in: ZRI-Bericht 5/91, University Kaiserslautern, 1991

KRISYS - a KBMS Supporting Development and Processing of Knowledge-based Applications in Workstation/Server Environments

N. M. Mattos
University of Kaiserslautern

5/91

December 1991

Abstract

Knowledge Base Management Systems (KBMS) should provide not only efficient and reliable management of knowledge bases (KB) but also means for constructing such KB in a stepwise fashion. In this paper, we present a KBMS prototype tailored to workstation/server environments that can be used flexibly as a tool for dynamically defining KB contents during application development and efficiently employed for KB manipulation during application processing. The support of these contradictory features of flexibility and efficient processing is reflected in the system architecture by means of an appropriate mapping scheme and of mechanisms to guarantee locality of KB accesses.

Contents

1. Introduction

During the last few years, the development of a new generation of systems capable of effectively supporting construction and processing of knowledge-based applications (KS) has emerged as an important research area in the database field. These systems, so-called Knowledge Base Management Systems (KBMS) [BM86,ST89], should provide an efficient and reliable management of large, shared knowledge bases (KB) and the means for building such KB. In KS, the model of problem solving of an application domain is explicitly represented in the KB, rather than being buried somewhere (i.e., appearing only implicitly) in the program's code. As a consequence, the knowledge incorporated into the system can be more easily analyzed by the knowledge engineer as well as potentially more easily extended or modified [CJV83]. In practice, this leads to a KS development that can be practically viewed as a KB construction task [My86]. This development is, in general, supported by a modeling tool (e.g., KEE [FK85,Fi88], LOOPS [BS83,SB86]) with which the knowledge engineer just has to 'describe' the knowledge of the application at hand in order to build the corresponding KS. In other words, KS development is almost restricted to an incremental modeling activity [MM89] involving knowledge acquisition, KB structuring, refinement, and evaluation with feed-back, rather than to the traditional activity of programming [Ha87].

This advantageous KS development characteristic has a tremendous impact on the functionality of KBMS since a new application development paradigm is being established. That is, instead of supporting only an efficient and reliable KB management (in analogy to DBMS), KBMS should also provide means for describing, evaluating, extending, and restructuring the knowledge of an application (just like knowledge engineering tools). They should be used as powerful modeling tools to be adequately employed for KS development as well as robust management systems to efficiently support KS processing.

Following the lines of these issues, we have performed a number of sizable prototype implementations to better evaluate important concepts for designing KBMS. The prime result of these studies was a refined understanding with regard to functionality, cooperation, performance of system components, and their mutual interferences, which we then exploited to derive the overall architecture of our KBMS, called KRISYS [Ma88b,Kr89]. In the following, we cannot repeat all investigation results in detail (see [Ma88a,Ma90,Mi88,MM89]), but only summarize our main findings that have greatly influenced our design and implementation.

To obtain an appropriate framework for KS development, the following features should be observed:

- ***Means for supporting an incremental KS development:*** During KS construction, the knowledge engineer should use the knowledge model as a tool for dynamically defining the contents of the KB. It has to be possible to directly create and modify such KB contents without losing any of the previously existing information. This will allow KB contents to be incrementally specified, thereby supporting knowledge reformulations.
- ***Elimination of the difference between KB schema and KB contents:*** As a consequence of the above requirement, the model should explicitly integrate meta-information (e.g., descriptions of object types or classes) into the KB, thereby eliminating any difference between KB schema and KB contents. A strict separation, as required by DBMS, neither enables an interaction of KB design and manipulation (e.g., for KB validation) nor reflects real world situations, where changes affecting the application model may also occur (e.g., in the case of schema evolution).

In the case of application processing, the following observations have to be carefully considered:

- ***Stability of KB structure during KS processing:*** During KS operation (in contrast to what happens during KS development), modifications in the structures of the KB (e.g., changing object types, adding new types, or integrity constraints) either do not occur or are very seldom. This strong stability should be exploited by a KBMS to choose appropriate storage structures and access paths for the KB, thereby increasing the application performance.
- ***Nearby application locality:*** Locality of reference should be exploited as far as possible; buffering KB contents close to the KS is the only means to guarantee efficient KB accesses. Moreover, the transfer of contents into such buffers should be based on processing characteristics of the KS since locality of accesses strongly depends on the problem solving strategy [St83] being used.

Finally, some mapping-critical issues should be observed to enable efficient KS modeling and processing:

- ***Handling of distinct mapping mechanisms:*** KBMS need, on one hand, a very general and flexible mapping of knowledge structures for the support of a KS modeling process in a stepwise fashion and, on the other hand, a very efficient (and probably not flexible) mapping for supporting KS processing. Hence, they cannot be constructed on the basis of a fixed, predefined mapping mechanism since this will favor either KS modeling or processing, but not both.
- ***Tailoring mapping mechanisms to application characteristics:*** After the modeling process, the structures of the KB of distinct KS differ greatly. Additionally, the kind of KB manipulation performed during the operation of a KS is also very distinct (e.g., rule-based, data-driven, or object-oriented). Therefore, the support of an 'application-oriented' mapping for the processing phase of a KS is essential for providing the required efficiency.

We feel that the above requirements are reflected in a number of quite new issues, which we incorporated in our prototypical KBMS KRISYS. Clearly, some of the ideas which we combined to apply in KRISYS are related to approaches developed in different projects [FK85,HR85]. However, these projects handle one or more of these issues in an isolated manner, not taking into account the practical use of them in the context of KBMS. In this paper, we describe our prototypical architecture for KBMS, showing above all how the concepts of KRISYS can be combined to effectively support development and processing of KS. Since KRISYS has already been running for more than two years, some of the aspects and viewpoints presented here are based on practical experiences obtained through the development of several KS implemented on top of KRISYS [DHMM89,DHMS90]. In the following, we first sketch an overview of our system architecture in chapter 2. After this, chapters 3, 4, 5, and 6 respectively describe how to employ KRISYS to construct KS, our mapping approach, the means for supporting application processing, and a delegation of tasks to the lower layers of the system.

2. Overview of the System Architecture

Conceptual KBMS architecture

From a conceptual point of view, there are three orthogonal ways of looking at KBMS [BL86,Ma88b]. These dominant and distinct viewpoints determine distinct facets of KBMS, corresponding in some sense to the three different aspects playing important roles when constructing and employing KS: the needs of their applications (i.e., knowledge manipulation means for solving problems), knowledge engineering support (i.e., modeling con-

cepts for KB construction), and suitable resources and implementation aspects (i.e., maintenance mechanisms for efficiently coping with knowledge storage and retrieval). Thus, KBMS should incorporate different features in order to be able to fulfil the requirements involved with these three different viewpoints. The support of these three classes of requirements leads to a natural division of KBMS architecture in three different layers, which we denote application, engineering, and implementation, where the corresponding features are especially considered (figure 1a). As such, the application layer looks at the information content of KB, the engineering layer considers the modeling aspects of an application expressed as KB contents, and the implementation layer views KB in terms of data structures and access algorithms.

Considering Workstation/Server Environments

The above observations, however, only show the general guidelines for the realization of KBMS. In order to derive the overall architecture of such systems, we also have to consider the essential aspects of their run-time requirements and their processing environment. In general, KS run on powerful workstations equipped with sufficient processing capability, main memory, and private disk space, which are typically dedicated to single users or knowledge engineers and may provide special functions (e.g., suitable graphic interfaces) for them. The workstations have access to a central server component, whose task is to provide access to centralized information (i.e., the KB) and to control its shared use. Therefore, from a hardware point of view, KBMS architectures should fit into a workstation/server environment with decentralized and autonomous processors. Among others, failure isolation, extensibility, and scalability of the entire system can be considered as key advantages of such architectures.

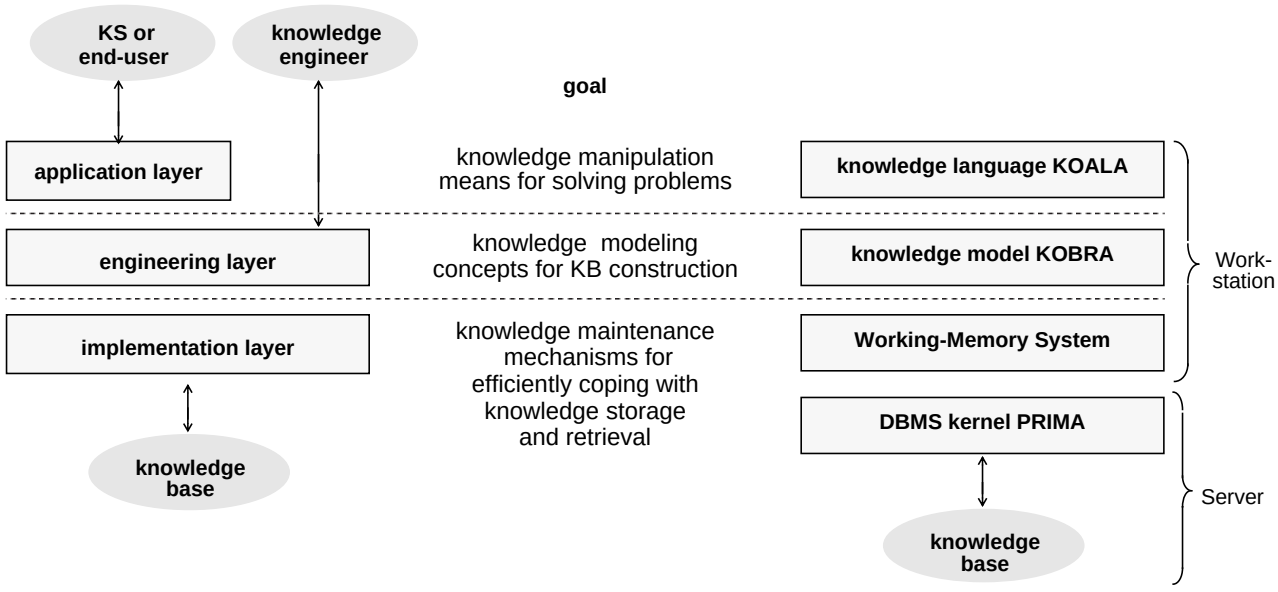
As a consequence of this environment, all questions related to 'coupling' system components should be carefully considered when designing KBMS. The closer the 'coupling' between server and workstation, the more they will mutually influence one another, provoking communication delays, network congestion, and failure dependence. Thus, the only reasonable approach is the use of loosely coupled system components relying on interfaces that minimize communication traffic and KB accesses. This has to be accompanied by the support of high degree of locality of reference to be maintained by the system (e.g., by using main memory data structures) on the workstation side because this will largely influence the performance of the complex problem solving strategies [St83] and reasoning processes manipulating the KB. Since KS are mostly used interactively, responsiveness is a prerequisite to be accepted by the users.

One step towards achieving the above goals is taken by integrating an object buffer into the workstation component to support locality of references [HHMM88]. An object buffer can be exploited to temporarily maintain relevant KB contents so that manipulations on such contents can be accumulated in it and propagation to the server can be deferred until the end of a modeling or processing step. However, the use of an object buffer is not sufficient for the minimization of workstation/server communication. Further steps should be taken by means of specifying an appropriate interface between server and workstation, thereby giving rise to the question as to where to place the 'borderline' between these components when considering the described conceptual architecture of KBMS. Such an interface plays a crucial role in the design of a KBMS since the functionality provided at the server side influences the amount of processing that will be performed at both server and workstation. Introduction of more semantics into the server (e.g., by placing engineering and application layers) means enhancing its functionality, thereby shifting a lot of processing from the workstation to the server (because the operations performed by the server become more powerful). Less semantics on the server side means less functionality, tending to leave more processing at the workstation.

At first glance, the enhanced server approach seems to be the most appropriate for a system architecture since it allows a concentration of features in one single component shared by several workstations. However, it neglects the advantages of such environments by ignoring the use of workstations as the key for solving many (hardware) aspects of failure isolation and scalability. Therefore, the introduction of more semantics into the server hinders the exploitation of available computing resources and processor 'power' by the workstations, increases communication overhead, and causes some kind of dependency of the workstations on the server. For these reasons, we argue that the 'borderline' between these two components has to be placed in the implementation layer so that most of the semantics provided by the KBMS (available only in the engineering and application layers) remains on the workstation side. Hence, the server provides only a 'neutral' interface for all management requests of the other layers located at the workstation. (Note that this interface plays a key role in determining to what extent our separation concerns may be satisfied, i.e., knowledge supply, communication overhead, etc. This will be however discussed later.)

Architecture of KRISYS

Reflecting the considerations discussed so far, KRISYS is architecturally divided in four different components (figure 1b) [Ma88b]. The application layer corresponds to the module implementing the knowledge language KOALA, which is responsible for the realization of the application interface of KRISYS [DLM90]. The terms of the mixed knowledge representation framework provided for the KB designer are defined by the KOBRA knowledge model, corresponding to the engineering layer. KOBRA offers flexible constructs for describing the application domain, such as mechanisms for integrating behavior into the application model, abstraction concepts for organizational purposes, general reasoning facilities for performing deductions, and several constructs for maintaining the semantic integrity of the KB [MM89,De90]. In order to fit into the above described processing environment, the implementation layer of KRISYS is divided in two modules. The DBMS kernel PRIMA [HMMS87,Hä88] located at the server concentrates on efficient and reliable KB management, whereas the Working-Memory System (WMS), placed on workstation side, embodies the 'nearby application locality' concept. It manages a special main memory structure, called working-memory, acting as an object buffer, in which



a) conceptual KBMS architecture b) overall system architecture of KRISYS

Figure 1: The KRISYS approach towards KBMS

requested KB contents are temporarily stored. The replacement of these objects is carried out in accordance with processing characteristics of the KS so that calls to the PRIMA kernel, accesses to secondary storage, as well as transfer overhead between server and workstations are substantially reduced.

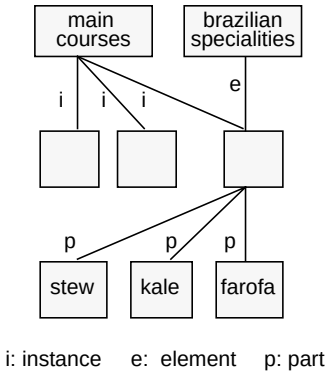
Due to the space limitations of this paper, we will not describe all aspects of KRISYS in sufficient depth, but concentrate on important issues for understanding the support of KS development and processing. So, we will first give an overview of the KS construction process by means of KRISYS in the next chapter, thereby sketching the concepts underlying the KOBRA model. Then, we present the scheme used to map knowledge structures to the DBMS kernel. Following this, the use of the WMS for increasing performance during KS processing is described. Finally, we discuss consequences of a possible delegation of functions to the server, thereby reinforcing the architectural arguments introduced in this chapter.

3. Modeling a KS with KRISYS

When describing the real world, people involuntarily apply some *abstractions* (classification, generalization, association, and aggregation) [BMW84,Br81,SS77] to organize their knowledge in some desired form. Abstraction permits one to suppress specific details of particular objects, emphasizing those pertinent to the problem or the view of information at hand. It is therefore the fundamental tool for organizing knowledge and one of the most important constructs to be supported by any data or knowledge model. For this reason, the model of KRISYS provides an integrated view of such abstraction concepts [Ma88a] to appropriately support the incremental KS development. Abstraction mechanisms, in contrast to existing DB techniques, advocate a modeling process in a stepwise fashion. That is, at each step, only some details of the problem are considered so that less relevant details are neglected until some later step [BMW84]. In such environments, the modeler may take advantage of the reasoning facilities provided by the abstraction concepts [Ma88a,RHMD87] in order to facilitate his task by exploiting the system to make deductions about objects as well as to guarantee the structural and semantic integrity of the KB. In the following, we discuss such exploitation of abstraction mechanisms for KS development. For this purpose, let us first take a brief look at the knowledge model of KRISYS. We will only introduce concepts necessary for a better understanding of our considerations, instead of providing a complete description of the KOBRA model (see [Kr89,MM89] for details). We will illustrate our examples by means of a KS that advises guests of a first class restaurant in Rio de Janeiro as to how they may combine dishes and wines.

3.1 An Overview of the KOBRA Knowledge Model

The KOBRA knowledge model [Kr89,MM89] integrates descriptive, organizational, and operational knowledge into one basic concept, called *schema* (not to be confused with a DB-schema), which represents every entity of the modeled world (e.g., Brazil's national dish 'feijoada' in figure 2). A schema (others call it frame, unit, or object) is uniquely identified by a name (i.e., object-identifier), and contains a set of attributes to describe its characteristics. Attributes can be of two kinds: *slots* are used for the representation of properties of a schema (e.g., price) and of its relationships to other schemas (e.g., recommended-wines); *methods* are used for expressing behavioral aspects of this entity (e.g., order-dish). In order to characterize an object in more detail, all kinds of attributes can be further described by *aspects* (possible-values and cardinality specification, default-value, user-defined aspects, etc.).




```
(feijoada
    (instance-of            (main courses)
        (cardinality            [1 1]))
    (element-of             (brazilian specialities)
        (possible-values        (or  (instance-of main-courses)
                                     (instance-of drinks))))
    (price                  (45)
        (possible-values        (and (integer) (interval <10 150>)))
        (unit                   (US$))
    (recommended-wines      (beaujoulais)
        (possible-values        (instance-of red-wines))
        (default                (bordeaux)))
    (order-dish             (begin .... end)
        (comment                (this operation orders a 'feijoada')))
    (ingredients            (stew kale farofa)
        (possible-values        (or  (instance-of meats or vegetables)
                                     (one-of (rice potatoes farofa))))
        (cardinality            [1 3])
        (diametric-reference    (is-ingredient-of))))
```

Figure 2: Example description of the schema 'feijoada'

For object structuring, KOBRA supports the abstraction concepts of classification, generalization, association, and aggregation. Their semantics are guaranteed by *built-in reasoning facilities* provided by the system and used as the basis for drawing particular conclusions about objects [Ma88a,RHMD87]. These concepts are incorporated into the model by means of special, system-controlled attributes (and as such can also be further specified by aspects). That is, they are seen as special, predefined relationships between objects, defining the overall organization of a KB as a kind of complex network of objects. Hence, each schema can be related to other objects by means of any abstraction concept. Thus, classification/generalization as well as association and aggregation form a directed acyclic graph (sometimes called lattice) rooted in a system-defined schema. Since each schema can be a node in each of these graphs, the KB can be seen as the superposition of three graphs. The same object can therefore represent a class with respect to one object and a set or even an instance with respect to another. 'Feijoada', for example, represents an instance of main courses, an element of brazilian specialities, and an aggregate composed of a stew, kale, and farofa (i.e., manioc flour toasted in butter). Thus, KOBRA supports an *integrated view of KB objects*, i.e., since there are no separate representations for sets, classes, instances, or complex objects, the difference between data and meta-data, which is usually apparent in existing data models, is eliminated in KOBRA so that meta-information is integrated into the KB [MM89].

As already mentioned, behavioral aspects of an entity are represented by means of methods. They express operations, which may change the object's properties, defining a kind of interface to other objects. KOBRA provides two further concepts for the specification of operational knowledge: *demons and general reasoning facilities*. Demons (similar to triggers) are procedures attached to attributes, which are automatically activated when attributes are manipulated. The general reasoning facility is provided by *rules* which can be evaluated for forward or backward reasoning. Important in this context is that both mechanisms are represented by means of the same concepts described above. As such, demons are user-defined schemas that are instances of a system-defined schema. Rules are instances of another predefined object from which they inherit attributes for the specification of their contents. So, they can be flexibly grouped into rule sets for organizational purposes, may receive further user-defined attributes like conflict resolvers and search strategies to influence the course of inference processes, etc.

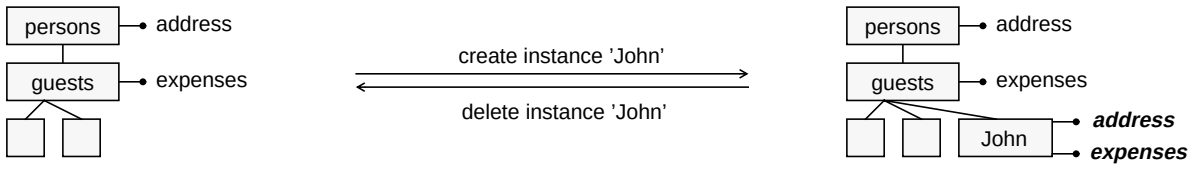
3.2 Exploiting KOBRA as Modeling Tool

As already mentioned, special 'automatic' reasoning facilities are supported by KRISYS as part of the abstraction concepts and activated by modification and retrieval operations whenever necessary.

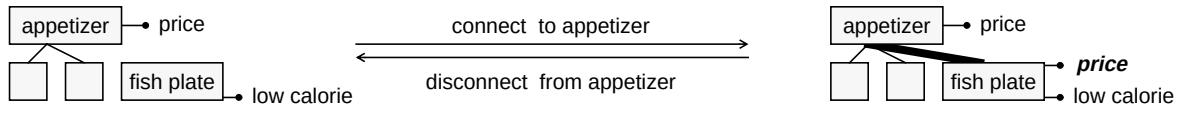
Classification/Generalization reasoning

The most usual of these reasoning facilities is the one built into the is-a hierarchy, i.e., inheritance. Since classes define the structure (i.e., attributes and constraints) of their instances and subclasses, by inserting objects in the KB and expressing the belief of the modeler that they are instances or subclasses of some classes, KOBRA can reason that they have the properties defined by these classes (see figure 3a).

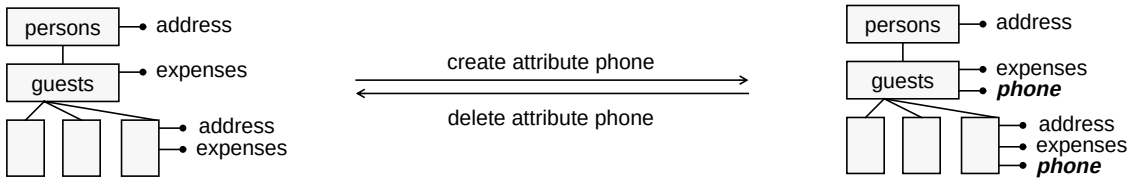
However, objects may exist in the KB *without the specification of a class*. Frequently, the KB designer knows about the existence of an object but not of its type so that it has to be first introduced in the KB without any instance-of specification. After this, the KB designer might reflect upon its type or he might want to see how the object would look if it were an instance of a particular class (e.g., he 'believes' at first that fish plate is an appetizer) and defines an instance-of relationship. (Note that in this case the modeler is using the KBMS as a tool to determine the structure of the objects, by dynamically defining instance-of relationships.) KOBRA will then deduce the object structure, generating a more detailed description of this object. Based on this new description, the KB designer might now realize that it does not correspond to the real world entity, starting the determination of the object type once again (i.e., he will realize that fish plate is not an appetizer, as illustrated in figure 3b). Therefore, instance-of relationships correspond, in truth, to current beliefs of the modeler, which may change at any time without affecting the existence of other objects or further descriptions (i.e., properties, constraints, etc.) that an object may possess. (Obviously, the same holds for objects connected by subclass-of relationships).



a) introducing / removing instances or subclasses



b) connecting / disconnecting instances or subclasses



c) creating / deleting attributes or constraints

Figure 3: Reasoning facilities of generalization and classification

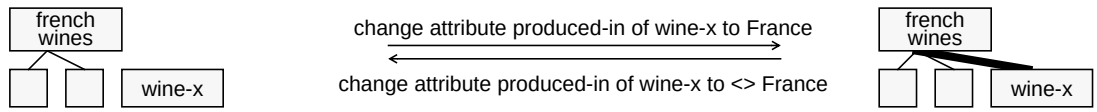
In analogy to a dynamic definition of relationships, attributes or integrity constraints (by means of aspects) can also be defined in a stepwise fashion. In this case, inheritance is employed to ensure that each instance or subclass has, at least, the attributes of its superclasses. If the attribute phone is added to the object guests, it

will be immediately inherited by all its subclasses and instances (figure 3c); if it is deleted, KRISYS automatically removes it from the corresponding subclasses and instances.

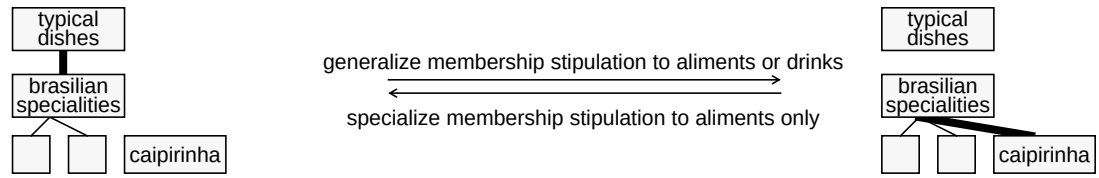
Association reasoning

Reasoning on association hierarchies is at first provided by the so-called membership stipulations, i.e., properties that an object must satisfy in order to be added to the group of elements of a set. Since KRISYS guarantees that every element of a set always fulfils the corresponding membership stipulation, it is possible to deduce, for example, that all elements of brazilian specialities are from Brazil. Consequently, by specifying membership stipulations based on the structure of the elements (e.g., elements of the set french wines must have France as the value of the slot produced-in), KOBRA can determine whether a change in an element should cause the dissolution of an existing association relationship or the creation of a new one because of the satisfaction of the membership stipulation of another set. As such, when one erases the value France from the slot which defines the site where a particular wine is produced, this wine is automatically removed from the set french wines (figure 4a).

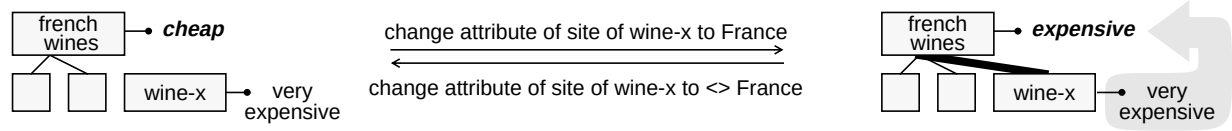
Similar conclusions can be drawn when modifying membership stipulations. In this case, not only may elements be disconnected from a particular set, but the set itself might be dissolved from its relationships to supersets since an object may be a subset of another only if it possesses more restricted membership stipulations. This is illustrated in figure 4b where the set brazilian specialities was disqualified as subset of typical dishes because its membership stipulation was generalized to accept also drinks, thereby qualifying 'caipirinha' (i.e., Brazil's national drink).



a) modifying values of attibutes of the elements



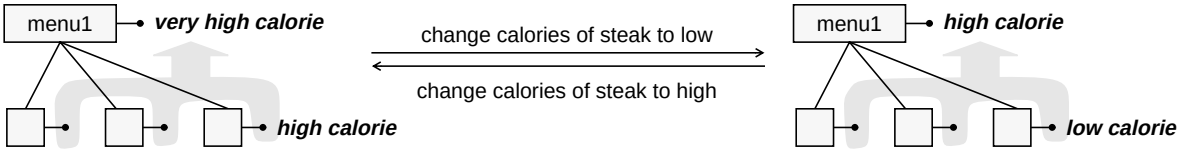
b) modifying membership stipulations



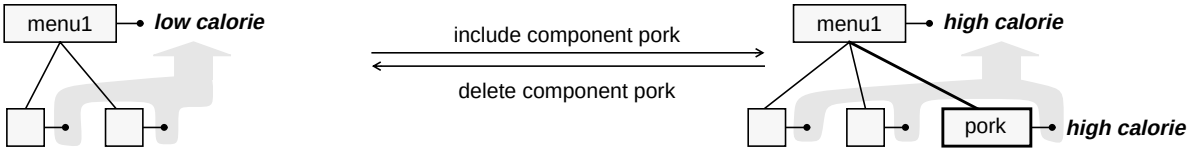
c) recalculating set properties

Figure 4: Reasoning facilities of association

Further reasoning capabilities are provided by so-called set properties, i.e., properties that describe the characteristics of the group of elements as a whole. Brazilian specialities, for example, has properties defining the average of price of specialities, their number, etc. Since such set properties are in reality based on characteristics of each individual element, conclusions about the values of set properties can be drawn from the elements' attributes. So, upon changes on elements (e.g., because of a modify operation) or on the relationships between elements and sets (e.g., because of the introduction or deletion of an element), the recalculation of the values of set properties is also performed by KRISYS (figure 4c).



a) modifying values of attributes of components



b) introducing / removing components

Figure 5: Reasoning facilities of aggregation

Aggregation reasoning

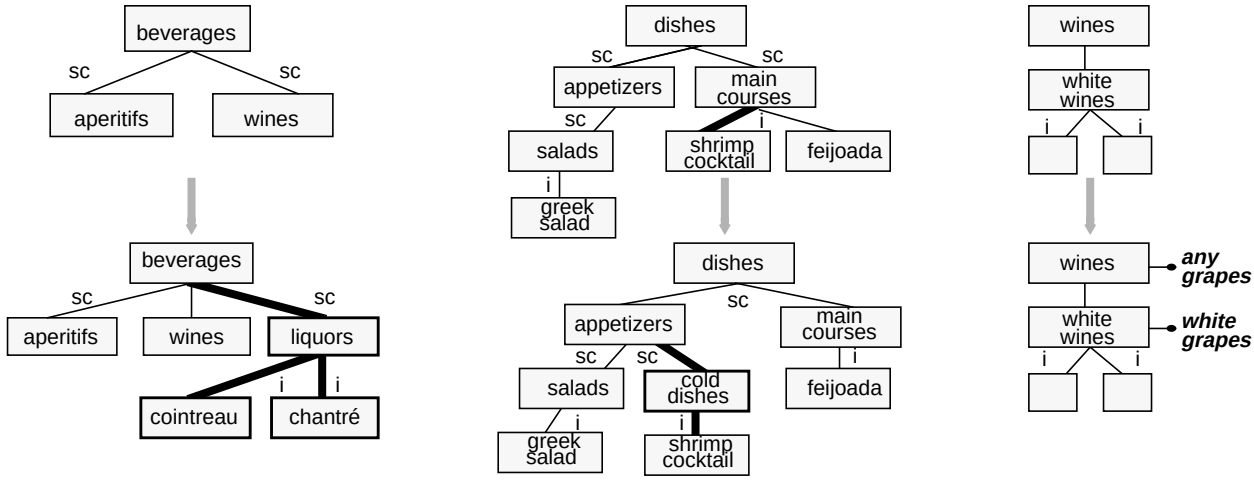
In the aggregation, reasoning is performed based on the specification of the so-called implied predicates, i.e., predicates expressing monotonically increasing or decreasing properties on the aggregation hierarchy. For example, the number of calories of a dish increases moving upwards on an aggregation since it is defined by the sum of the corresponding parts' calories. The same may occur for the dish's price, weight, etc. Hence, conclusions can be drawn by simply guaranteeing the truth of such implied predicates, e.g., when the cook invents a new way to prepare steaks, thereby reducing their calories, KRISYS automatically reconsiders the number of calories of all objects composed of steaks (e.g., menu1 in figure 5a). In the same manner, similar conclusions are drawn, for example, when objects are introduced in or removed from an aggregation, as illustrated in figure 5b.

Knowledge reformulations

It is important to observe that the reasoning facilities described above can also be viewed as a means for undertaking the maintenance of the structural and semantic integrity of the KB [De90]. For example, inheritance is employed to ensure that each instance or subclass has, at least, the attributes and satisfies the constraints prescribed by its superclasses. Since KRISYS guarantees that such integrity holds even if changes to the KB structure occur, knowledge reformulations may be carried out effortlessly without affecting any other aspect of the existing knowledge. This feature is essential for KS development support since most prototypes perform poorly at the start because of an erroneous transformation of the application world to objects of the knowledge model, an initial neglect of some refinements, or even because the knowledge engineer simply learns some more details about the application domain. By diagnosing the weakness and deficiencies of the KS, the knowledge engineer can go back to some of the developing phases either to conceptualize, structure, or refine some further aspects of the application world previously introduced into the KB, leading to an incremental, evolutionary development. To support such KB evaluation, KRISYS provides several means, such as for example, simulation of rule-based inference processes, error handling inside of methods, stepwise rollback of the execution of demons, rules, and methods, trace facilities, definition of break conditions, etc.

Refinements or changes of existing specifications are certainly the easiest modifications to be performed on KB. These are directly achieved by using KRISYS operations to define new aspects, membership stipulations, and implied predicates, delete old ones, restrict or generalize them, or even move them up and downwards on

the corresponding hierarchies. In the example shown in figure 6c, a constraint for the instance-of relationship was introduced and correspondingly specialized to control the insertion of instances of wines into the right subclass according to the kind of grape used for their production.



a) redesigns b) restructurings c) refinements

Figure 6: Examples of knowledge reformulations

Restructuring usually generates new concepts and modifications in the abstraction hierarchies. For example, the chef is not satisfied with the defined structure of dishes. He objects to the fact that cold dishes (e.g., shrimp cocktail) are not a kind of main course but an appetizer. Thus, the knowledge engineer has to introduce cold dishes as a subclass of appetizers and change the superclass of the corresponding dishes to that new class (figure 6b). The same occurs in the case of redesigns after a new conceptualization of the real world. In order to supply the guests with a more extensive selection of beverages, the head waiter wants to have liquors included in the 'carte'. Such information requires, therefore, the introduction of a further subclass of beverages and the corresponding instances into the KB (figure 6a).

Restructuring and redesigns demand adding new objects and attributes to the KB, inserting new classes, sets, or aggregates into existing hierarchies, moving objects and attributes up, down, or sidewards on such hierarchies, renaming objects and so on. In general, the most radical changes come from adding or deleting abstraction relationships between objects. But even in the case of these radical changes, KOBRA provides direct and flexible ways to achieve such reformulations, keeping the correctness of the affected built-in reasoning facilities, but without losing any of the correct information previously introduced into the KB. For example, when moving shrimp cocktail to the right position in the hierarchy in figure 6b, none of its attributes, values, and constraints, received because it is a dish, will get lost.

3.3 Summary

KRISYS puts special emphasis on a complete support of abstraction concepts in order to use their semantics as a basis for drawing conclusions about the objects and for maintaining the integrity of the KB. Such an integrated view of abstraction concepts is one of the most important aspects that differentiates KRISYS from existing knowledge engineering tools like ART [Wi84], KEE [FK85,Fi88], KNOWLEDGE CRAFT [FWA85], LOOPS [BS83,SB86], etc., as well as from existing DBS. These systems either neglect the existence of some of these concepts or make a kind of 'hodgepodge' with their semantics (see [MM89] for details). KRISYS on the other hand keeps a clear separation of these concepts, but exploits its underlying semantics to be used as a powerful modeling tool, enabling the process of KS development to be

- easier (since the knowledge model is also an active agent in this process, continuously making deductions about object structures),
- more flexible (since it is possible to repeat previous modeling steps in order to make corrections or improvements on the KB), and
- more consistent (since KRISYS is continuously maintaining the structural and semantic integrity of the KB by enforcing the reasoning facilities of the abstraction concepts).

4. Mapping Knowledge Structures to the Server

After having described the support of KS development by means of the KOBRA model, we will now concentrate on the mapping approach of KRISYS. So, we first give a short overview of the MAD model [Mi89], the model provided by the server of KRISYS. Then, we describe how the knowledge structures of KOBRA are mapped to MAD. For sake of clarity, whenever we talk about data model, database (DB), or DB schema, we mean the model and the representation of the KB at the server side.

4.1 The MAD Model and its Adequacy for Mapping KOBRA Structures

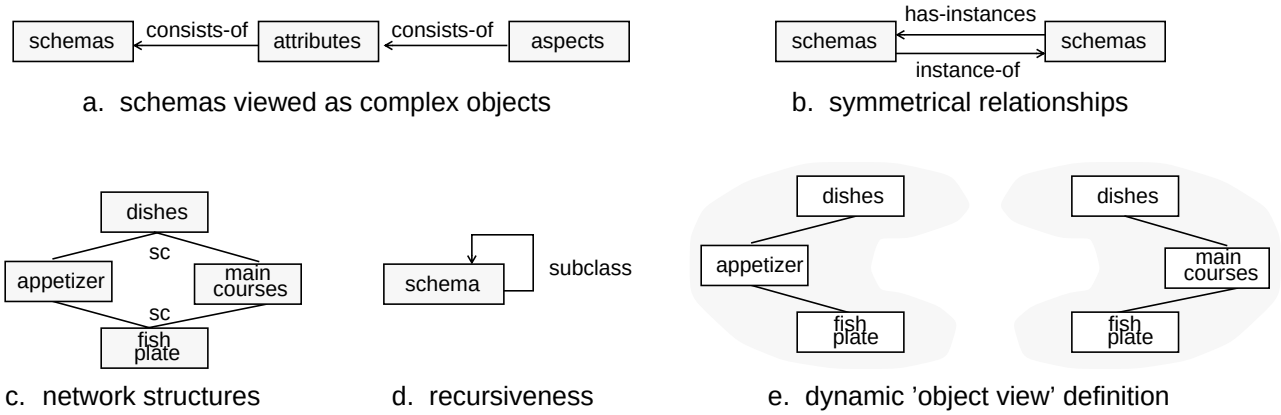
Short notes on MAD

The DBMS kernel chosen for KRISYS, named PRIMA, PRototype Implementation of the MAD model, was developed for supporting applications that require a suitable representation of complex objects, i.e., those whose inner structures (the components) are also objects of the DB [Hä88,HMMS87]. The basic modeling constructs of the MAD (Molecule Atom Data) model [Mi89] are denoted atoms, which, in analogy to tuples in the relational model, are composed of attributes and have their structure determined by an atom type. Atoms possess an identifier which is used for a *direct and symmetric representation of relationships* (1:1, 1:n, n:m) by means of links, providing a view of the DB as a complex *network* of atoms. For each specified link, there is always a corresponding back-reference in the related atom, whose mutual referential integrity is automatically maintained by the system. *Complex objects* are dynamically defined by the specification so-called molecules as a graph having atoms as nodes and relationships (i.e., links) as edges. Thus, molecules are *dynamically derived views* of the atom network. Since such views may, in turn, be used to construct other more complex views, molecules may be *recursively* defined.

Knowledge structures underlying the KOBRA model

MAD enables the mapping of KOBRA knowledge structures in an effective and straightforward manner [Mi88,Ma90]. From the description of KOBRA, one may observe the occurrence of only three different constructs in our knowledge representation framework: *schemas* representing real world objects, *attributes* expressing their properties and relationships, and *aspects* describing the attributes. From a structural point of view, schemas are therefore composed of attributes, which, in turn, consist of aspects, suggesting the internal representation of *complex objects* (figure 7a). Second, one may note that schemas possess predefined attributes expressing organizational axes (such as instance-of for classification and subclass-of for generalization) representing *symmetric n:m relationships* (figure 7b) between schemas (back references are respectively has-instances and has-subclasses). These relationships are especially important because they provide different ways to organize schemas into *network hierarchies* (figure 7c). For the construction of such hierarchies, one might have to *recursively* follow a particular relationship in order to structure schemas of a higher level

(figure 7d). Finally, one may observe that during the manipulation of a schema, one is defining a kind of *dynamic view* of this object since one is always dealing with only part of an abstraction hierarchy (in figure 7e, one is 'viewing' the properties of fish plate considering their characteristics of either an appetizer or a main course).



a. schemas viewed as complex objects

b. symmetrical relationships

c. network structures

d. recursiveness

e. dynamic 'object view' definition

Figure 7: KOBRA constructs from a structural point of view

From the description above, one may conclude that the constructs offered by MAD allow an accurate and suitable mapping of KS-oriented structures. The question now is whether such powerful constructs can provide the required functionality for KS development and processing with acceptable efficiency.

4.2 The Mapping Scheme of KRISYS

The efficiency of KRISYS functions directly depends on the chosen MAD schema for the representation of KOBRA structures together with the corresponding mapping mechanism.

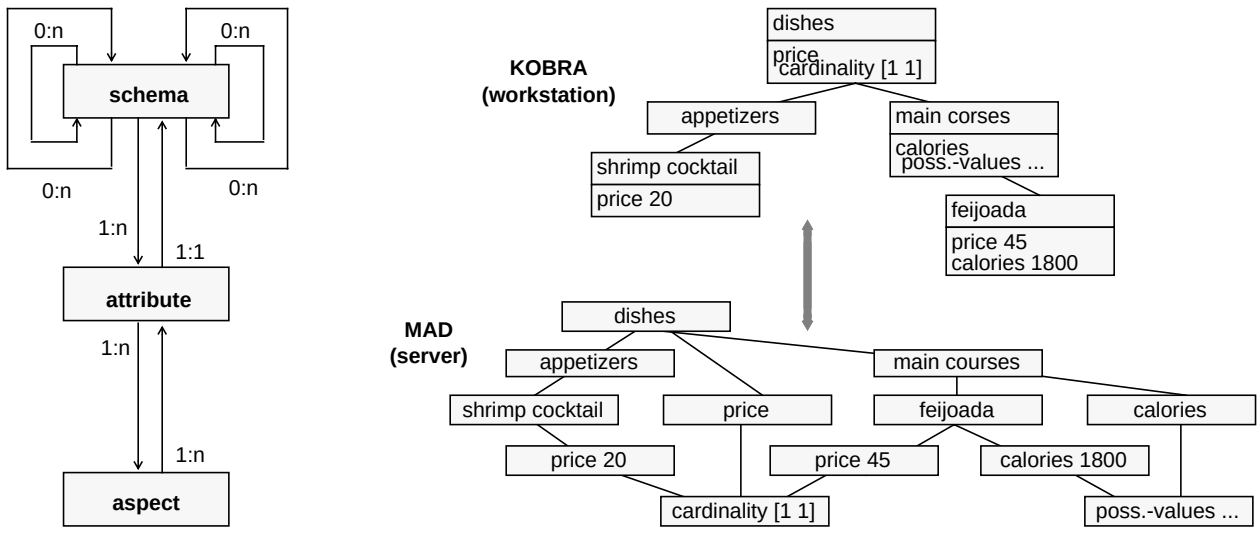
During KS development

From the description of KS development given in chapter 3, it should be clear that for the construction or modeling phase, KRISYS has to provide a flexible mapping scheme especially appropriate for efficiently supporting functions that modify KB structures since object definitions and abstraction hierarchies are constantly being changed or extended. Hence, when mapping KOBRA structures to MAD, a separation between objects representing meta-information (e.g., classes) to be introduced in the DB schema of MAD and objects representing 'common' information (e.g., instances) to be included in the DB leads to an ineffective mapping because most KOBRA operations would provoke modifications on the MAD schema (e.g., to define, remove, expand, or shrink atom types, change attribute specifications, etc.) which are hard to accomplish in PRIMA. (Note that this is the case of most DBMS, which are designed along the lines of a quite stable schema information.) Thus, it is necessary to abstract from the role (i.e., class, instance, set, and so on) played by an object at the KOBRA level in order to be able to treat them all in the same manner at the MAD level only by means of operations on the DB. Since such roles are intrinsically related with the semantics of KOBRA, it is important to abstract from such semantics and consider KRISYS objects only from a structural point of view.

From such a viewpoint, schemas, attributes, and aspects are related to each other to compose the KOBRA objects, leading to a MAD schema (graphically illustrated in figure 8a) that contains three atom types (respectively 'schema', 'attribute', and 'aspect') connected via the references ('has_attributes' and 'has_aspects'). Here, aspect specifications are shared between several attributes preventing redundancy, especially in the

case of inherited attributes (see figure 8b). We explicitly exploit the capability of the MAD model to handle network structures in a direct and non-redundant manner. (Such an adequate modeling is hard to achieve using data models that only allow for hierarchical structures, e.g., NF^2 models [SS86,Da86]). The attributes expressing abstraction relationships are not represented as ordinary 'attribute' but as recursive MAD references (e.g., 'has_subclasses', 'is_subclass_of', 'has_instances', etc.) between the atom type 'schema' (only partially shown in figure 8a). Since there is always a back reference for each MAD link, each of the organizational axes is represented by a pair of references.

The mapping of KOBRA objects to MAD atoms is performed at the workstation side of KRISYS, whenever objects are stored into or discarded from its object buffer in order to allow a buffer representation embodying the semantics of the KOBRA model (figure 8b). Therefore, for the realization of KOBRA operations, the query language of MAD (MQL) is employed to fetch objects from the DB. For example, the definition of a new class in the middle of a generalization hierarchy, which requires the inheritance of its attributes, generates an MQL statement to select all relevant classes and instances from the DB. This task is accomplished by dynamically defining a recursive molecule starting with the root atom type 'schema' moving on to all subclasses (using the reference 'has_subclasses' in a recursive manner) and then to their instances (by exploiting the reference 'has_instances'). For each object reached, their attributes and aspects are also retrieved (via 'has_attributes' and 'has_aspects'). The selected part of the hierarchy is then modified in the object buffer, during which inheritance conflicts are solved, the inexistence of cycles or other ambiguities (e.g., an object having a same object as subclass and instance at the same time) in the hierarchy are checked, etc. At some time later on (probably at the end of a modeling step), the modified hierarchy is then discarded from the buffer and propagated back to the DB, where it is updated via a single modify statement. This operation changes the recursive molecule in accordance to the given hierarchy by inserting atoms not yet stored and updating the corresponding connections.



a) MAD schema diagram b) mapping process

Figure 8: Representation of KOBRA objects in MAD during KS development

Note, however, that when introducing such modifications back into the server no changes need to be made in the schema information of MAD. Creations and deletions of KOBRA schemas (even classes, sets, or aggregates) are mapped to modifications on atoms of type 'schema'; manipulations of attributes (e.g., definitions, inheritance, etc.) provoke inserts, deletes, and updates on instances of the 'attribute' type; and changes on as-

pects (including their specialization) require only operations on 'aspect' atoms. Hence, the required flexibility for incrementally defining, correcting, and extending the KB during KS development is well supported.

During KS processing

For KS processing, where flexibility is in general no longer required, more specific mappings should lead to significant gains in performance. Naturally, in order to exploit such specific mapping schemes, the MAD schema and the corresponding transformation process of KOBRA objects into MAD atoms must not be fixed to a certain mapping scheme (as during the modeling phase), but has to be tailored to the structures and processing characteristics of the application. As mentioned in chapter 1, when the modeling process of a KS is completed, the structures of its KB differ very much from the KB of other KS (figure 9). Therefore, the necessity to provide an 'application-oriented' DB schema with the corresponding mapping process for the operation phase of a KS becomes apparent. Nevertheless, most existing prototypes [Ni87,Ba87,Fi87] present only fixed, predefined mapping schemes neglecting the exploitation of application characteristics for optimization purposes [WK90].

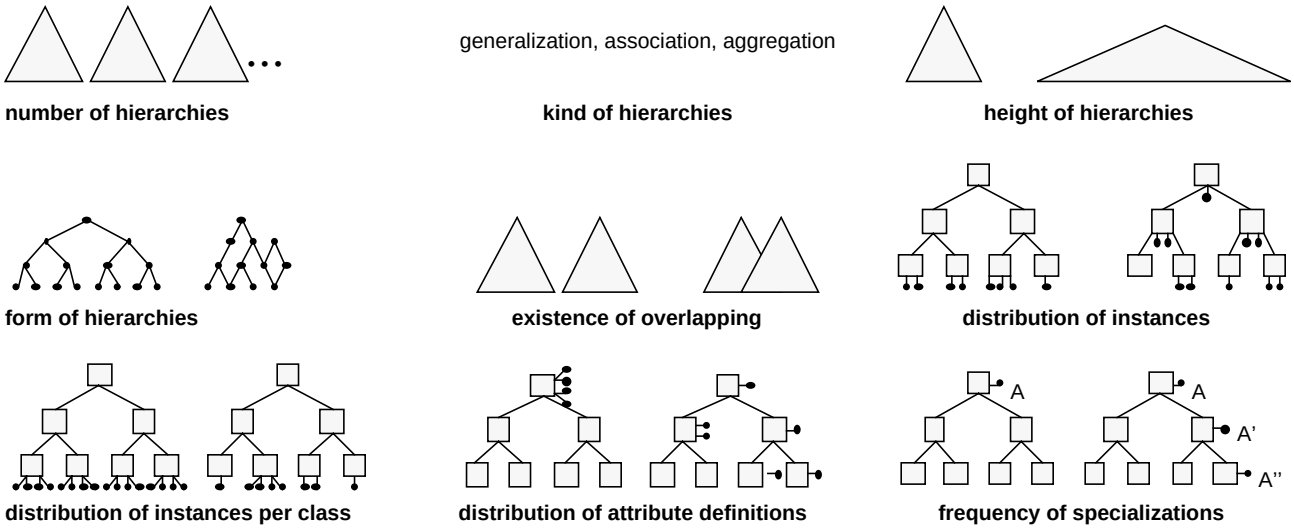


Figure 9: Some differences of KB structures of KRISYS applications

KRISYS, on the other hand, uses such information to generate an application-oriented, and consequently very effective mapping scheme for each KS. Our approach is to extend the development process of KS by including a latter phase, called optimization, during which KRISYS specifies a new MAD schema tailored to the KS at hand and automatically performs a DB transformation. The most appropriate DB schema for a KS is determined by means of an analysis of the structures of the KB, that are completely specified and explicitly represented in KRISYS just before the optimization takes place [Ma90]. For example, the attributes of a KOBRA object, which are represented as separate atoms in the generic mapping, could all be represented as attributes of a single atom in the specific mapping, and classes could be used to define corresponding atom types. Once the new DB schema is generated, a compilation of the KS is then performed, thereby also precompiling the application queries encoded within methods, demons, and rules. In other words, the optimization phase represents the 'exit' of an interpretative (flexible but probably not so efficient) KB manipulation, which is necessary during KB construction, and the 'entry' into a compiled (efficient but not so flexible) KB manipulation, which is necessary during KS operation.

Although the generation of such new DB schema appears to be quite simple at a first glance, it involves the consideration of a lot of information and requires complex decisions to be made. Just as an example, not every

class should generate an atom type because there might be classes that will never receive direct instances in a generalization hierarchy. So, one could deduce a rule like 'every class possessing direct instances should have a corresponding atom type'. However, there are classes with direct instances, but presenting the same structure (i.e., attributes and aspect specifications) of a common superclass. They were (probably) defined only for organizational purposes. In this case, not they, but only this superclass should define an atom type, in which all instances would be inserted. One would now conclude that 'every heterogeneous class (i.e., showing an individual structure) with direct instances should generate an atom type'. Nevertheless, even this conclusion is not always correct because there are a lot of cases in which grouping heterogeneous classes has good reasons to be carried out. For example, when two heterogeneous classes are used to define a set (e.g., elements of brazilian specialities are either drinks or main courses), it might be important to put them together into one single atom type because KRISYS wants to exploit the corresponding membership stipulation or the attributes involved in the calculation of set properties to specify appropriate access paths. In this case, the only practical solution would be to create three different atom types and split the instances of both classes over these three: the first atom type including only common attributes of these classes, on which the definition of the membership stipulation and set properties is based (e.g., produced-in, price, etc.); and each of the other two having the individual attributes of each class (e.g., percentage of alcohol for drinks and recommended wines for main courses). Naturally, in the case of queries searching for such objects, accesses to two different atom types would be necessary. However, this can be efficiently supported by using MAD references or even clustering mechanisms to keep both parts of each KOBRA object close together.

From the short discussion above, one can conclude that there is no fixed procedure to be followed in order to derive the best MAD schema for a KS, but only some heuristics that have to be carefully considered. Hence, the use of rule-based reasoning mechanisms seems to be very appropriate here since they provide the required expressiveness and flexibility to represent and interpret such heuristics. So, during the optimization phase, KRISYS exploits the rule-based mechanisms provided by the KOBRA model to generate a tailored mapping scheme. For this purpose, it interacts with the knowledge engineer whenever necessary in order to get additional, relevant information for making its decisions (e.g., the number of expected instances per class, elements per set, the relevance of membership stipulations for access purposes, etc.). In the case of an erroneous estimation, the optimization phase would then be repeated some time in the future in order to generate a new, more appropriate MAD scheme. KRISYS will then reconsider the increasing costs of this new transformation since several objects might be now stored in the KB so that the same issues used for optimization of DB design can be applied here.

Note, however, that when a transformation of the MAD schema is performed or repeated, the application is not affected at all since it still sees only the unchanged KB view provided by the KOBRA model. In contrast to that, existing DBS (even relational ones) cannot always guarantee such immunity of their applications to all kinds of changes in the DB (i.e., data independence). For example, if one unites two relations to materialize a join, the application view of the DB turns out to be quite different. (The work with views certainly eases the problem but does not lose it completely when considering all kinds of updates.) For these reasons, we believe that our approach of using a model for the internal KB representation completely independent of the external model of the system is the only means for *constructing KS under a kind of 'knowledge representation concealing principle'* that guarantees local confinement of all modifications and improvements in KRISYS throughout the lifetime of its applications. This independence is in turn exploited as basis for generating various, suitable mapping

schemes, thereby providing a kind of adaptability of our system to support the 'contradicting' requirements of flexibility and efficiency.

5. Processing Model

Before we describe the dynamic behavior of our system, it is necessary to recapitulate the main issues of the KRISYS architecture. At the server side, the PRIMA kernel implements MAD, thereby offering general management functions to efficiently provide KB maintenance. In this layer, great care was taken to guarantee an efficient molecule extraction and inclusion by means of a variety of storage structures and other tuning mechanisms (e.g., molecule materialization or molecule caching) [Hä88]. At the workstation side, three different layers realize the interface of the system, the knowledge representation framework underlying the KOBRA model, and the 'nearby application locality' concept. This concept is achieved by a component, denoted Working-Memory System (WMS) [LM89].

Hence, from the kernel's point of view, the task of the WMS is the embedding of MAD structures into an environment in which they can be easily and efficiently manipulated by the upper layers of KRISYS. This is achieved by exploiting an object buffer in order to avoid long execution paths of KB accesses (involving workstation/server communications) and time-consuming requests to secondary storage. Thus, the objective of the WMS is to considerably reduce the path length and to minimize the number of kernel calls when accessing KB objects. The use of an object buffer, called working-memory, permits the storage of requested objects in a main-memory structure offering almost direct access at costs comparable to a pointer-like access. Consequently, the WMS supports a processing model aimed at a high locality of object references.

To reach this end, the WMS exploits the concept of processing contexts representing the knowledge needed during a specific KS processing phase [Ma90,MW84]. Consequently, they directly depend on the problem solving strategy being used by the KS. For example, KS exploiting constraint-propagation strategies [St81] have contexts corresponding to the several constraints used to reduce the search space during KS consultation. In the same manner, a problem-reduction strategy [Mc82] determines contexts for processing each partitioned subproblem, and generate-and-test [BF78] organizes KB contents according to pruning activities (for an overview of such strategies see [St83]). For this reason, the knowledge engineer, as the one who knows the KS in fullest details, knows about the existence of contexts. His task is to make known existing contexts in order to supply KRISYS with valuable information for performance improvement. From a modeling point of view, defining contexts means making explicit the knowledge about the KS problem solving behavior, thereby improving KB semantics.

In principle, contexts are composed of several KB objects (in general of different types) and objects may be elements of several contexts. They are explicitly represented as KOBRA schemas and completely specified by means of the query language of KRISYS, being therefore totally independent of the chosen mapping mechanism to MAD. Thus, they may be specified or deleted at any time during KB construction or even dynamically during KS consultation without additional costs. A dynamic definition is particularly important because needed contexts are often established on the basis of the results of preceding phases of the problem solving process. The WMS exploits the specification of a context to generate a complex set-oriented access to the kernel to fetch required objects and store them into the working-memory. So, most or perhaps all objects referenced during a processing phase of the KS are found in the working-memory so that only a few or no further calls to the

kernel are necessary. At the end of the processing phase, the corresponding context is then discarded from the working-memory and the context requested by the following phase is loaded into it.

Note that the WMS is also exploited as a basis for efficiently implementing the mechanisms provided by the KOBRA model during KS development. Considering the KOBRA operation described in the previous chapter which inserts a class into a generalization hierarchy, the WMS dynamically defines a processing context to extract the corresponding part of the generalization hierarchy from the kernel loading it into the working-memory in a single MAD operation. In this main-memory structure, inheritance is then efficiently performed without further MAD calls. At the end of the modeling step, when all necessary integrity checks are already done, the extracted context is discarded from the working-memory and propagated back to the KB. The support of rules during KS processing is also carried out in a similar manner. Before an inference process takes place, the corresponding context (i.e., the rules, objects specified in the condition part as well as objects possibly affected by the action part) is extracted from the kernel and stored in the working-memory for forward or backward reasoning. As a consequence, reasoning tasks are carried out in the workstation, thereby drastically reducing communication overhead.

In closing, one may observe that processing contexts are the most important mechanism used by the WMS of KRISYS to improve access efficiency. Viewed from the knowledge engineer, a context is a collection of objects needed by a specific phase of the problem solving process of the KS, i.e., the knowledge necessary to work out a particular problem. KRISYS, on the other hand, views it as a collection of objects which have to be brought into the working-memory by just one very efficient KB access. Following this approach, when specifying contexts, the knowledge engineer is still working in his framework, without being involved with internal KBMS aspects, although KRISYS is supplied with enough information for performance improvement [LM89].

6. Why not Enhance the MAD Model with KOBRA Semantics?

Throughout the scope of this paper, we have emphasized that KRISYS was developed to support workstation-oriented KS development and processing. As already mentioned, the aspects of this environment are reflected in the division of the system architecture in two main parts, having the KOBRA model at workstation side and the MAD model as interface of the central server. Considering the constructs underlying these models, it should become clear that KOBRA is located at a higher semantic level than MAD. Upon realizing the existence of this gap in the KRISYS architecture, one would directly come up with the idea of enhancing MAD with KOBRA semantics, thereby equalling both models to eliminate such a gap. The 'new MAD' model would then provide the same mechanisms as KOBRA, but would certainly rely on an internal interface supporting similar semantics and providing the same services as the existing MAD model. Regarding the architecture of KRISYS, the question now would be where to set the 'borderline' between server and workstation. By placing it on the internal interface of the 'new MAD', one would again be defining the architecture presented by KRISYS so that all arguments discussed in this paper would be equally relevant. By placing it on the enhanced MAD interface, all the functionality of KOBRA would be completely delegated to the server KRISYS, otherwise KOBRA semantics would have to be duplicated at the workstation (implying that the MAD enhancement is meaningless). Such a complete delegation would hopelessly overload the server component of KRISYS with additional processing since the maintenance of abstraction concepts, the execution of their built-in reasonings, the evaluation of methods, demons, and rules are now completely undertaken by the server. Processing at workstation would be limited to a minimum, leading to a more or less centralized system architecture that neglects the advantages

provided by workstation/server environments. Hence, efforts to equal MAD and KOBRA do not offer solutions but only shift the existing semantic gap into the MAD model.

When examining a partial delegation of KOBRA features, other problems rise. Let us consider, for example, the delegation of only the semantics of the abstraction concepts so that a lot of processing capability would still be necessary at workstation side for evaluating methods, rules, etc. Here, two basic alternatives for carrying out such task can be followed. In the first one, modifications on KOBRA objects are just accumulated in the working-memory and propagated to MAD only at the end of a modeling step when the KB integrity (partially maintained by MAD) is then checked (e.g., the truth of implied predicates, values of set properties, inheritance rules, etc.). In the second one, propagation of object changes are not postponed but immediately carried out in order to allow an instantaneous check of KB integrity. Unfortunately, both alternatives do not present reasonable solutions. The first one should be disqualified because the whole modeling step (which is a long-term activity) would have to be rolled back upon integrity violation and because the mentioned integrity must be immediately checked otherwise flexible reactions involving interactions with the knowledge engineer (e.g., to solve inheritance conflicts, eliminate ambiguities in the abstraction hierarchies, etc.) would either not be possible or lead to additional transfer of objects and communication overhead. The second one enables such immediate integrity checks but has to be ruled out because it contradicts our efforts to support locality of references and minimum of communication between server and workstation.

Certainly, the above observations did not exhaust all possibilities for a thinkable delegation of KOBRA functions to MAD. (For a detailed discussion on this topic see [De90]). However, we believe that we have given enough arguments to conclude that the semantics provided by KOBRA at the workstation side should not be introduced into MAD. The semantic gap existing in the KRISYS architecture is not a mismatch but only a *natural reflection of the stepwise abstraction process realized at each layer of our system*. On the server side, MAD treats knowledge structures simply as a kind of network of complex objects to be consistently, reliably, and efficiently managed. At the workstation side, these structures obtain semantics, known only by the KOBRA model, remaining, for this reason, outside the server. In this way, the server is not overloaded by modeling specific aspects being, therefore, able to concentrate on an efficient KB maintenance. On the other hand, the KOBRA model is not bounded by the semantics provided by the MAD model being, as a consequence, able to come nearer to the application semantics by offering a rich and powerful spectrum of concepts for KS modeling and processing. This partitioning is further favored by the set orientation of the MAD interface and the locality preservation of the WMS, both minimizing workstation/server communication. Additionally, the loose coupling greatly facilitates failure isolation. This is a very critical design objective because a large number of users may be affected by any kind of failure and because interactive KS development is typically very long. Summarizing, we argue that the kernel should provide only features of a basic nature: complex object handling including recursion, set orientation, efficient access to secondary storage, and transaction support (at least for concurrency control and recovery purposes on the server side).

7. Summary

KRISYS is a KBMS prototype appropriately constructed to support KS development and processing. For KS development, the KOBRA model provides a rich spectrum of mechanisms for knowledge modeling, thereby advocating the KB construction in a stepwise fashion. In other words, KS development is realized by a cyclic series of activities including knowledge reformulations during which the reasoning facilities of the abstraction concepts are applied for continuously making deductions about the object structures and maintaining the se-

mantic integrity of the KB. Thus, KRISYS can be used as a powerful modeling tool allowing a direct and flexible extension, modification, and restructuring of KB contents.

For KS processing, it offers an efficient framework for the exploitation of the application locality, taking advantage of the knowledge about the access behavior of the KS in order to reduce the path length when accessing KB objects and to minimize I/O operations as well as transfer overhead. KRISYS, therefore, accomplishes fast access to stored KB objects by exploiting the existence of processing contexts whose contents are temporarily kept close to the application in its working-memory.

Finally, its architecture fits nicely into the most realistic and prevailing architectural environment for KS, i.e., workstation environments. Here, workstation-oriented processing is effectively enhanced by placing the working-memory, the KOBRA model, and the system interface together with the application into the workstation and the PRIMA kernel into the server. This natural separation of the system in two main parts interacting via the MAD model interface is assumed to greatly reinforce its means for KS development and processing. KRISYS strengthens its support of the modeling process and of the application operation by means of different mapping mechanisms to the PRIMA kernel: a general scheme is provided during KS development in order to allow for a flexible KB construction, and a specific, tailored to the application at hand during KS operation in order to obtain significant gains in performance.

Acknowledgment

I would like to thank the several students that have participated of our KBMS project whose combined effort has resulted in the implementation of KRISYS. Also acknowledged are the careful reading and helpful comments of T. Härder, F-.J. Leick, B. Mitschang, and J. Thomas as well as the help of M. Burkart, I. Littler, and H. Neu during the preparation of the manuscript.

References

Ba87 Banerjee, J., et al.: Data Model Issues for Object-Oriented Applications, in: ACM Transactions on Office Information Systems, Vol. 5, No. 1, 1987, pp. 3-26.

BF78 Buchanan, B.G., Feigenbaum, E.A.: DENDRAL and Meta-DENDRAL: Their Application Dimension, in: Artificial Intelligence, Vol. 11, No. 1, 1978, pp. 5-24.

BL86 Brachman, R., Levesque, H.: The Knowledge Level of KBMS, in: [BM86], pp. 9-12.

BM86 Brodie, M.L., Mylopoulos, J. (eds.): On Knowledge Base Management Systems (Integrating Artificial Intelligence and Database Technologies), Topics in Information Systems, Springer-Verlag, New York, 1986.

BMW84 Borgida, A., Mylopoulos, J., Wong, H.K.T.: Generalization/Specialization as a Basis for Software Specification, in: On Conceptual Modelling (Perspectives from Artificial Intelligence, Databases, and Programming Languages), Topics in Information Systems, (eds.: Brodie, M.L., Mylopoulos, J., Schmidt, J.W.), Springer-Verlag, New York, 1984, pp. 87-114.

Br81 Brodie, M.L.: Association: A Database Abstraction for Semantic Modelling, in: Proc. 2nd Int. Entity-Relationship Conference, Washington, D.C., Oct. 1981.

BS83 Bobrow, D.G., Stefik, M.: The LOOPS Manual, Xerox PARC, Palo Alto, CA, 1983.

CJV83 Clifford, J., Jarke, M., Vassiliou, Y.: A Short Introduction to Expert Systems, in: Database Engineering, Vol. 3, No. 4, December 1983, pp. 3-16.

Da86 Dadam, P., et al.: A DBMS Prototype to Support Extended NF^2-Relations: An Integrated View on Flat Tables and Hierarchies, in: Proc. ACM SIGMOD Conf., Washington, D.C., 1986, pp. 356-367.

De90 Deßloch, S.: Handling Integrity in a KBMS Architecture for Workstation/Server Environments, Research Report, University of Kaiserslautern, 1990, submitted for publication.

DHMM89 Deßloch, S., Härder, T., Mattos, N., Mitschang, B.: KRISYS: KBMS Support for Better CAD Systems, in: Proc. 2nd International Conference on Data and Knowledge Systems for Manufacturing and Engineering, Gaithersburg - Maryland, Oct. 1989, pp.172-182.

DHMS90 Deßloch, S., Hübel, C., Mattos, N., Sutter, B.: KBMS Support for Technical Modeling in Engineering Systems, in: Proc. 3rd International Conference of Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, Charleston - South Carolina, July 1990, pp. 790-799.

DLM90 Deßloch, S., Leick, F.J., Mattos, N.M.: A State-oriented Approach to the Specification of Rules and Queries in KBMS, Research Report, University of Kaiserslautern, 1990, submitted for publication.

Fi87 Fishman, D.H., et al.: IRIS: An Object-Oriented Database Management System, in: ACM Transactions on Office Information Systems, Vol. 5, No. 1, 1987, pp. 48-69.

Fi88 Filman, R.E.: Reasoning with Worlds and Truth Maintenance in a Knowledge-based Programming Environment, in: Communications of the ACM, Vol. 31, No. 4, April 1988, pp. 382-401.

FK85 Fikes, R., Kehler, T.: The Role of Frame-based Representation in Reasoning, in: Communications of the ACM, Vol. 28, No. 9, Sept. 1985, pp. 904-920.

FWA85 Fox, M., Wright, J., Adam, D.: Experience with SRL: an Analysis of a Frame-based Knowledge Representation, Technical Report CMU-CS-81-135, Carnegie-Mellon University, Pittsburgh 1985.

Ha87 Hayes-Roth, F.: Expert Systems, in: Encyclopedia of Artificial Intelligence, Vol. 1, (ed.: Shapiro, S.C.), John Wiley & Sons, New York, 1987, pp. 287-298.

Hä88 Härder, T. (ed.): The PRIMA Project Design and Implementation of a Non-Standard Database System, SFB 124 Research Report No. 26/88, University of Kaiserslautern, Kaiserslautern, 1988.

HHMM88 Härder, T., Hübel, C., Meyer-Wegener, K., Mitschang, B.: Processing and Transaction Concepts for Cooperation of Engineering Workstations and a Database Server, in: Data and Knowledge Engineering, Vol. 3, 1988, pp. 87-107.

HMMS87 Härder, T., Meyer-Wegener, K., Mitschang, B., Sikeler, A.: PRIMA - A DBMS Prototype Supporting Engineering Applications, in: Proc. 13th VLDB Conf., Brighton, UK, 1987, pp. 433-442.

HR85 Härder, T., Reuter, A.: Architecture of Database Systems for Non-Standard Applications (in German), in: Proc. GI-Conference on Database Systems for Office, Engineering and Science Environments, IFB 94, Springer-Verlag, Karlsruhe, 1985, pp. 253-286.

Kr89 The KBMS Prototype KRISYS - User Manual, Version 2.3, Kaiserslautern, West Germany, 1989.

LM89 Leick, F.J., Mattos, N.M.: A Framework for an Efficient Processing of Knowledge Bases on Secondary Storage, in: Proc. of the 4th Brazilian Symposium on Data Bases, Campinas-Brazil, April 1989.

Ma88a Mattos, N.M.: Abstraction Concepts: the Basis for Data and Knowledge Modeling, in: 7th Int. Conf. on Entity-Relationship Approach, Rom, Italy, Nov. 1988, pp. 331-350.

Ma88b Mattos, N.M.: KRISYS - A Multi-Layered Prototype KBMS Supporting Knowledge Independence, in: Proc. Int. Computer Science Conference - Artificial Intelligence: Theory and Application, Hong Kong, Dec. 1988, pp. 31-38.

Ma90 Mattos, N.: An Approach to DBS-based Knowledge Management (invited talk), in: Proc. 1st Workshop "Information Systems and Artificial Intelligence", Ulm - West Germany, March 1990.

Mc82 McDermott, J.: R1: A Rule-based Configurer of Computer Systems, in: Artificial Intelligence, Vol. 19, 1982, pp. 39-88.

Mi88 Mitschang, B.: Towards a Unified View of Design Data and Knowledge Representation, in: Proc. of the 2nd Int. Conf. on Expert Database Systems, Tysons Corner, Virginia, April 1988, pp. 33-49.

Mi89 Mitshang, B.: Extending the Relational Algebra to Capture Complex Objects, in: Proc. of the 15th VLDB Conf., Amsterdam, 1989, pp. 297-306.

MM89 Mattos, N.M., Michels, M.: Modeling with KRISYS: the Design Process of DB Applications Reviewed, in: Proc. the 8th Int. Conf. on Entity-Relationship Approach, Toronto - Canada, Oct. 1989, pp. 159-173.

MW84 Missikoff, M., Wiederhold, G.: Towards a Unified Approach for Expert and Database Systems, in: Proc. of the 1st Int. Workshop on Expert Database Systems, Kiawah Island, South Carolina, October 1984, (ed., Kerschberg, L.), Benjamin/Cunnings Publ. Comp., Menlo Park, CA., 1986, pp. 383-399.

My86 Mylopoulos, J.: On Knowledge Base Management Systems, in: [BM86], pp. 3-8.

Ni87 Nixon, B., et al.: Implementation of a Compiler for a Semantic Data Model: Experiences with Taxis, in: Proc. of ACM SIGMOD Conf., San Francisco, 1987, pp. 118-13.

RHMD87 Rosenthal, A., Heiler, S., Manola, F., Dayal, U.: Query Facilities for Part Hierarchies: Graph Traversal, Spatial Data, and Knowledge-Based Detail Supression, Research Report, CCA, Cambridge, MA, 1987.

SB86 Stefik, M., Bobrow, D.G.: Object-Oriented Programming: Themes and Variations, in: AI-Magazine, Vol. 6, No. 4, Winter 1986, pp. 40-62.

SS77 Smith, J.M., Smith, D.C.P.: Database Abstractions: Aggregation and Generalization, in: ACM Transactions on Database Systems, Vol. 2, No. 2, June 1977, pp. 105-133.

SS86 Schek, H.-J., Scholl, M.H.: The Relational Model with Relation-Valued Attributes, in: Information Systems, Vol. 11, No. 2, 1986, pp.137-147.

St81 Stefik, M.J.: Planning with Constraints (MOLGEN: Part 1), in: Artificial Intelligence, Vol. 16, No. 2, 1981, pp. 111-140.

St83 Stefik, M.J., et al.: Basic Concepts for Building Expert Systems, in: Building Expert Systems, (eds.: Hayes-Roth, F., Waterman, D.A., Lenat, D.B.), Addison-Wesley, Reading, Mass., 1983, pp. 59-86.

ST89 Schmidt, Thanos (eds.): Foundations of Knowledge Base Management, Springer-Verlag, 1989.

Wi84 Williams, C.: ART the Advanced Reasoning Tool: Conceptual Overview, Inference Corporation, Los Angeles, 1984.

WK90 Willshire, M., Kim, H.-J.: Properties of Physical Storage Models for Object-Oriented Databases, in: Proc. PARBASE 90 - Int. Conf. on Databases, Parallel Architectures and Their Applications, 1990, pp. 94-99.