KRISYS - A MULTI-LAYERED PROTOTYPE KBMS
SUPPORTING KNOWLEDGE INDEPENDENCE

Nelson Mendonca Mattos

University Kaiserslautern, Department of Computer Science,
P.O. Box 3049 , D-6750 Kaiserslautern, West Germany
e-mail: mattos@uklirb.uucp

Abstract

aper discusses architectural issues of Knowledge Base Man- ent Systems and describes the architecture of KRISYS, a sys- hose goal is the effective and efficient management of large, knowledge bases. Focal points are primarily the design de- s and the system's features: knowledge independence, object- ation, mechanisms for knowledge organization, data-driven tation, in heritance mechanisms, reasoning facilities, etc. lly, some of these issues which we have been combining to a KBMS context, are similar to approaches developed in dif- isolated projects. Instead of giving a detailed comparison of proaches and those of other projects, we show in this paper ey can be architecturally combined to build realistic KBMS.

ords: AI architectures, Knowledge Base Management Sys- Database support for Knowledge-Based Systems

1. Introduction

hnology has produced a variety of knowledge-based systems anging from simple expert systems to complex natural lan- understanding systems. When used for large-scale applica- KS are faced with problems of managing very large volumes wledge: virtual memory sizes are not large enough to store owledge to be handled, and operations on knowledge (i.e. in- e) are computationally intolerable when knowledge bases re maintained on secondary storage devices.

problems show that the applicability of KS is limited, since riate systems for the efficient knowledge management do ist. Approaches combining KS with traditional Database ement Systems (DBMS) for this purpose have failed for sev- asons (see [1] for a description of the deficiencies of DB sup- r KS).

lution to KS problem is to develop a new generation of sys- aimed at the efficient management of large, shared KB. Anal- to DBMS, these systems are called Knowledge Base Manage- Systems (KBMS) [2].

bservation has motivated our research efforts to identify im- t design issues regarding an architecture for KBMS. Our in- ations have shown that these issues are strongly influenced ee classes of requirements. Firstly, KBMS should satisfy the ements of their applications (i.e. KS or end-users). Secondly, ust support the needs of the KB-designer, who plays a very tant supporting role in this context. And thirdly, some imple- tion aspects in terms of data structures and algorithms be taken into account, in order to manage the knowledge ef- ly. Thus, KBMS should provide features obtained from three ent points of view. So, we believe that KBMS should be archi- lly divided in three different layers: implementation layer, ering layer and application layer, which respectively support f the above classes of requirements.

We feel that the above points are involved with a number of quite new ideas, which we incorporated in a multi-layered prototypical KBMS. Clearly, some of these ideas, which we combined to apply in a KBMS context, are related to approaches developed in different projects [3,4]. However, these projects handle one or more of these ideas in an isolated manner, not taking into account the practical use of them in KBMS. In this paper, we describe our prototypical architecture for KBMS, called KRISYS, showing above all how these and other approaches can be combined in order to build realistic KBMS. To motivate our ideas, we first review the results of our investigation: the KBMS architectural issues that arise when addressing the above mentioned three classes of requirements.

2. Architectural Issues

Traditionally, knowledge representation systems or KS components responsible for the knowledge management have been designed just to support what we called the engineering layer. Since the manner of knowledge organization and access is visible at the external interface of those systems, KS possess information about the knowledge organization and retrieval possibilities built into their logic (i.e. embedded in their programs). Any modification in the knowledge structures therefore requires program modifications. Changing the kind of repre sentational framework used, for example from frames to semantic networks, would be even impossible, since this would mean throwing the entire KS away and implementing a new one.

However, with complex KS and very large KB on the horizon, this dependence on the framework supported by a knowledge representation system promises to be very problematic. *Knowledge indepe dence* as an analogy to data independence seems to be the key answer to this problem. Knowledge representation systems must be characterized not in terms of the representational framework they use, but functionally, in terms of what they know about the KS domain.

This idea of abstraction aimed at the independence of knowledge motivated us to introduce another layer (the application layer) over the engineering layer in the architecture of KBMS. At the object-abstraction interface, KS can, therefore, work independently from the specification of the representational framework supported by the engineering layer. In fact, KS are not interested in things like the complexity of frame structures, the variety of links in a semantic network, the properties of inheritance mechanisms or the power of reasoning facilities. They are really interested in what they can ask or tell the KBMS about the knowledge of their domain, which is stored in the KB managed by the KBMS. This motivated us also to believe that the KS interface is the one supported by the application layer (object-abstraction interface). The interface for the KB-designer is, however, a different one. He (in opposition to the KS) is concerned with the aspects of the representational framework. He decides, whether a specific information is to be represented as a way knowledge is represented and manipulated).

2.1 The Application Layer

d at knowledge independence, the application layer should nowledge functionally, in terms of only two basic types of opns supported at this level: one to enable the KS to ask the S questions to be answered on the basis of the knowledge kept KB and another one to permit the KS to tell the KBMS new edge to be maintained in the KB. Therefore, at the object-abon interface the KB can be compared with an abstract data hat interacts with the end users or KS through a set of "ask" ell" operations. Thus, the way the knowledge is captured or ed is hidden from the KS. Because of this, no distinction can de between knowledge that is extensionally stored and that has to be implied (intensionally). Whether just simple re- capabilities or inference of some kind is required to answer tion, is not to be decided by the KS, but in the layer.

unctional view of the KB has also been introduced in KRYP- 3], where ask and tell operations are split in two different ines: a terminological Tbox, and an assertional Abox. However, ues supported by KRYPTON do not meet the idea of knowlndependence. Before making changes in a KB a KRYPTON as to decide for example, whether the KB`s theory of the should imply these changes (tell operation at the Abox) or the vocabulary should include them (tell operation in the Tbox). er words, he has to decide whether these changes should be ionally represented or extensionally. We argue that such a on should not be made by the user but in the application layer, t the way knowledge is represented can be hidden from the n`s users. We believe that this is the only possible way to view as an abstract data type. Furthermore, KRYPTON`s interface t been developed for a KBMS context and consequently is neilexible nor powerful enough to meet the requirements of S applications. (For example, ask statements are restricted to questions).

uery language provided at the object-abstraction interface naturally permit adequate and flexible ways to select the apon's domain knowledge and to cause changes in the KB cording to the many ways knowledge can be accessed and ed. Furthermore, it must be set-oriented. This permits the KS ain several "pieces-of-knowledge" with just one operation, re communications overhead between KS and KBMS and offer- enormous optimization potential for the lower layers. Addi-

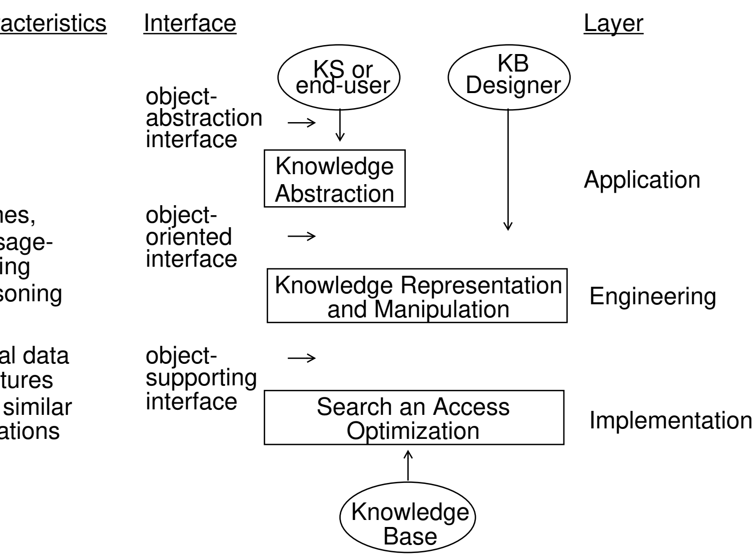


Figure 1: Overall architecture of a KBMS

2.2 The Engineering Layer

The engineering layer focusses on a KBMS from the point of view of the KB-designer. Here the KBMS is seen in terms of how the knowledge of an application can be represented, organized, and manipulated. In other words, this layer deals with descriptive, organizational and operational aspects of a knowledge representation model supported for the KB-designer at the engineering layer's interface (object-oriented interface). Concerning the descriptive principles:

- Entities in the application's world should correspond to exactly one object in the model.
- Each object should be composed of attributes expressing the properties of the entities.
- Attributes are multiple-valued and should be further described (e.g. data type of the attribute values, etc.).
- Objects are described by object types (i.e. classes), which should also be objects of the model.
- Classes characterize prototypical instances, i.e., properties described in the classes might be contradicted by particular instances.
- Objects can be instances of several classes.
- Relationships between entities should be expressed as properties of objects [5].

Concerning the organizational principles, the abstraction concepts [6,7,8], extensively used in AI knowledge representation systems, are the best candidates:

- Objects are to be grouped into classes (classification), which, in turn, should be organized into class hierarchies providing the means for the overall organization of the KB (generalization).
- Inheritance must be supported, in order to eliminate unnecessary definitions. Multiple inheritance should also be allowed, and the notion of inheritance should be that of default [9,6].
- Similar element objects should be organized into set-element hierarchies (association).
- Aggregation of component objects should be supported, and might be applied recursively, so that another organizational dimension of the KB can be defined.

Operational issues should primarily support object orientation. This will allow the attributes of the objects to store either extensional or intensional data, which are accessed in a uniform way (i.e. by sending object messages). Moreover, object orientation makes it possible to extend the functionality of the layer interface very easily, since it permits the KB-designer to define his own functions, which as attributes of objects can also be directly addressed at the interface by message passing. Reasoning facilities should also be provided. The knowledge representation model should offer already implemented general purpose reasoning functions, not prohibiting, however, that these functions may be substituted by special ones developed by the designer. Mechanisms for checking integrity constraints are to be provided too. To define constraints this layer should enable the KB-designer to define procedures that are linked to attributes of objects and are automatically activated before or after such attributes are accessed or updated. These procedures attached to data will then perform the necessary checks, and the corresponding actions to keep the KB in a consistent state.

s supply to the engineering and application layers. For this , many of the issues here are related to traditional DB prob-

equisite to identify the specific requirements to be supported s the investigation of the behavior of KS running on second- orage environments. In order to pursue this investigation, we dapted some available KS to work on such environments [10]. oservations may be summarized as follows:

e accesses made at the implementation layer interface (object- pporting interface) are mainly to tiny granules refering to in- idual attributes of objects rather then to objects as a whole.
e attributes' access frequencies differ very much. Dynamic at- utes (i.e., the specific knowledge of a consultation) are access- with very high frequencies; static attributes (i.e., the expert owledge of the KS) on the other hand with very low frequen- s.
e dynamic knowledge can be kept temporarily, since it only ex- sses information of a particular consultation.
multi-user environments, the KBMS must maintain as many sions as the number of users working with the KB for the dy- nic knowledge. These versions are then accessed individually, that synchronization should only be controlled for the static owledge.
each phase of the problem solving process, accesses concen- e on just some objects of the KB. These objects build a context ich contains the knowledge needed to infer the specific goal of t phase. The context needed in each phase can be established some information that the KS deduces during the preceding ases (i.e., dynamic knowledge), so that it is possible to deter- ne the next context needed at the end of a phase. This enables ynamic preplanning of the accesses to the KB that should be d at this layer for optimization purposes.
rage structures for the KB should give more priority to the op- ization of the retrieval operations, since modifications in the uctures of the KB (e.g. changing object types) are very seldom.

3. The KRISYS-Prototype

3.1 Overview of the System Architecture

YS is architecturally divided in three different modules, which the aspects of the previously discussed layers (Figure 1). The ation layer corresponds to the Query-Processing Manager e 2). Here, knowledge independence is supported by the YS Object-Abstraction Language (KOALA), which character- e object-abstraction interface. To implement the engineering we choose a Frame-System (KRISYS Frame System KFS) , in turn, supports the interface to the KB-Designer and to uery-Processing Manager, i.e., the object-oriented interface. mplementation layer requirements are fulfilled by the Work- emory and Context Manager (WMCM) and by a DBMS Ker- VMCM embodies the nearby application locality concept, i.e., eservation of the locality of the application's object processing

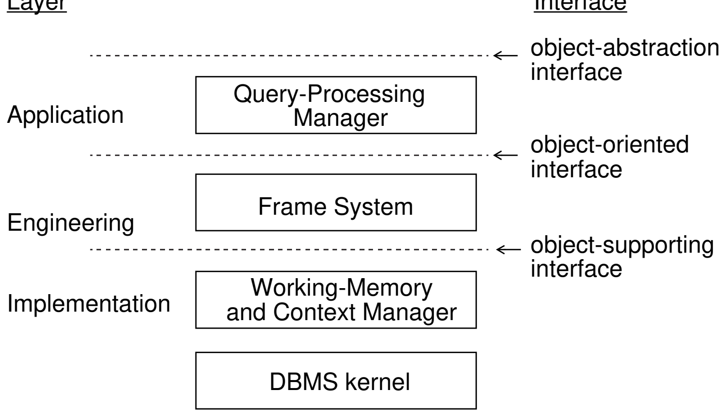


Figure 2: The KRISYS Architecture

3.2 Query-Processing Manager

The application layer of KRISYS can be better characterized by the description of its object-abstraction interface, the language KOALA, which defines all interactions between a user or application and the system, which occur with two types of operations:

ASK - to ask KRISYS, whether an "expression" is true, retrieving the qualified knowledge, and
TELL - to tell KRISYS, that an "expression" is true, changing the KB appropriately

The Ask-Statement

An Ask-statement contains two parts; a projection and a selection (i.e. qualification) part.

ASK[<projection>][<selection>]

The last one specifies the expression to be proved, which is structured in accordance to the first-order predicate calculus. The selection is evaluated following the model-theoretic approach [11]. In other words, ask-statements are evaluated as true or false with respect to the KB. During evaluation, the Closed World Assumption (CWA) [12], the Unique Name Assumption (UNA), and the Domain Closure Assumption (DCA) are taken as granted. For example, the query (ASK (IS_INSTANCE tweety penguin)) would be true only if there exists both tweety and penguin as objects of the KB related by the abstraction concept of classification.

The logical formulas are built with predicates (e.g. IS_INSTANCE in the above example). These use either constants, variables or functions as their terms. The last one can, in turn, use variables or constants as terms again, in order to address the objects of the KB. KOALA supports functions and predicates to be applied to objects (schemas), attributes (slots), attribute values (slot values) and attribute descriptions (aspects). Some important object predicates are those used to express the abstraction concepts of classification/ instantiation (IS_ INSTANCE) generalization/specialization (IS_ SUBCLASS) and association (IS_ELEMENT, IS_SUBSET). The next example shows a query which asks whether or not elephant is a specialization of mammal and an element of the set animals_from_africa:

```
(ASK (AND (IS_SUBCLASS elephant mammal)
          (IS_ELEMENT elephant animals_from_africa)))
```

To manipulate attributes, KOALA offers functions to select all slots of an object, only the inherited ones, the inherited ones from a particular object, the slots defined for an object (i.e. not inherited), its instance_slots, its own_slots, its standard_slots or its user-defined_slots. These functions have as their result a set of slots, which are to be interpreted as the components of the specified object (abstraction concept of aggregation). If someone would like to ask,

(IS_IN nutriment (INHERITED_SLOTS camel
(SUPERS mammal))))

ons about slot values can be expressed by a lot of predicates, use functions to access the slot value itself (e.g. SLOTVALUE name> <schema_name>). The equality predicate can be used or example, to express that a slot value is equal to a specific

(EQUAL milk (SLOTVALUE nutriment camel)))

rical comparison predicates are also provided. Furthermore, other functions, it is possible to evaluate arithmetic expres-such as, sum, average, minimum and maximum value of a set nbers or even to count the number of elements of a set.

ute description predicates allow the application to formulate ons about slot value restrictions, cardinality specifications, or example, the query

```
(IS_POSSIBLE_VALUE fly (ASPECTVAL
                          POSSIBLE_VALUE
                   form_of_locomotion mammal)))
```

s, whether the value fly is allowed in the slot of_locomotion of the object mammal.

ientation is achieved by using variables in the logical formu-ariables express either objects, attributes, attribute values or te descriptions of the KB. During evaluation they are instan-with all values, which satisfy the logical formulas. Because of CA, formulas, which contain variables that cannot be instan-are assigned 'false'. To ask whether there exists any bird annot fly, one can use variables to create the following ques-

```
(EXIST ?X (AND (IS_INSTANCE ?X bird)
               (NOT(IS_IN fly (SLOTVALUE
                 form_of_locomotion ?X))))))
```

bles can also be used to construct questions involving more ne object (roughly analogous to a database join). The user can uild queries combining variables that represent different (i.e. objects, attributes, attribute values or attribute descrip-For example, to know whether there is any object in the KB as stored a specific value in any of its attributes, someone l pose the question

```
(EXIST ?X ?Y (AND (IS_SCHEMA ?X)
                  (IS_IN ?Y (ALL_SLOTS ?X))
                  (EQUAL specific_value
                         (SLOTVALUE ?Y ?X)))))
```

ently, the user is also interested to see the instances, which his logical formulas (e.g. to identify the dogs descending from me parents), in addition to the boolean value of these formu-e can specify this in the projection part of the ask-statement. ample, if someone would like to know which mammals can he should ask

```
(?X) ( AND  (IS_INSTANCE ?X mammal)
            (IS_IN swim (SLOTVALUE
                     form_of_locomotion ?X))))
```

rojection also enables the user to express exactly which parts accessed information (objects, attributes etc.) he would like to his can be achieved by using specific projection clauses or by ning different variables (e.g. variables that represent objects hose that represent attributes). For example, if someone to see the dogs stored in a KB together with the value of the ey have in common with cats, one would ask KRISYS the fol-g query:

Recursive queries may be expressed using the PATH predicate, with which a special class of recursion equations, called generalized transitive closure (GTC) [13], can be specified. Unfortunately, due to space limitations it is not possible to discuss these aspects here. (For a detailed description see [1]).

The Tell-Statement

The tell-statement takes a sentence and asserts that it is true. The effect is to change the KB into one whose contents imply that sentence. Naturally, it can also occur that the contents of the KB can already imply the sentence asserted. In this case, no changes will be made, since they are not required. For example, if the KS asserts that penguin is a subclass of bird, expressing "(TELL (IS_DIRECT_SUBCLASS penguin bird))" two situations can occur. Either the KB already contains this information, requiring no changes, or the KB does not contain it and consequently changes must be made. Here, many things can happen. If neither bird nor penguin exist as objects in the KB, both will be created and related to each other by the abstraction concept of generalization. If only one of them exists, the other one will then be created and related to the first one as specified above. And if both of them exist only the generalization relationship will be built.

TELL<assertion>[WITH<selection>]

The sentences to be asserted are specified in the assertion part of the tell-statement. An assertion is syntactically similar to the selection part (see ask-statement), however, much more restrictive. It is, for example, not possible to specify formulas combined with logical connectors, in order to avoid ambiguities (note, that if someone asserts p v q, it is impossible to know whether p, q or both are true). Nevertheless, several assertions can be specified, which will then be interpreted independently, i.e., KRISYS will make each one true:

```
(TELL   (IS_DIRECT_SUBCLASS penguin bird)
        (IS_DIRECT_INSTANCE tweety penguin)
        (EQUAL frankfurt_zoo (SLOTVALUE address tweety))
```

If the same assertions should be made valid for several objects, a KS can specify this requirement in just one tell-statement by using variables to represent these objects. The variables would then be instantiated during the evaluation of the selection part, which has the same syntax and works just like the selection of the ask-statement. After this evaluation, the assertions will then be applied for each instance of the variables that satisfy the selection. The following example shows a query expressing the assertion that every animal that lives in water (here represented as elements of the set water-animal) can swim:

```
(TELL  (IS_IN swim (SLOTVALUE locomotion ?X))
       WHERE  (IS_ELEMENT ?X water_animal)).
```

Another example for the use of variables shows the following query which expresses that every owner of a dog of the KB should also be an object of the KB:

```
(TELL  (IS_SCHEMA ?X)
       WHERE (FOR ALL ?Y WITH (IS_INSTANCE ?Y dog)
                  (IS_IN ?X (SLOTVALUE owner ?Y)))).
```

Summarizing this chapter, we can say that the object-abstraction interface is completely specified by the above two operations. "Tell" takes a sentence and asserts that it is true, changing the KB into one whose contents imply the assertion. "Ask" takes a sentence and checks on the basis of the current contents of the KB whether it is true or not. Schematically, they can be described as

TELL : KB x sentence ==> KB **Sentence is true !**
ASK : KB x sentence ==> T/F **Is sentence true ?**

dden from the user. The KB is characterized as an abstract ype, specified only by these two operations rather than by a n implementation structure.

3.3 The Frame-System

oose a frame system [4] to implement our engineering layer, se we believe that it offers the necessary framework to repre- he descriptive, organizational and operational aspects of the n knowledge of any KS.

t-Oriented Representation

KRISYS Frame-System (KFS) the three aspects mentioned are incorporated in its basic concept: the schema. A schema symbolic representation of a real world entity. It is composed hema name and of a set of attributes. Attributes can be of two

ts, used to describe the descriptive and organizational aspects he schema, and

thods, used to describe its operational aspects.

fore, KFS could be named object oriented. It allows both at- ve characteristics and procedural properties of the real world s to be integrated into a schema.

ns in KFS occur by sending messages to the objects, which can, , communicate with other objects by message passing. This pt supports the important principle of data abstraction. That rder to request the methods of some objects to be performed, umptions have to be made about the implementation and in- representation of the objects (i.e. slots and methods).

and methods have the same structure. Both possess name, type and a schema-name, that specifies where the attribute fined (origin). The type indicates whether an attribute repre- haracteristics of the schema itself, or of its instances. Char- tics of a schema are described by the types OWNSLOT and METHOD, whereas the instance ones by INSTANCESLOT NSTANCEMETHOD, respectively.

iated with the attributes there can be aspects, which are used cribe the attribute more exactly. Five aspects are predefined S for slots: possible-values, cardinality, comment, default and ns. Figure 3 shows the structure of the schema elephant. Since ributes shown in the example describe characteristics of ele- itself, they have type OWNSLOT or OWNMETHOD (the ng of the slots INSTANCE-OF and ELEMENT-OF will be de- d later).

```
phant
NSTANCE_OF          mammal              OWNSLOT  GLOBAL
LEMENT_OF           savanna-animal      OWNSLOT  GLOBAL

rm-of-locomotion    (walk swim)         OWNSLOT  animal
  POSSIBLE_VALUES      (walk fly swim)
  CARDINALITY          [1,3]
  DEFAULT              (walk)

abitat              (savanna)           OWNSLOT  animal
  POSSIBLE_VALUES   INSTANCE_OF habitat-class)
  CARDINALITY       [O, ∞]
  COMMENT           (This slot contains the places, where this
                      animal  is found in the nature)
at                  (<code>)            OWNMETHOD  animal
  PARAMETER         (food)
```

Figure 3: Example Definition of the Schema Elephant

slots, that are automatically activated when these slots are access- ed. This concept of data-oriented computation is very useful to rep- resent intensional data or to check complex integrity constraints. For example, if someone wants to represent some information whose value changes with respect to other data (e.g. the exact age of a person, which changes every day), he needs a mechanism that generates the extensional value of this information automatically, each time this information is accessed. This can be realized in KFS by using demons.

Demons are stored in a KFS schema (like any other), which has, however, the predefined schema DEMON as its most superclass. As an instance of DEMON this schema inherits the method attributes GET, PUT, ADD and RETRACT, where the code for the respective demons are stored (represented by schema D1 in Figure 4).

The linkage between slot and corresponding demons is done by storing in the demons aspect of the slot (S1) the schema name of the respective demons (D1). Whenever an access to the slot is made KFS checks if there is a demons aspect defined in this slot, activating it by sending a message to the schema specified there, demanding the evaluation of the corresponding demon. That is, the method specified in GET will be invoked if a get-access has been issued to KFS, the method PUT is invoked by issuing a put-access etc. By not storing the demon-code directly in the aspect of the slot, as is done in many systems, KFS allows many slots to use the same demons without having to introduce redundancy in the representation. KFS additionally allows the KB-designer to specify when the demon should be activated (i.e. before or after the access to the slot). This flexibility allows the use of demons for many different purposes. Demons applied to check integrity constraints should be activated before updating the value of a slot, whereas those used to trigger actions when particular slots are accessed should be activated after the access itself.

```
schema-x
   :
   S1  <S1-value> OWNSLOT <S1-origin>
       POSSIBLE-VALUES (...)
       :
       DEMONS D1
   :

D1
   INSTANCE_OF   (DEMON)
   GET          <get-code)        OWNMETHOD  DEMON
   PUT          <put-code>        OWNMETHOD  DEMON
   ADD          <add-code>        OWNMETHOD  DEMON
   RETRACT      <retract-code>    OWNMETHOD  DEMON
```

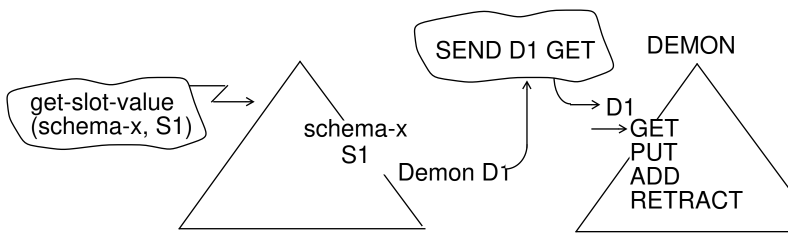


Figure 4: Activation of Demons

Knowledge Organization

In order to enable the KB-designer to structure the KB adequately, KFS supports various abstraction concepts. These are represented as relationships between objects specified by predefined slots, which are contained within every schema. The abstraction concept of generalization is represented by the slots SUBCLASS_OF and HAS_SUBCLASSES. INSTANCE_OF and HAS_INSTANCES are used to specify the classification relationship. The association conny special slot for its representation.

ating objects to another, hierarchies are built, where a partic- chema can play the role of a class (i.e. object type), an in- , a set and/or an element. Since a schema is able to reflect an ce and a class as well, we need two different slot and method to know whether a slot represents characteristics of itself NSLOT and OWNMETHOD) or of its instances (INSTANC- T and INSTANCEMETHOD).

tance

f the advantages of organizing the schemas into abstraction chies are the built-in reasoning facilities they provide [6]. By g objects with abstraction relationships, special "automatic" ing facilities are supported as part of each modification and al operation. The most usual and important of these reason- cilities is the one built-in the is-a hierarchy i.e. inheritance. In only instanceslots and instancemethods are inherited, since wnslots and ownmethods represent characteristics of their as. Between schemas linked with association relationships no tance occurs. Here, as well as over aggregation relationships, kinds of built-in reasoning facilities are provided. For a de- description of these facilities see [6].

ning

lso supports general reasoning facilities. Rules in KFS are ented as schemas, which are also organized in special hierar- having two predefined schemas as superclasses. One of these as specifies the structure of the rules, i.e. this schema ES) contains some instanceslots (condition, action, etc.) which erited by each rule. In order to construct a rule base, the KB- er has only to store the particular contents of the condition, , etc. in the respective slot of each rule. The other top level a (RULE-SETS) contains methods corresponding to the stan- nference strategies, which are then inherited by each set of

icular rule of KFS can, therefore, be used in both reasoning ons (i.e. forward-reasoning and backward-reasoning) without g to be stored redundantly. The reasoning direction of the are dynamically defined when the KS requests the activation of the inference strategies. Since these are implemented as ds, the activation of reasoning mechanisms is made by send- message to a rule set. This will then activate its corresponding d (forward-reasoning or backward-reasoning) evaluating f its rule elements.

arizing this chapter, we can argue that KFS supports all is- eeded at the engineering layer. Descriptive aspects are satis- y the rich declarative part of our frame model, i.e. the struc- f schemas, slots, slot values and aspects. Organizational as- are supported by the abstraction hierarchies and inheritance. tional requirements are met by allowing the KB-designer to y his own methods (object orientation), by supporting at- procedures (data-oriented computation) and by offering rea- g mechanisms.

3.4 Working-Memory and Context Manager

eady mentioned, the goal of this component of KRISYS is to e a framework for the exploitation of the application's locali- s, therefore, desirable to have a mechanism that enables re- n of DBMS calls and the reduction of the path length when ac- g the objects of the KB, allowing the application to reference s almost directly. This is reached by storing needed objects rarily in a special main memory structure called working- to the stored objects.

The approach used to support locality, is to dynamically extract the context (see 2.3) needed in each phase of the application's problem solving process from the DBS. Therefore, during changes of processing phases the old context would be discarded from the working-memory and the new one loaded into it. Since the application's accesses will then be concentrated on the objects of the loaded context, only a few DBS calls will be made during the next processing phase. Due to the set-oriented specification of the required objects, the DBS can use its optimization potential, reducing I/O and transfer overheads. Furthermore, the tiny granules of the application's accesses will not bring any inefficiency, since the path length when accessing the objects is now very short. (Details about the internal structure and implementation aspects of the working-memory are provided by [1]).

	duration of consultation (CPU-sec)	ratio to main memory approac
main mamory approach	146	1
direct coupling	28105	~ 170
coupling with application buffer	2946	~ 18
WMCM	643	~ 4

Figure 5: Performance Comparison of Different Coupling Approaches between KFS andDBS

The performance of WMCM has been compared with the performance of other coupling approaches between KFS and DBS (Figure 5). In this case we use a main memory-based form (i.e. when the whole KB is stored in main memory) as basis for this comparision. Coupling KFS and DBS directly seems to be the most inefficient alternative, due to the long path length (from application to DB-buffer) by the application's very tiny access granules. This problem is eliminated by using an application buffer, which uses LRU as allocation/deallocation strategy, to keep the most recently used objects. This approach needs, however, to perform very many DBS calls, since individual objects are frequently being extracted from the DBS and stored into it. This is particularly critical after changes of processing phases, when most of the needed objects are not found in the buffer. We solved this problem by a set-oriented fetch, as described above, which accomplishes an efficiency very close to the main memory approach. Indicative performance figures of a particular example as illustrated in Figure 5 are derived in [10].

3.5 DBMS Kernel

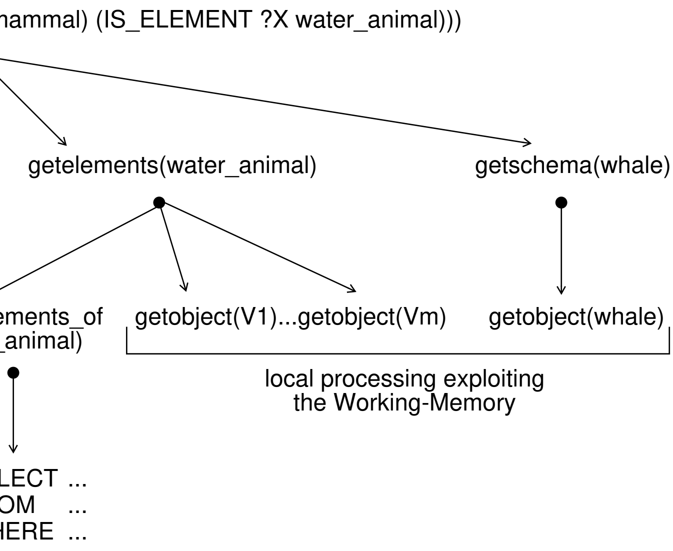
Due to space limitations it is not possible to discuss the aspects of the DBMS kernel and the mapping of frames with it in sufficient depth.

The DBMS kernel architecture chosen for KRISYS [14, 15,16] was developed for the support of the so called non-standard applications. For this reason, our kernel, named PRIMA, offers neutral, yet powerful mechanisms for managing KB of all types of KS efficiently: storage techniques for a variety of object sizes, flexible representation and access techniques, basic integrity features, locking and recovery mechanisms, etc. [17]. PRIMA is a *PR*ototype *I*mp mentation of the *MA*D model. MAD provides dynamic definition and handling of objects, based on direct and symmetric managerocess are found in [18,19, 20].

3.6 Summarizing Example

chapter, we illustrate the control flow through the KRISYS by giving a simplified example of the operations needed to a KS query at each of the system interfaces (we consider the ng-Memory as been empty at the beginning of an evalua- Since a complete description is too space-consuming, we re- ourselves to essential properties expressing the flavor of the al operations.

ee structure of Figure 6 indicates the calling hierarchy need- the evaluation of an ASK-statement (where control opera- tc. are dropped for simplicity reasons). Exploitation of infor- n encountered in the Working-Memory would save (expen- calls to the DBMS kernel interface (with accesses to the S buffer or even to the disks). Perfect locality preservation avoid any kernel calls for repeated object references.



n ASK-statement

presented the design of our prototype, named KRISYS. We have advocated the division of KBMS in three different layers, which respectively support the requirements of the application, the needs of the KB-designer and the requirements of an efficient management of the KB. The main philosophy of the system is the idea of abstraction aimed at the independence of knowledge. At the system interface, knowledge is seen not in terms of the flexible representational framework supported by the engineering layer but functionally, in terms of the application's beliefs about its domain. In order to achieve this functional view of the KB, it is necessary to restrict the communication between application and KBMS to ASK and TELL operations, however, providing:

- multiple ways to select the application`s domain knowledge,
- user defined projections,
- flexible forms to cause changes in the KB,
- set-orientation, and
- recursion.

A second very important philosophy is the effective support of the needs of the KB-designer. Clearly, this philosophy has also been the goal of most existing knowledge representation systems which try to support these needs by focusing on the improvement of the expressiveness (i.e. descriptive aspects) of their representation model. However, in a KBMS context expressiveness is just one of the three aspects to be considered at the engineering layer. Here, organizational and operational aspects are equally as important as descriptive ones. Because of this, an effective support of the needs of the KB-designer is to be achieved by a mixed knowledge representation framework offering:

- object-orientation,
- data-driven computation,
- mechanisms for knowledge organization (i.e. all abstraction concepts),
- inheritance and other built-in facilities, and
- reasoning mechanisms,

all of them uniformly integrated with a very rich descriptive part.

A number of concepts used in the KRISYS implementation pay attention to performance requirements. Most important is the framework provided by the implementation layer for the exploitation of the application locality. A single-user version of KRISYS is now complete. KRISYS is to be considered a research vehicle for a variety of KS. It is intended, therefore, to run as a tool for building KS and as a generic system, aimed at the effective and efficient management of large knowledge bases.

Acknowledgement

I should like to thank T. Härder and B. Mitschang for the helpful comments on the revision of this paper, and H. Neu and I. Littler for preparing the manuscript.

References

[1] Mattos, N.M.: KRISYS - A Multi-layered Prototype KBMS Supporting Knowledge Independence, Research Report ZRI No. 2/88, University of Kaisers lautern, 198

[2] Mylopoulos, J.: On Knowledge Base Management Systems, in: [21], pp. 3-8.

[3] Brachman, R.J., Fikes, R.E., Levesque, H.J.: Krypton: A Functional Approach to Knowledge Representation, in: Computer, Vol. 16, No. 10, 1983, pp. 67-73.

[4] Fikes, R., Kehler, T.: The Role of Frame-based Representation in Rsoning, in: Commications of ACM, Vol. 28, No. 9, September 1985, pp. 904-920.

[5] Borgida, A.: Survey of Conceptual Modeling of Information Systems, in: [21], pp. 461-470.

Schrefl, M., Tjoa, A.M., Wagner, R.R.: Comparison-Criteria for Semantic Data Models, in: Proc. IEEE 1st Int. Conf. on Data Engineering, Los Angeles, 1984, pp. 120-125.

Brodie, M.L., Mylopoulos, J., Schmidt, J.W. (eds.): On Conceptual Modelling (Perspectives from Artificial Intelligence, Databases, and Programming Languages), Topics in Information Systems, Springer, New York, 1984.

Borgida, A., Mylopoulos, J., Wong, H.K.T.: Generalization/Specialization as a Basis for Software Specification, in: [8], pp. 87-114.

Härder, T., Mattos, N., Puppe, P.: On Coupling Database and Expert Systems (in German), in: State of the Art, Vol. 1, No. 3, 1987, pp. 23-34.

Gallaire, H., Minker, J., Nicolas, J.-M.: Logic and Databases: A Deductive Approach, in: ACM Comp. Surveys, Vol. 16, No. 2, June 1984, pp. 153-185.

Reiter, R.: On Closed World Databases, in: Logic and Databases, (eds. Gallaire, H. and Minker, J.), Plenum, New York, pp. 56-76.

Dayal, U., Smith, J.M.: PROBE: A Knowledge-Oriented Database Management System, in: [21], pp. 227-257.

Härder, T., Reuter, A.: Architecture of Database Systems for Non-Standard Applications (in German), in: Proc. of the GI-Conf. on Database Systems for Office, Engineering and Science Environments, 1985, pp. 253-286.

Dadam, P., et al.: A DBMS Prototype to Support Extended NF2-Relations: An Integrated View on Flat Tables and Hierarchies, in: Proc. ACM SIGMOD Conf., Washington, D.C., 1986, pp. 356-367.

Paul, H.-B., Schek, H.-J., Scholl, M.H., Weikum, G., Deppisch, U.: Architecture and Implementation of the Darmstadt Database Kernel System, in: ACM SIGMOD Conf., San Francisco, 1987, pp. 196-207.

Härder, T., Meyer-Wegener, K., Mitschang, B., Sikeler, A.: PRIMA - A DBMS Prototype Supporting Engineering Applications, in: Proc. 13th VLDB Conf., Brighton, 1987, pp. 433-442.

Mitschang, B.: A Molecule-Atom Data Model for Non-Standard Applications - requirements, data-model design, and implementation concepts (in german), PhD Thesis, University of Kaiserslautern, 1988.

Mattos, N.M.: Mapping Frames with the MAD model (in German), Research Report No. 164/86, Univ. Kaiserslautern, 1986.

Härder, T., Mattos, N., Mitschang, B.: Mapping Frames with New Data Models (in German), in: Proc. GWAI'87, Springer, 1987, pp. 396-405.

Brodie, M.L., Mylopoulos, J. (eds.): On Knowledge Base Management Systems, Springer, New York, 1986.