which the inference actions at the inference layer are activated. To represent this control knowledge we use program flow diagrams [DIN66001] in G-KARL. They consist of tasks and decision nodes. In the same way as data flow diagrams may be refined a task may be refined to an entire diagram. The hierarchy at the control layer must correspond to the hierarchy at the inference layer, i.e. every task corresponds to an inference action at the inference layer.

The diagrams at the control layer have to be structured [BöJ66]. Structured diagrams may easily be mapped to the linear notation of KARL for the control layer which provides sequence, loops, and branches. Tests in branches and loops are expressed by necessary conditions in L-KARL.

4 The Graphical Representation of L-KARL formulae

The graphical representation mentioned in section 3 has already been introduced for KARL in a slightly modified way [FAS97]. The main difference is the use of the OMT notation for static knowledge at the domain and at the inference layer.

The elementary inference actions, the views, and terminators at the inference layer as well as sufficient and necessary conditions at the domain layer are represented in the KARL model of expertise using formulae in L-KARL. Up to now no graphical means have been provided for these formulae. So in order to be able to understand a model of expertise these formulae have to be understood as well. In the following section a graphical language which is a partial language of G-KARL is introduced for these formulae. In general such a graphical representation is better to understand and is thus better suited for the communication with the domain specialist or user and allows a modeling which is less faulty. Because these representations are mappable to L-KARL formulae they have a defined semantics and they are executable.

4.1 Sufficient conditions

Subsequently we introduce graphical primitives to describe sufficient conditions at the domain layer and to describe views, terminators, and elementary inference actions. An elementary inference action for instance is considered as a flow of single objects from the input stores to the output stores. This flow may be manipulated within such an inference action. For instance objects with special properties may be filtered out, new objects may be created with properties dependent on other objects etc. A similar flow metaphor has already been used in [Shn92] for describing formulae in propositional logic.

The different parts which manipulate the flow of objects are shown in figure 2

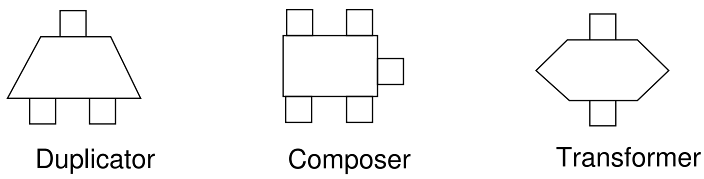


Figure 2. Graphical primitives for the object flow.

These parts are connected via pipes in which the objects flow. The effects of these parts are described in the following:

- The *Duplicator* has one input and *n* output pipes. Every object which flows into the duplicator is duplicated *n*-1-times and every duplicate leaves the duplicator using one output pipe.

- The *Composer* has n input pipes, n output pipes at the bottom and one output pipe at the right. It produces all n-tuples of the objects where each n-tuple contains one object of every input pipe which meet the defined requirements. These requirements are defined using attribute values and/or associations and aggregations. The n-tuples leave the Composer using the right output pipe. An object which entered the Composer at an input pipe and which was not included into an n-tuple (because it does not fit to n-1 other objects taking the requirements into account) leaves the Composer at the output pipe opposite to the input pipe. To decide this the Composer waits for all input objects. So a Composer may also be used for negation: all objects which do not fit to n-1 other objects leave the Composer by the corresponding output pipe.
  If the Composer has only one input pipe it works like a filter. Those objects which meet the specified requirements leave it by the right output pipe. The other ones leave it by the bottom output pipe.
- The *Transformer* allows to create new objects, new associations, or new aggregations from those objects (or tuples of objects) which flow into the Transformer.

A class name which marks an input flow of the diagram specifies that instances of this class flow within this pipe. A quoted class name specifies that the class and all its subclasses itself flow within this pipe. If a Transformer produces new objects the structure of the objects of the output flow is also defined by a class name or a quoted class name. If a Transformer produces a relationship like an association or an aggregation the name of the relationship specifies the output flow. Object flows may be merged if the objects (or n-tuples of objects) have the same structure.

Figure 3 shows a small example. We want to specify the binary association *fit_together*.

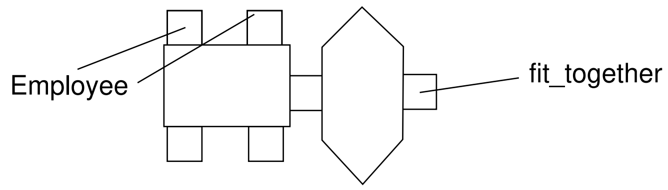


Figure 3. A sufficient condition for the association *fit_together*.

Each instance of this association relates two employees. Two employees fit together if they are both smokers or both non-smokers and if they do not work in the same project. The Composer produces all pairs of employees which meet these requirements. The Transformer then specifies that the resulting pairs of employees are instances of the binary association *fit_together*.

4.2 Conditions

For the denotation of conditions for Composers the instance diagrams are extended by instance diagrams for relationships. The additional instance diagrams use the same representation as used for relationships except that all lines are dotted.

Conditions for single values are expressed within these instance diagrams similar to the graphical data base language QBE [Zlo75] using a textual form:

- Constants are entered directly into the instance diagrams.

- Equality of single values may be expressed by value examples (variables).
- In order to compare values, constants and value examples may be combined using the binary relations: $<$, $\leq$, $\neq$.
- Constants and value examples may be combined with functions.

For instance we want to find a pair of employees who are both teachers, both have swimming as hobby and the first earns more than twice than the second. This condition is represented in G-KARL as shown in figure 4.

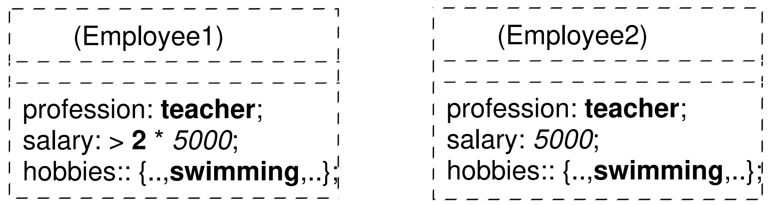


Figure 4. Conditions using value examples (in italics) and constants (in bold).

Conditions for set-, and list-valued attributes are also expressed graphically using edges labeled by relation symbols:

- Set-valued attributes may be compared using the binary relations: $=$, $\subset$, $\subseteq$, $\not\subset$, $\neq$.
- List-valued attributes may be compared using the binary relations: $=$, $\neq$.

Conditions between objects using relationships like aggregation and association are graphically represented using symbols for the different relationships.

Similar to the Composer conditions the assignments for the Transformer are specified graphically:

- New objects produced by the Transformer are shaded; new relationships are depicted by fat lines.
- For resulting new objects or new instances of relationships the attributes and identifiers may be described using constants, functions and value examples (variables).
- Entire lists or sets may be assigned using arrows.
- For resulting relations it may be specified which object has which role in the relation.

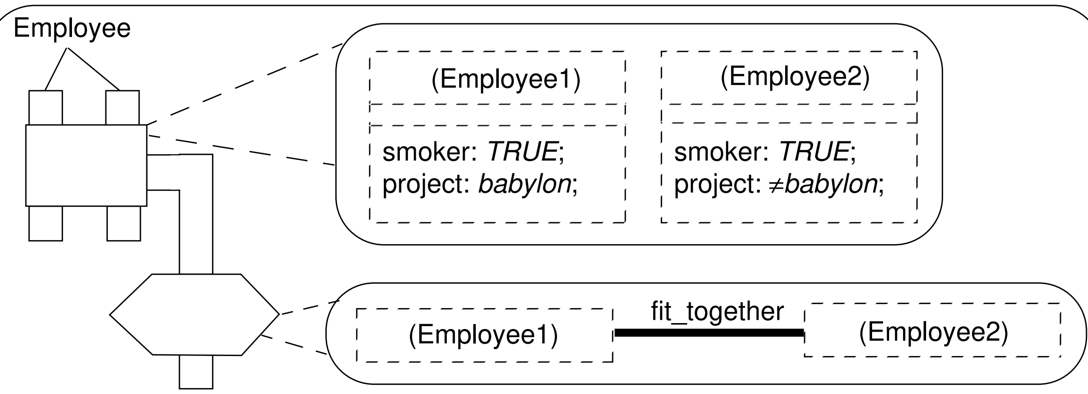


Figure 5. Graphical representation of the Composer condition and the Transformer of the sufficient condition in figure 3. The employee objects are numbered because the resulting 2-tuples of the composer contain two instances of the class Employee

Figure 5 shows the completed graphical representation of the sufficient condition in figure 3. Two employees are combined to a 2-tuple within the Composer if they have the same values for the attribute *smoker* and different values for attribute *project*. The equality is expressed by the same example values (variables) for both attributes. The Transformer then defines for each 2-tuple of employees the new instance of the association *fit_together*.

Objects and object tuples may flow back to an earlier part of the diagram. So recursive expressions like the definition of a transitive closure of a binary association are specifiable.

4.3 Necessary conditions

To express necessary conditions pipes within the diagrams may be marked either by an arrow or by a cross indicating that at least one object (object tuple) must flow through or no object (object tuple) may flow through, respectively. For instance the constraint that a place may only be assigned once is represented as shown in figure 6. The Composer produces 2-tuples of *Placing* objects where the *place* attribute has the same value for both. The second Composer acts like a filter. It lets only those 2-tuples of *Placing* objects flow through which have both the same identifier *placing1*. If one tuple leaves the second Composer at the bottom output pipe (the objects do not have the same identifier, but they have the same value for the attribute *place*) the necessary condition is violated. Necessary conditions at the domain layer are used during modeling to guard the restrictions.

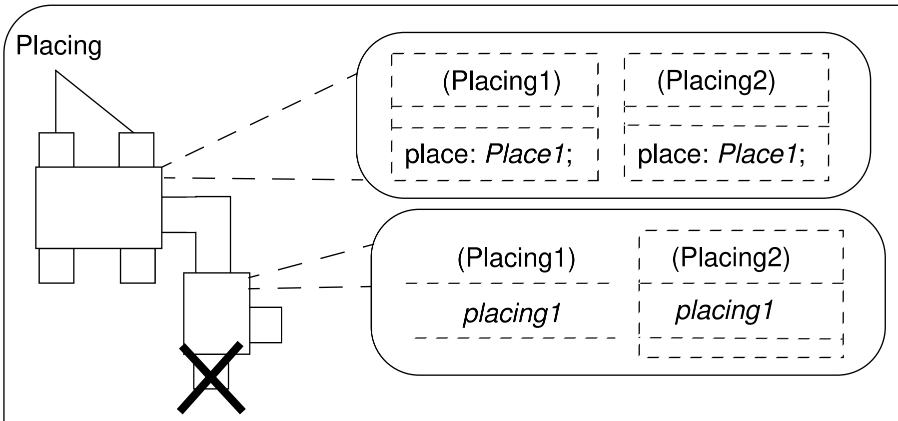
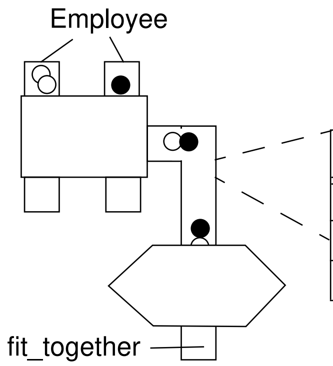


Figure 6 Graphical representation of a necessary condition. *Place1* is a value example (variable) for the value of the attribute *place*. *Placing1* is a value example for the identifier of the objects.

4.4 Validation by Animation

KARL is an executable language. Because every graphical representation of a sufficient and necessary condition or a specification of a view, terminator, or elementary inference action may easily be mapped to a set of L-KARL-rules, these specifications are executable too. Running KARL models of expertise with test cases has been shown to be the most important possibility to validate this model. Up to now it was not possible to observe the execution of parts of an elementary inference action. Only the input before the execution and the output after the execution of such an inference action has been examinable. By exploiting the introduced graphical representations the complete model of expertise may be animated which allows to look into an elementary inference action during execution and thus supports modeling and debugging of single elementary inference actions.

During animation flowing objects are represented as circles. Objects from different classes are distinguished by different colours. n-tuples of objects are represented by n objects glued together which flow through the pipes. The observation of the object flow gives an impression of how many objects and tuples of objects are produced by the different elements. For debugging purposes the set of all objects or tuples of objects which have ever flown through a pipe may be examined at every time during animation. Figure 7 shows a snapshot of an animation.The pairs of employees which have ever flown through the right pipe are visualized using tables.



(Employee1)				**(Employee2)**			
id	smoker	project	...	id	smoker	project	...
hans	no	P12	...	angy	no	P11	...
tom	yes	P01	...	pet	yes	P15	...
...	...	...	...	...	...	...	...

Figure 7. Animation of the sufficient condition.

4.5 Semantics and Expressive Power

Every graphical representation in G-KARL may be mapped to a set of rules in L-KARL. L-KARL is a language based on predicate logic that has been enriched by additional modeling primitives stemming from object oriented languages. For L-KARL a semantics based on the semantics of first-order logics has been defined [FAS97], [Fen95].

This transformation may be briefly sketched as following:

- For every input pipe i of the diagram in which objects of class C flow the rule
  $p_i(X) \leftarrow X \in C$ is introduced.
- For every pipe j in which tuples of n objects flow an n-ary predicate p_j is introduced.
- For every Duplicator with input pipe i and the output pipes $j1,...,jn$ the following set of rules is introduced:
  $P_{jk} \leftarrow P_i$, for $1 \leq k \leq n$,
  where the atom P_i uses the m-ary predicate p_i which has been assigned to the input pipe, and the P_{jk} use the m-ary predicates for the output pipes.
- The condition of a Composer c is expressed in an L-KARL formula F_c using the values of the attributes of the input objects and using aggregations and associations which relate these objects to other objects.
  Now for every used bottom output pipe j a rule is introduced:
  $Q_{cj} \leftarrow P_{c1} \wedge .. \wedge P_{ck} \wedge \neg F_c$,
  where the atom Q_{cj} uses the predicate which has been introduced for the output pipe and the P_{ci} use the predicates which have been introduced for the input pipes.
  For the right output pipe the following rule is introduced:

$R_c \leftarrow P_{c1} \wedge .. \wedge P_{ck} \wedge F_c$,

where R_c uses the predicate introduced for the right output pipe.

- A Transformer t is mapped to a rule

  $Q_t \wedge F_{t1} \leftarrow P_t \wedge F_{t2}$.

  where the atoms P_t and Q_t use the predicates introduced for the input pipe and for the output pipe, respectively. Formula F_{t2} accesses values and relationships of the input objects and F_{t1} represents a formula about the new values of the objects and the new created relationships.
- If two pipes i and j are merged to pipe k the following two rules are introduced:

  $P_k \leftarrow P_i$,

  $P_k \leftarrow P_j$,

  where the P's use the predicates for the different pipes.
- For every output flow of a diagram a rule is created which maps the internal representations into the representations for the corresponding representation used within the static view.

By this way our example in figure 5 may be mapped to the following set of rules in L-KARL (free variables in a rule are assumed to be universally quantified at the head of the rule):

For the input pipes the following rules are introduced:

$p_1(X) \leftarrow X \in Employee.$

$p_2(X) \leftarrow X \in Employee.$

Between Composer and Transformer two-tuples of objects flow. So a binary predicate p_3 is introduced for that pipe.

The Composer is mapped to the following rule:

$p_3(X,Y) \leftarrow p_1(X) \wedge p_2(Y) \wedge F.$

F is the formula which expresses the condition of the Composer:

$X[smoker\colon T, project\colon P] \wedge Y[smoker\colon T, project\colon Q] \wedge P \neq Q.$

The Transformer is mapped to the rule:

$fit\_together(X,Y) \leftarrow p_3(X,Y).$

By this way each graphical representation in G-KARL gets a defined semantics which is given by the semantics of the set of L-KARL rules it is mapped to.

Every L-KARL rule may in turn be mapped to a graphical representation in G-KARL. So G-KARL provides the same expressive power as L-KARL. L-KARL enriches Horn logics with negation (normal programs) by additional modeling primitives. So the expressive power of L-KARL equals the expressive power of normal programs [Llo87].

5 Tools and Applications

The hypermedia based MIKE-tool [Neu93] supports the entire development process in MIKE by different editors. Among others it provides graphical editors for the static, functional, and dynamic view to the model of expertise. Up to now for elementary inference actions and sufficient and necessary conditions L-KARL formulae must be entered in their textual linear form. The MIKE-tool integrates an interpreter for KARL [Ang93] and provides debugging facilities for the model of expertise like breakpoints etc. The debugger operates on the graphical representation of the functional view. Intermediate results in stores and terminators may be examined in their linear L-KARL notation.

The reimplementation of the MIKE-tool which is currently under way will realize the graphic representation for the different views presented in this paper. We will supplement (not substitute) the linear notation of elementary inference actions and sufficient and necessary conditions in L-KARL by the graphical language presented here. The debugger will be extended by the animation facilities for elementary inference actions described in section 4.4.

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 [FLS+95]. It resulted in a prototypical kbs providing support in selecting the most appropriate technique for pulling down a building. This case study exemplified that using the graphical primitives for the static, functional and dynamic view (not including the graphical primitives for L-KARL formulae) the model of expertise can be built up and evaluated in close cooperation with the domain expert.

As a second problem for comparing different knowledge engineering approaches an elevator configuration task has been posed [IJHCS96]. 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 and the available parts of G-KARL for representing the knowledge on an abstract level [PFL+96]. For the validation of the resulting complex model of expertise it was even indispensable to have a running prototype.

6 Related Work

Within the KADS framework [SWB93] two different languages have been developed: the language CML [SWA+94] to describe models of expertise semi-formally on a conceptual level and the language ML^2 [vHB92] to describe models of expertise formally. Guidelines and tool support are available to semi-automatically transform a CML-description into an ML^2-description [vHA96]. A subset of ML^2 is executable by a simulator [THR91] in order to support the validation by testing. In contrast to that KARL and G-KARL combine these three different aspects in one language: specification on a conceptual level, formal specification, and operational specification. Thus the knowledge engineer has to learn only one language and there are no consistency problems between different representations of the same knowledge. To support the development in KADS a number of tools are available. They allow to represent the model of expertise of KADS in a similar way as our current MIKE-tool. Similar to that the formal parts in ML^2 have to be entered in its textual form. So there are no graphical means to support the development of these parts.

Within the VITAL [JoM93] project also two languages have been developed: the language VITAL-KML [JoM93] represents the conceptual model semi-formally and the language $K_{BS}SF$ [DMW93] represents the functional design model formally, which corresponds to our model of expertise. $K_{BS}SF$ is operational and thus allows to validate the resulting model by testing. So again the knowledge engineer has to learn two different languages. VITAL presents a lot of tools to support the development. They also provide similar graphical editors like the current MIKE-tool. Similar to the tools for KADS the formal languages are supported in their textual form only.

A lot of work has been done in the data base community to develop languages to graphically express queries. SNAP [BrH86] and CANDID [ScT92] are based on semantic data models, VILD [LSN89] is a graphical language for an object oriented data model. Both, semantic data models and object oriented data models provide a rich set of primitives similar to the modeling primitives for static aspects in G-KARL. For none of the above mentioned languages a defined formal semantics is available. None of them are embedded into a global conceptual model which describes functional and dynamic aspects of an entire system.

The INCOME approach [OSS94] for developing information systems uses an integration of high-level Petri nets and semantic data model concepts for conceptual modeling. The transitions of the Petri nets are inscribed by a logical language for NF^2 relations. For this language a graphical representation is available which resembles the tabular notation in QBE [Zlo75]. So a conceptual model representing the static, functional, and dynamic view is available and transitions are described in a high level language partially graphically. However the transitions are not expressed using the graphical concepts of the semantic data model. Instead a transformation process into an NF^2 representation has to take place.

In contrast to the KIDS/VDM approach [Led96] which translates a VDM-SL specification into an equivalent REFINE specification the executability of a G-KARL-specification is achieved by mapping G-KARL primitives to KARL primitives. Because G-KARL is just another representation of the same language (KARL) the problem of handling two different specification languages is avoided. In that way, the use of the MIKE-tool only requires the familiarity with one formal specification language.

7 Conclusion

In this paper we have introduced the language G-KARL. G-KARL provides the following features:

- G-KARL is based on a conceptual model which allows to describe the static, functional, and dynamic view to a kbs on a high level of abstraction. This model represents all functional requirements for the system which has to be developed, but it abstracts from all design decisions and from all implementational detail.
- KARL and G-KARL allow a smooth transition from a model where only the top level control flow is described and all inference actions are described declaratively to a model where most components are described in an algorithmic manner and where the elementary inference actions only cover a small functionality. Though in the first case the elementary inference actions cover a large functionality they may be described in an abstract and compact manner using L-KARL formulae or using G-KARL diagrams.
- G-KARL is a formal and executable language. Thus every language primitive gets its defined semantics. In the future such a formal description enables automatic consistency checks and allows the verification of the final system against this model. The executability of G-KARL allows to validate the model by testing and debugging.
- G-KARL allows to represent the conceptual model entirely graphically. This representation combines textual representations and graph-based representations

which enables a knowledge engineer (i) to express simple things with simple means (by textual representations) and (ii) to represent complex structures in their entirety (by graphical means). Using the diagrams of the static view to express conditions allows the system modeler to visually recognize the structure of data instead of remembering this structure which again eases the modeling and validation process (cf. [Shn92]).

- G-KARL and L-KARL may be used interchangeably to specify inference actions, views, terminators, and necessary and sufficient conditions. So the preferred representation may be used and it will be possible to switch between both representations.

The combination of these properties enables the conceptual model to be built by a prototyping process. With our current MIKE-tool (which does not yet support the representation of all formal parts graphically) we have already evaluated in some external projects that the use of graphical means strongly supports the communication of the knowledge engineer with the user and the domain specialist. This allows to integrate both into the development process which has been shown to be indispensable to arrive at applicable kbs. With the introduction of graphical means also for formal parts of our models this necessary interaction may be further supported. After the implementation of tools for these additional graphical representations this will be thoroughly evaluated.

References

[AFS96] J. Angele, D. Fensel und R. Studer: Domain and Task Modeling in MIKE. In A. Sutcliffe, D. Benyon, F. van Assche (eds.): *Proceedings der IFIP WG 8.1/13.2 Joint Working Conference, Domain Knowledge for Interactive System Design*, Geneva, May 1996, Chapman & Hall, 1996.

[Ang93] J. Angele: 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).

[BöJ66] C. Böhm, G. Jacopini: Flow Diagrams, Turing Machines And Languages With Only Two Formation Rules. In *Communications of the ACM*, Vol. 9, Nr. 5, Mai 1966, 366-371.

[BrH86] D. Bryce, R. Hull: SNAP: a graphics based schema manager. In *Proc. of the Int. Conf. on Data Engineering*, Los Angeles, 1986, 151-163.

[Cla89] W.J. Clancey: The Knowledge Level Reinterpreted: Modeling How Systems Interact. In: *Machine Learning 4*, 1989, 285-291.

[DIN66001] German Industry Norm (DIN), Nr. 66001.

[DMW93] J. Domingue, E. Motta, S. Watt: The emerging VITAL workbench. In: N. Aussenac, G. Boy, B. Gaines, M. Linsert, J.-G. Ganascia, Y. Kodratoff (eds.): *Knowledge Acquisition for Knowledge-Based Systems, Proceedings of the 7th European Workshop (EKAW'93),* Toulouse, France, September 6-10, 1993), Lecture Notes in AI no 723, Springer-Verlag, Berlin, 1993.

[FAS97] D. Fensel, J. Angele, and R. Studer: The Knowledge Acquisition and Representation Language KARL. To appear in: *IEEE Transactions on Knowledge and Data Engineering*, 1997.

[Fen95] D. Fensel: *The Knowledge Acquisition and Representation Language KARL.* Kluwer Academic Publisher, Boston, 1995.

[FeS96] D. Fensel and R. Straatman: Problem Solving Methods: Making Assumptions for Efficiency Reasons. In: Shadbolt, N. et al. (eds.): *Proceedings of the 9th European Knowledge Acquisition Workshop (EKAW'96)*, Nottingham, United Kingdom, May 1996, Springer LNAI, Vol. 1076, 1996, 17 - 32.

[FLS+95] W. Fichtner, D. Landes, Th. Spengler, M. Ruch, O. Rentz, and R. Studer: 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).

[IJHCS96] IJHCS: *International Journal of Human-Computer Studies (IJHCS)*, 44, special issue, 1996.

[Jon90] C.B. Jones: *Systematic Software Development Using VDM*, 2nd ed., Prentice Hall, 1990.

[JoM93] W. Jonker, E. Motta: Functional design activity using KBSSF and VITAL-OCML. ESPRIT-II Project 5365: VITAL: A methodology-based workbench for KBS Life cycle support. DD221.3, 1993.

[Led96] Y. Ledru: Using KIDS as a Tool Support for VDM. In *Proceedings of International Conference on Software Engineering ICSE 1996*, Berlin, Germany, 1996.

[Lin94] M. Linster (ed.): Sisyphus: Models of Problem Solving. In: *Int. Journal of Human-Computer Studies 40, 2* , special issue, (February 1994).

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

[LSN89] M.-K. Leong, S. Sam, D. Narasimhalu: Towards a visual language for a mulit-media database system. In E. Knuth, L.M. Wegner (eds.), *Proceedings Visual Database Systems*, North-Holland, 1989, 465-495.

[Neb96] B. Nebel: Artificial Intelligence: A Computational Perspective. In: G. Brewka (ed.): *Principles of Knowledge Representation, Studies in Logic, Language and Information*, CSLI Publications, Stanford, 1996, 237-266.

[Neu93] S. Neubert: Model Construction in MIKE (Model-Based and Incremental Knowledge Engineering). In N. Aussenac et al. (eds.), *Knowledge Acquisition for Knowledge-Based Systems, Proceedings of the 7th European Workshop (EKAW´93),* Toulouse, France, September 6-10, 1993), Lecture Notes in AI no 723, Springer-Verlag, Berlin, 1993.

[OSS94] A. Oberweis, G. Scherrer, and W. Stucky: INCOME/STAR: Methodology and Tools for the Development of Distributed Information Systems. In: *Information Systems 19*, 8, 1994, 643-660.

[PFL+96] K. Poeck, D. Fensel, D. Landes, and J. Angele: Combining KARL And CRLM For Designing Vertical Transportation Systems. In: *International Journal of Human-Computer Studies (IJHCS)*, 44, 1996, 435-467.

[RBP+91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy und W. Lorensen: *Object-Oriented Modeling and Design*. Prentice Hall, 1991.

[ScT92] M. Schneider, C. Trepied: Extensions for the graphical query language CANDID. In E. Knuth, L.M. Wegner (eds.), *Proc. Visual Database Systems*, North-Holland, 1992, 185-199.

[ShG92] 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, Springer, 1992, 208-220.

[Shn92] B. Shneiderman. *Designing the User Interface*. Addison-Wesley Publ. Co., 1992.

[Spi92] J.M. Spivey: *The Z Notation. A Reference Manual.* 2nd ed., Prentice Hall, New York, 1992.

[SWA+94] A.Th. Schreiber, B. Wielinga, H. Akkermans, W. van de Velde, and A. Anjewierden: CML: The CommonKADS Conceptual Modeling Language. In: Steels et al. (eds): *A Future of Knowledge Acquisition, Proc. 8th European Knowledge Acquisition Workshop (EKAW´94)*, Hoegaarden, Lecture Notes in Artificial Intelligence 867, Springer-Verlag, 1994.

[SWB93] G. Schreiber, B. Wielinga, and J. Breuker (eds.): *KADS. A Principled Approach to Knowledge-Based System Development*, Knowledge-Based Systems, vol 11, Academic Press, London, 1993.

[THR91] A.T. Teije, F. V. Harmelen, M. Reinders: Si(ML)^2: a prototype interpreter for a subset of (ML)^2. ESPRIT Project P5248 KADS-II, report KADS-II/T1.2/TR/UvA/005/1.0, University of Amsterdam, 1991.

[vHA96] F. van Harmelen and M. Aben: Structure-preserving specification languages for knowledge-based systems. In: *International Journal of Human-Computer Studies (IJHCS)*, 44, 1996, 187-212.

[vHB92] F. van Harmelen and J. Balder: (ML)^2: A Formal Language for KADS Conceptual Models. In: *Knowledge Acquisition 4*, 1, 1992.

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

[Zlo75] M.M. Zloof: Query by Example. In *Proceedings of AFIPS Conference*, 1975, NCC, 431-437.