in: Tagungsband derGI-Fachtagung ‘Datenbanksysteme für Büro, Technik und Wissenschaft’,Informatik aktuell, März 1993, Braunschweig, pp. 124-143.

The KRISYS Project: a Summary of What We Have Learned so far

S. Deßloch, F.-J. Leick, N.M. Mattos[1], J. Thomas
University of Kaiserslautern, CS Department
P.O.Box 3049, 6750 Kaiserslautern, Germany
e-mail: {dessloch,leick,thomas}@informatik.uni-kl.de

Abstract

KRISYS is a prototype of a Knowledge Base Management System whose first implementation was completed at the University of Kaiserslautern in 1989. Since then, the system has been used for the development of various applications which allowed us to perform a well-founded evaluation of the system. In this paper, we summarize our evaluation by describing the major lessons we have learned from the design and implementation of KRISYS and, above all, from its use in the development of these applications. We address issues related with the concepts available for application modeling and processing, the support of designing an application, as well as the overall means for efficient processing in a workstation/server environment. Additionally, we point out in how far these experiences validate our approach or stimulate improvements and future research.

1. Introduction

In the last years, substantial research efforts in the area of Database Management Systems (DBMS) have been conducted to support advanced or so-called non-standard database applications [HR85]. This research was, among other things, sparked by the lack of semantic expressiveness in current DBMS [HK87, KDE90]. They do not support the following modeling concepts, which are indispensable in obtaining a more accurate model of complex application domains:

- Abstraction concepts [BMW84,Br81,Ma88a,SS77] are primarily important for the support of a semantically enriched object description. Additionally, they define means for object organization [MM89] which, in turn, can be used to describe distinct application aspects [MDL91].
- There is a need for the integration of behavior into the application model in the form of procedural attributes, user-defined functions, or methods [At89,MMM92]. Such procedures can be used to describe actions in which application objects are involved, thereby permitting the integration of application-oriented operations into the system [DHMM89].
- Reasoning facilities [DK76,Fr86] are necessary to exploit intensional information, to deal with incomplete specifications as well as to control the overall application process, thereby also supporting an active system behavior [DHMM89].

Besides the drawbacks of their modeling concepts, current DBMS also fail to support the process of developing a complex application. This deficiency is becoming even more apparent due to the increasing costs that originate from the use of different models and tools in the development phase (e.g., ER model) and the operation phase (e.g., relational model), which is required when current DBMS are applied as the underlying management system. The need for a single, uniform tool for modeling support has a significant impact on the functionality of future DBMS: they have to be able to act not only as management systems, but also as modeling tools for developing complex applications!

Finally, the processing needs of the applications have to be fulfilled in an efficient and reliable manner, requiring also the consideration of an appropriate runtime environment for non-standard applications [HM90,Ma91]. While ‘classical’ DBMS technology was largely based on a centralized system architecture, workstation/server environments have emerged as typical for advanced application systems. Therefore, the overall architecture of future DBMS should be suitable for such a hardware environment [DFMV90, HHMM88, KDG87, Ma91]. In this setting, locality of reference should be exploited as far as possible; buffering objects close to

1. The address of Mr. Mattos is: IBM, Database Technology Institute, 555 Bailey Av., San Jose - CA - 95150, USA, e-mail: mattos@stlvm14.vnet.ibm.com.

the application seems to be the only means to achieve efficient object references. Also, coupling some kind of 'DBMS' and 'XPS' components in existing architectures is responsible for cumbersome handling and for quite poor performance in most cases [Ma90]. For this reason, the integration of knowledge-based and DBMS techniques in an effective way is one of the main issues to be addressed.

The enhancement of DBMS according to the above mentioned requirements resulted in so-called Knowledge Base Management Systems (KBMS) [BM86,Ma89,ST89]. In such systems, pieces of applications in form of user-defined functions, methods on abstract data types, abstraction relationships, and inference rules are moved inside the KBMS for better performance and higher flexibility.

Along these lines, the KBMS KRISYS (Knowledge Representation and Inference System) was developed at the University of Kaiserslautern [Ma89]. More than thirty diploma and project thesis works were involved in the overall project. The system became completely operational in 1989 [Kr89], when we started to develop several applications from different areas with it. Since 1991, KRISYS is also successfully used in a practical semester course on KBMS and object-orientation at our university.

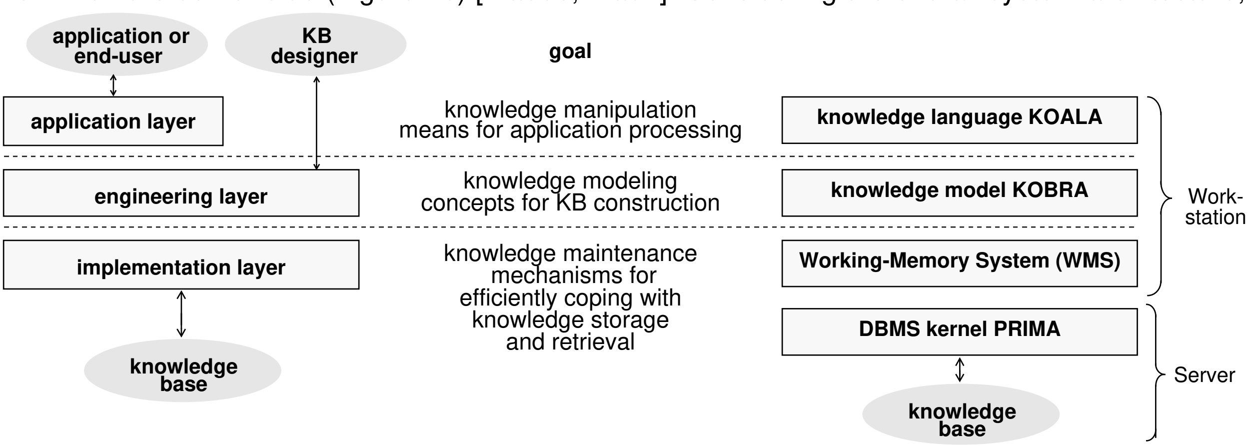
The experiences gained with the development of these applications as well as the feedback received by using KRISYS in this practical course served as a broad and solid basis for evaluating the system from various points of view. While some of the results consolidated our approach towards KBMS, others helped to reveal some deficiencies. The goal of this paper is to summarize the results of this evaluation and present the 'concrete lessons learned' in the KRISYS project so far. After this introduction, Section 2 gives a brief overview of the architecture of KRISYS and of its main components. In Section 3, the main part of the paper, we present the results of our evaluation followed by some conclusions and an outlook which are given in Section 4.

2. A Brief Overview of KRISYS

2.1 Overall System Architecture of KRISYS

From a conceptual point of view, there are three orthogonal ways of looking at KBMS [BL86,Ma88b], corresponding to the different kinds of requirements that should be supported by these systems: the needs of the applications (i.e., knowledge manipulation means for solving problems), knowledge engineering support (i.e., modeling concepts for KB construction), and suitable resources and implementation aspects (i.e., mechanisms for efficiently coping with knowledge storage and retrieval). The support of these three classes of requirements leads to a natural division of the KBMS architecture in three layers, which were denoted in the KRISYS project as application, engineering, and implementation layer [Ma89] (Figure 1a).

KRISYS follows this conceptual architecture of KBMS, refining it in order to become suitable for a workstation/server environment: The implementation layer is divided into the working-memory system residing at the workstation (together with the application and engineering layer components) and the DBMS kernel managing the KB on the server side (Figure 1b) [Ma88b, Ma91]. Considering the overall system architecture, KRI-



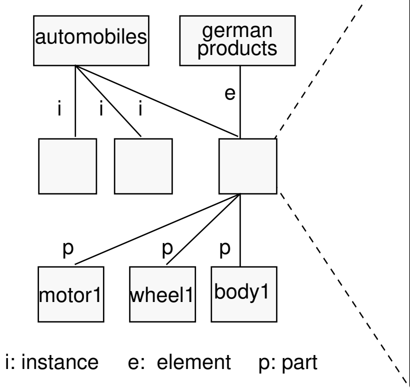
a) conceptual KBMS architecture b) overall system architecture of KRISYS

Figure 1: The KRISYS approach towards KBMS

SYS can therefore be seen as an ‘incarnation’ of the DBMS-kernel architecture for non-standard DBS proposed in [HR85]. In the following, we give a short description of the different system components.

2.2 KOBRA

The KOBRA knowledge model [Ma89], corresponding to the engineering layer, provides an object-centered representation of the application world for the KB designer. It supports the specification of descriptive, organizational, and operational knowledge in an integrated manner. That is, all these kinds of knowledge are incorporated in one basic concept, called **schema** (not to be confused with a DB schema), which is used to represent the entities of the world being modeled. A schema (others call it object) is uniquely identified by a name (or object identifier), and contains a set of attributes to describe its characteristics. Attributes are used for the representation of descriptive knowledge, i.e., properties of a schema and its relationships to other schemas (in this case, they are called slots), as well as for the specification of operational knowledge, i.e., behavioral aspects of an entity (in this case, they are called methods). In order to characterize a schema in more detail, attributes can be further described by aspects (possible-values, cardinality, etc.). For example, the object ‘mercedes-500’ shown in Figure 2 has slots such as ‘price’ or ‘has-motor’, and methods such as ‘order’. The slot ‘price’ is further described by a 'possible-values' aspect and 'unit' as a user-defined aspect.




```
(mercedes-500
    (instance-of              (automobiles)
        (cardinality              [1 1]))
    (element-of               (german products)
        (possible-values          (or (instance-of product sets)
                                      (one-of expensive-cars cheap-cars))))
    (price                    (50000)
        (possible-values          (and (integer) (interval <1000 100000>)))
        (unit                     (US$))
    (producer                 (mercedes)
        (possible-values          (instance-of auto companies))
        (cardinality              [1 1]))
    (sell                     (begin “ transfer price to producer “
                                  “change ownership” end)
        (comment                  (operation to sell the automobile)))
    (has-motor                (motor1)
        (possible-values          (instance-of auto-motors))
        (cardinality              [1 1])
        (diametric-reference      (motor-of))))
```

Figure 2: Example description of the schema 'mercedes-500'

For representing organizational knowledge, KOBRA supports the abstraction concepts of classification, generalization, association, and aggregation [Ma88a] which are incorporated into the model by means of special, system-controlled attributes. That is, these concepts 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 schemas by means of any abstraction concept. Classification/generalization as well as association and aggregation each form a directed acyclic graph 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, for example, represent a class with respect to one object and a set or even an instance with respect to another. In other words, KOBRA supports an *integrated view of KB objects*, i.e., there are no separate representations for sets, classes, instances, or complex objects. For example, ‘mercedes500’ in Figure 2 is at the same time an instance of ‘automobiles’, an element of ‘german products’ and an aggregate consisting of ‘motor1’, ‘body1’, and ‘wheel1’. Therefore, the separation of data and metadata, which is a characteristic of existing data models, is eliminated in KOBRA so that meta-information is integrated into the KB [MM89]. A similar approach can for example be found in the language F-Logic [KL89]. The semantics provided by the abstraction concepts [Ma88a,RHMD87] are guaranteed by the system by means of *built-in reasoning facilities* which also enforce the *integrity constraints inherent in the abstraction concepts* [De90]. The best known of these reasoning facilities, *inheritance*, is built into the classification and generalization concepts, and allows the system to derive the structure of classes and instances based on the definition of their (super-) classes and to control model-inherent integrity by refusing attempts to delete inherited attributes (for a description of other built-in reasoning facilities see [Ma88a, MM89, De90]).

Besides the support of methods, KOBRA provides the concepts of demons and rules for the specification of operational knowledge [Ma89, De90]. This allows the KB designer to utilize *different programming paradigms* (object-oriented, data-oriented, and rule-based) when implementing applications. Demons allow for the attachment of procedures to attributes, which are (similar to triggers in DBMS) automatically activated when the attributes are accessed. General reasoning facilities are supported by rules defined in terms of conditions (if-part) and actions (then-part), which are specified by means of KOALA (see Sect. 2.3). Rules can be flexibly grouped together into rule sets according to reasoning tasks. KOBRA provides methods for forward and backward reasoning which are activated with respect to such rule sets. In order to influence the course of inference processes, the user can specify flexible control parameters, like conflict resolvers, search strategies, termination conditions. Demons as well as rules and rule sets are themselves *represented as objects of the KB* and are organized by means of the abstraction concepts.

Finally, the KOBRA model provides means for maintaining the semantic integrity of a KB [De91]. In order to specify constraints for *attribute value consistency*, the possible values and cardinality aspects can be used to restrict the value domain and number of values allowed for an attribute. More complex constraints can be realized by employing the concepts of rules and demons, which also allows to incorporate reactions on constraint violations. Additionally, the KOBRA model regards *methods as units of integrity*, similar to (nested) transactions in DBMS. If an integrity violation occurring during the execution of a method cannot be resolved, a roll-back operation is initiated for the method. Roll-back continues in a cascading manner up to the top level of the (arbitrarily nested) method invocation, unless some method within the hierarchy requests to handle integrity violations internally. If so, the roll-back operation is terminated at this point, and control is returned to the requesting method.

In order to support not only the description of an application model using the above mentioned concepts, but also the process of application development, KRISYS allows the *interactive construction of a KB in a stepwise fashion* [MM89]. After each design operation (e.g., the definition of a class, the reorganization of a class hierarchy, the creation of an attribute, etc.) immediate feedback is provided in the sense that the consequences of a design decision (as, e.g., the inheritance of attributes to existing classes or instances) are directly reflected in the KB state. At any point during the design process, the KB designer may validate the application model by performing operations on the knowledge base (e.g., by activating methods or starting a reasoning process). A *design environment* [Kr89] offers the possibility to save design states and restore them later in case some design decisions turn out to be wrong, and allows the user to test operational knowledge (as for example methods or demons) without having to fear an erroneous behavior of them.

2.3 KOALA

KRISYS provides as its user and application interface KOALA, a high-level, descriptive language for retrieving and manipulating KB contents. (A detailed description of this language is given in [DLM90].) Information is retrieved from a KB using the ASK statement. For example, the statement

```
(ASK   ((?car SLOTS price))                                  <------ Projection-Part
       (AND (IS-INSTANCE ?car automobiles *)
            (IS-IN motor-1 (SLOTVALUES has-motor ?car))))    <------ Selection-Part
```

retrieves the names and prices of all instances of ‘automobiles’, which have ‘motor1’ as motor. The selection part of the ASK statement is expressed as a formula using logical connectives and predefined predicates (e.g., IS-INSTANCE) and functions (e.g., SLOTVALUES) that embody the semantics of the KOBRA model. *Set-oriented queries* are specified by using so-called query variables (e.g., ?car) in the selection formula, which are then instantiated during query evaluation. The variables may be used in the projection clause in order to precisely describe the desired information.

It is also possible to specify *implicit and explicit joins* in a selection by nesting SLOTVALUE-functions and using more than one query variable. Furthermore, queries may involve additional logical connectives (disjunction, conjunction), negation, quantifiers, as well as special predicates and functions for expressing *(generalized) transitive closure* queries, making the language in some aspects more powerful than SQL [DLM90].

The TELL statement is used to manipulate KB contents. For example, the statement

```
(TELL   (IS-ELEMENT ?car expensive-cars 1)                    Assertion-Part
        WHERE
        (AND (IS-INSTANCE ?car automobiles *)
             (> (SLOTVALUE price ?car) 20000)))               Selection-Part
```

will make sure that all cars costing more than 20.000 dollars are direct elements of the set ‘expensive-cars’. Again, *set-oriented changes* are easily achieved using corresponding query variables in the selection and the assertion part. Note that changes are not specified in terms of insert, update, and delete operations, but in a *state-oriented* manner by describing the goal state of the KB in the assertion part [Ma89, DLM90]. It is the task of the system to figure out how to achieve this goal. For example, the above statement may involve the insertion of objects into a set, as well as the creation of the object ‘expensive-cars’ if it does not exist already. Moreover, (parts of) the goal state may already be contained in the current state (before the execution), so that there might be no changes necessary at all. Thus, state-oriented changes free the user or application from knowing the exact state of the KB, when specifying updates. However, only a subset of KOALA can be used to describe the goal state in the TELL statement so that ambiguities can not arise. TELL statements may also contain *multiple assertions* within the same statement. Additionally, it is possible to *access meta-information* within both ASK and TELL statements through special predicates and functions. (These two latter issues also make KOALA go beyond SQL [DLM90].)

In KRISYS, *rules are defined by a TELL statement*, meaning that the rule condition part (the if-part) corresponds to a TELL selection, and the conclusions (the then-part) to the TELL assertions. KOALA is therefore the language used to write queries as well as to specify rules.

2.4 The DBMS Kernel PRIMA

The DBMS kernel chosen for KRISYS, named PRIMA (PRototype Implementation of the MAD model), concentrates on efficient and reliable KB management on secondary storage at the server side and provides application independent data management functions at its interface. PRIMA 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 [Mi89a] are called 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 of 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.

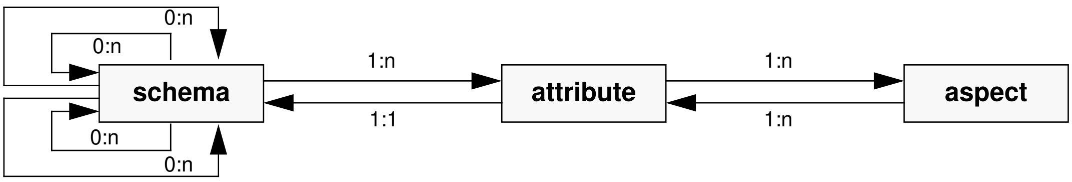


Figure 3: MAD-schema diagram

KOBRA knowledge structures (i.e., schemas representing real world objects, attributes expressing their properties and relationships, and aspects describing the attributes) are mapped to MAD in a straightforward manner [Mi88,Ma90a]. The MAD schema contains three atom types (‘Schema’, ‘Attribute’, and ‘Aspects’) connected via the references (i.e., relationships) has_attributes and has_aspects[2]. The abstraction relationships are represented as recursive MAD references involving the atom type ‘schema’ (see Figure 3).

2. The MAD schema can be seen as a kind of meta-schema, reflecting the basic modelling concepts of KOBRA, and not the application domain (i.e., specific classes, sets, etc.)

2.5 The Working-Memory System (WMS)

The task of the WMS [LM89] is the support of a processing model based on the ‘nearby application locality’ concept. The WMS provides two mechanisms in order to fulfil this task.

First, it maintains an application buffer called Working Memory (WM), which temporarily holds the objects being used by the application in order to avoid long execution paths of KB accesses (involving workstation/ server communications) as well as time-consuming requests to secondary storage. The WM represents KOBRA objects in a format similar to the MAD structures described above (i.e., separated in ‘schema’, ‘attribute’, and ‘aspect’ parts) and offers access to the stored objects via hash tables.

Second, the WMS provides the concept of processing contexts representing the knowledge needed by an application during a specific processing phase. In principle, contexts are composed of several KB objects (in general of different types) and objects may be elements of several contexts. They have to be defined by the KB designer and are represented explicitly as KOBRA schemas and specified by means of KOALA. During application processing, the WMS exploits the specification of a context to generate a complex set-oriented access to the kernel in order to prefetch required objects and store them in the WM as a new application processing phase is entered. Thereafter, most or perhaps all objects referenced during the processing phase are found in the WM, 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 WM and the context requested by the following phase is loaded.

3. What have we learned?

The KRISYS system became fully operational in 1989, when a number of applications from different areas began to be developed (see Table 1 for examples).

Title	Application Area / Problem Class	References
MED2 XPS-shell [Pu86]	Diagnosis	Mi89b
XPS for Trip Planning	Classification	Rh89
Real-Estate Valuation Support	Decision Support in Finance	Mö91
‘Intelligent’ CAD	Architectural Design	DHMM89, MDL91, Th90
TechMo	Mechanical Design	DHMS90, DHMS91, Du91
Mapping Generator	Physical DB Design	Sch91, Su91, Kn92
3D Objects	Spatial Reasoning and Integrity	Sch89, Mö90
Restaurant	(sample KB for practical course and demos)	DD91
Dialogue Component Tool	XPS	DK91
Integrated Product Model Environment	Design	Re90
Publication Management	Information Retrieval	Kr90
Multi-Media Application	Multi-Media	Zi91

Table 1: Applications of the KBMS KRISYS

While some applications were restricted in functionality, size, and depth and were developed either for demonstration purposes or to evaluate specific modeling concepts, others (esp. the first six applications in Table 1) can be more or less seen as ‘realistic’ in the sense that they cover a ‘real-world’ problem in sufficient depth and generality. The variety of realized applications mentioned above on one hand allowed to validate in how far the modeling and processing requirements of the applications were met by KRISYS, and on the other hand permitted to evaluate the techniques for supporting an efficient overall processing within a workstation/server environment. Since these aspects do not primarily depend on an increased KB size, we usually refrained from populating the KBs with a large amount of data, but concentrated on the realization of the overall application functionality. A fairly large amount of data/knowledge was only collected once for a KB for car repair diagnosis [Bo83] with the MED2 XPS-shell [Pu88], which contained over 2500 deductive rules, besides a larger number of other objects, representing symptoms, diagnoses, technical analyses, etc.

In the following, we will present our evaluation by stating the concrete ‘lessons’ we have experienced. The lessons are grouped into subsections according to the main topics of our evaluation. While the first group covers representation and application processing aspects, the second is mainly concerned with our experiences in providing a modeling tool functionality in addition to the support of knowledge management tasks. The third group contains the lessons about the overall concepts for knowledge processing in KRISYS w.r.t. a workstation/server environment.

3.1 Application Modeling and Processing

Modeling Concepts

Lesson 1: The support of modeling constructs for representing descriptive, operational, and structural knowledge is essential [DHMM89].

Besides the concepts for representing descriptive aspects of the application domain (i.e., schemas and slots), constructs for specifying structural and operational information (i.e., the abstraction concepts as well as methods, demons, and rules) were widely used in all KRISYS applications[3]. This observation, which comes to no surprise, again emphasizes the need for advanced data and knowledge modeling facilities in order to support the construction of more precise and complete application models.

Lesson 2: The four abstraction concepts (classification, generalization, association, and aggregation) are at the heart of knowledge modeling [MM89].

The complete support and the clear distinction of the abstraction concepts has turned out to be the fundamental concept for modeling our application KBs. Especially the distinction between the notion of a class (generalization) and a set (association), and the support of an integrated view on the abstraction concepts, i.e., the possibility to model an object as playing different roles (e.g., a class, a set, or even an instance) at the same time, have helped to improve the clarity of the application domain description[4]. Without an integrated view, redundancy would have to be introduced into the KB (e.g., different schemas for representing the object as a class, set, and instance) and would have to be handled by the application.

Operational Concepts

Lesson 3: The (sum of) constructs for operational knowledge should be computationally complete.

When looking at the ‘amount’ of application processing realized within the representational framework of KRISYS, compared to the processing performed in an application program outside the KB, significant differences between the applications became apparent. While in some applications only basic processing tasks were performed in the KB, others were realized completely with the modeling constructs of KRISYS. The question ‘Where does the KB end and where does the application program begin?’ can only be answered by the KB designer. Therefore, a KBMS must not anticipate the answer by limiting the expressive power of its operational concepts (e.g., to sequences of DML statements), thereby restricting the amount of application processing that can be integrated into the KB.

Lesson 4: A KBMS should support different programming paradigms.

We had chosen to support in KRISYS several concepts for representing operational information (methods, demons, and rules), which allow the application designer to use different programming paradigms (object-oriented, data-oriented, and rule-based) when realizing the application. In our evaluation we found out that a ‘typical’ application (w.r.t. the paradigm(s) used) does not exist. Methods were used in many applications, but were usually combined with either demons or rules, or both. As certain paradigms may be more suitable for certain problem classes than others, a KBMS should allow the application designer to flexibly choose the paradigms that fit best.

3. Note, that this is not necessarily the case for the operational aspects, since an application designer might choose to dispense some of the operational constructs of KRISYS completely, and realize application processing outside the KB by embedding KRISYS statements for interacting with the KB in application programs written in a general programming language (CommonLisp).

4. Note, an integrated view does not contradict the clear separation of distinct abstraction concepts [Ma88a, MM89]!

Integrity Constraints

Lesson 5: A uniform, powerful and flexible concept for modeling integrity constraints must be provided [De90].

Our applications, especially the ones from the area of design such as the intelligent CAD system and TechMo, posed ambitious requirements w.r.t. constraint modeling. In these systems, various kinds of application constraints ranging from physical laws to legal regulations and (user-defined) design restrictions (i.e., goals) had to be represented, requiring an increased flexibility, concerning

- the constrained objects, i.e., if a constraint should be valid for all objects of a certain class, or only for specific instances (like a design goal),
- reactions on constraint violations, which have to include corrective actions, user interaction, or even tolerating/recording the inconsistencies, besides the (usual) roll-back operation,
- the specification of checking operations and reactions, which can sometimes become very complex, making procedural specifications necessary,
- the time of activation, which must not be restricted to the immediate/deferred activation scheme, but should be relatable to arbitrary units or levels of integrity, as well as
- the 'usage' of a constraint: While in some cases a constraint has to be used to check consistency (e.g., of a design object), the same constraint is used in a different situation in a rule-like fashion to derive or compute new information (e.g., if a design object is not completely specified by the user)[5].

In all, the KRISYS concepts used for constraint modeling (i.e., special aspects, demons, rules) left a lot to be desired w.r.t. the above requirements. Additionally, the fact that several modeling concepts can (and sometimes have to) be used for representing constraints, which sometimes have different notions of activation, possible reactions, etc., rendered the modeling of an application more difficult (from an integrity point of view).

Lesson 6: Low-level concepts (e.g., triggers, demons) do not offer a satisfactory modeling of constraints.

Although event-based or trigger-like concepts usually provide the flexibility required to fulfil several of the above mentioned points, they are not suitable as modeling concepts for constraints, but should only be seen as an internal basis for their implementation. As the number of constraints of an application exceeded a certain limit, the semantics of the set of demons used for their realization was hard to capture. Several demons were necessary to realize a single constraint, and the overall effect of their activation was hard to foresee.

Lesson 7: An object-centered framework offers various opportunities for the integration of concepts for integrity constraints into the knowledge model [De91].

The decision to represent constraints uniformly as special (first-class) objects of the KB turned out to be especially good. This approach offered the advantage that constraints were represented explicitly and could be organized in the KB according to various criteria, using the abstraction concepts. In KRISYS, we have represented demons as well as rules in this way. In our applications, we could therefore easily locate certain constraints, and modify or extend them. Also, the ability to introduce new attributes for the constraints as an additional description turned out to be a useful selection criterion for querying the KB w.r.t. its constraints, which contributed to increase the modifiability of the constraint set.

Another opportunity was given by the possibility to associate the activation of integrity constraints with the execution of methods, allowing a flexible specification of activation time. In this context, the methods could be seen as (arbitrarily nested) units of integrity, allowing the realization of various integrity levels (for a detailed description of how methods and demons can be used to model different levels of integrity, see [De91]). Moreover, update operations in object-oriented systems usually involve only single attributes (and not complete tuples or records as in relational systems). This granularity of updates, which is consequently reflected in the definition of update events, is more suitable for initiating integrity checks than in conventional DBS, because most constraints are specified for (combinations of) attribute values.

5. This requirement questions to some extent the difference between rules and integrity constraints, which is usually made in KBMS!

Query Language

Lesson 8: Access to KB contents should rely on a declarative, set-oriented query language.

KRISYS provides two separate external interfaces[6]: KOALA, a declarative query language, and a functional interface embedded in LISP[7], which offers a predefined set of functions for basic interactions (e.g., accessing slotvalues, creating and deleting objects, calling methods, etc.) based on object identifiers (i.e., schema names). While KOALA can be used for ad-hoc queries and for KB interaction within methods or application programs in a declarative fashion, the functional interface is more suitable for programming in a more object-oriented, navigational style.

Our experiences concerning the usage of these interfaces were the following:

- The ‘navigational’ interface proved to be appropriate and sufficient for some, but not all types of application methods. Obviously, associative access could be addressed only inadequately, therefore requiring a declarative language.
- Even in cases where the navigational language was sufficient to implement some tasks, it required quite complex programming efforts that could be expressed in KOALA without any difficulties, leading to simpler and more readable method definitions. We observed that specially for this purpose the features of KOALA that go beyond SQL [DLM90] (e.g., multiple updates on multiple classes within one single TELL statement) were very useful.
- In spite of the ‘superiority’ of KOALA when compared to the navigational interface, KB designers started realizing methods using only the navigational language and often stayed with this interface as long as possible (some applications were even realized without using KOALA at all [Th90]). Basically, the following two reasons were given by the KB designers to justify this. First, getting familiar with a new language based on a different semantics or paradigm than the procedural one requires efforts that should not be underestimated. However, in cases in which the KB designers became familiar with KOALA, they appreciated its advantages and found the use of KOALA in methods more appropriate then using the navigational interface. Second, KOALA was conceived primarily as an ad-hoc query language, and little effort was put into the development of a satisfactory coupling or integration with the host language (i.e., CommonLisp). Therefore, the well-known problems of impedance mismatch between query and host language occurred, and additional programming efforts were necessary to interpret the results of an ASK statement within the Lisp program or to pass parameters into a TELL statement, thereby discouraging the use of KOALA in methods.

All in all, our experience proved that the support of a declarative, set-oriented query language in a KBMS environment, even in combination with an object-oriented representation, is necessary and useful[8]. However, significant efforts have to be put into an appropriate integration of declarative and object-oriented programming languages in order to harmonize these two paradigms as far as possible.

Lesson 9: For using a query language as a basis for rule definition, a state-oriented language semantics is desirable.

Another argument for supporting a query language is that a declarative query language can resemble an adequate basis for the definition of deduction rules in a KBMS environment. In contrast to deductive DBS where rules can only be used to derive intensional information, rules in KRISYS can also be used to modify the extension of the KB upon their activation. For this purpose, the notion of state-oriented changes as supported by KOALA, which was briefly introduced in Section 2, has shown to simplify the specification of rule-based systems significantly [DLM90, Rh89]. Without state-orientation, one would have to clearly identify in the conclusion-part of the rules the exact operation that has to be performed on the KB upon rule activation (e.g.,

6. The reason for providing two alternative interfaces is mainly a historical one: The implementation of KOALA was completed after the ‘object-oriented’ interface had already been realized. Nevertheless, this double effort of implementation gave us the opportunity to provide completely different interfaces for the KB designer, and the chance to find out which one is more suitable for his/her purposes.
7. This interface is also provided as part of a graphical environment running on X-Windows.
8. Note, that these observations have only considered the needs of the users and KB designers. Efficiency considerations will be discussed in subsequent parts of this section.

insertion of a new object or modification of an existing one)[9]. Because this operation is dependent on the KB state at the time of activation and is not necessarily known when the rule is being specified, it leads to the multiplication of the number of rules: Predicates for determining the actual state of the KB (e.g., existence or inexistence of an object) would have to be included into the condition part in order to apply the correct operation, causing additional rule definitions for each distinguished state.

3.2 Knowledge Modeling vs. Knowledge Management

Support of modeling activities

Lesson 10: Support of application development and processing within the same framework is necessary.

KRISYS was constructed to appropriately support both the design and the operation of applications within the same representational and operational framework. The possibility to develop the application in a stepwise fashion and to immediately validate design decisions was greatly appreciated by all application designers. Moreover, we learned during several practical courses based on KRISYS that the possibility to interactively develop and run applications facilitated getting acquainted with the system and sped up the development of these applications. Instead of being forced to define the structure of a knowledge base before actually working with the system, users could immediately try out the features of KRISYS without any previous specification.

Lesson 11: The KB design process has to be supported by an appropriate system environment that enables (re-)designs, taking back design decisions, saving and restoring design states.

Throughout the application developments, we made the experience that the user should not be forced to definitively integrate his design decisions or even whole design steps into the KB because in a lot of cases he/she is not yet sure about the correctness of his/her decisions. KBMS should allow an easy reverting of such activities whenever necessary.

To this end, KRISYS offers the design environment [Kr89], which proved to be essential in the design of our applications. When design decisions turned out to be wrong, a consistent, earlier design state could be restored by simply going back to the corresponding KB state. Moreover, this feature allowed the user to validate operational knowledge, as for example methods or demons, without having to fear an erroneous behavior of them. By allowing the user to save the state of the KB prior to execution, the design environment relieved the user from the gravity of decisions and allowed him or her to use KRISYS more freely as a modeling tool. Without such kind of support, the designers would have been forced to either keep copies of the complete KB or issue complex compensating actions to restore a consistent KB state in case of an erroneous action.

Lesson 12: The dynamic behavior of the abstraction concepts, providing built-in reasoning and consistency, proved to be the basis for stepwise, interactive KB design [MM89].

The abstraction concepts are the means to organize a KB, i.e., to define its structure. As already pointed out in Section 2, built-in reasoning facilities (e.g., inheritance) and consistency conditions are associated with these concepts, describing their semantics. In KRISYS, the semantics of the abstraction concepts is guaranteed dynamically after each operation, which turned out to be very effective for supporting an easy modeling of a KB. As the designers changed abstraction relationships, defined new attributes, and created or specialized aspect definitions, the built-in reasoning facilities were automatically activated to reflect the consequences of such operations (e.g., changes in the object structure through inherited attributes) or detect and prevent inconsistencies, such as cyclic abstraction relationships or inheritance conflicts. This behavior proved to be essential for an immediate validation of design decisions.

Lesson 13: Eliminating the difference between regular and meta-information is advantageous for supporting modeling activities.

In KRISYS, this elimination, which is reflected in the knowledge model as well as in the query language, has shown to be an important support for KB design in several ways. First of all, the integration of meta-information and 'regular' information provides the KB designer with a uniform operational and representational plat-

9. This type of definition is comparable to the specification of the action-part in ECA rules or triggers.

form for defining the structure of a KB as well as for effectively validating and testing applications. In KOBRA, the schema is the representational basis for structural information (e.g., a schema as a class) as well as for regular knowledge (e.g., a schema as the instance of a class), and all operations are carried out on schemas of the KB. Second, the integrated access to meta-information within the query language KOALA provides a useful means for retrieving meta-information and reorganizing KB contents during KB design. KOALA turned out to be a good support for the designer when he/she needed to keep track of the current KB structure or to perform restructuring measures.

Additionally, the implementation of the design environment was greatly facilitated by the fact that meta information is stored in the KB and can be treated just like regular knowledge. We could easily use the mechanisms already implemented for effective data management (like transactions and savepoints) in order to keep track of design decisions and to make them revertible.

Relationship between Design and Operation Phase

Lesson 14: Design phase and operation phase can be distinguished [Ma91].

XPS tools (e.g.,KEE [FK85,Fi88], Knowledge Craft [FWA85]), which support applications comparable to those of KBMS, assume that an application is subject to design changes continuously throughout its life time. For this reason, these systems do not distinguish between the design phase and the operation phase of an application. Contrary to this assumption, we have found that for the applications we have developed with KRISYS, it was reasonable to distinguish between the two phases since there was a dramatic shift of interest when moving from one phase to the other. In the design phase the KB designer(s) were interested in flexibly structuring the application. When processing (parts of) the application in this phase, he/she was interested in the validation of the KB, aiming at the redesign of the KB if the application did not exhibit a desired behavior. In the operation phase, the user of the application (who was in most cases not the KB designer) was interested in an efficient processing of the application. Therefore, in each application we developed, we observed a certain point at which the design phase ended and the operation phase started because the KB designer became satisfied with the behavior of the application. At this point, efficiency, and not flexibility, became the major issue (see also lesson 16).

Lesson 15: A strict separation of the design phase from the operation phase without the possibility to perform redesigns is however undesirable.

During the complete life-cycle of our applications, we have observed points, where even in the operation phase a reconsideration of the KB structure was necessary (e.g., new requirements became apparent or structural changes in the KB became necessary). In such situations, a return to the design phase was then needed. However, these situations were not frequent (compared with the iterations in the initial design phase) and usually had only limited effects on the running application, leaving large parts of the structure of the KB untouched. Nevertheless, they were enough to show another argument for integrating the functionality of both a knowledge modeling tool and a knowledge management system within the same environment. A strict separation, as prescribed in DBMS by the need to define and maintain DB schema and actual database in different locations or even in different representational frameworks, is a severe obstacle in the fulfillment of the above stated requirement.

Lesson 16: Design and operation phase differ fundamentally in the kinds of operations applied [Ma91].

We have performed an in-depth analysis of the different phases of the applications' life cycle w.r.t. to the operations performed during each phase. Thereby, we observed that during the design phase operations provoking changes in the KB structure were prevailing, as for example the generation of classes, the creation of attributes, the restructuring of sets or the definition of aggregation hierarchies. Also, the operational knowledge represented in the KB (i.e., methods, rules, demons) was frequently changed until it was free of errors and exhibited the desired behavior. On the other hand, during application processing (i.e., in the operation phase) these kinds of operations occurred only very rarely (or not at all). In this phase, objects were selected by conditions based on the value of certain attributes or on their relationships to other objects. Attribute values were manipulated by the system, instances and elements were created and/or deleted [Sch91]. Also, operational constructs (e.g., methods) were activated, but only rarely modified. Therefore, the flexibility needed during the design phase was no longer required in the operation phase. This observation to some extent contradicted the assumption made by XPS tools, as stated in lesson 14.

Lesson 17: After the design phase is finished, optimizations have to be performed.

As demonstrated above, the operational phase does not require the functional flexibility of the design phase because the operations performed in these two phases differ. Therefore, we learned that switching from the design phase to the operational phase also allows changing the emphasis of system support to improve performance. To reach this goal, an additional phase, the so-called optimization phase, has to be introduced between design and operational phase, which can be initiated explicitly by the KB designer at the end of the design phase. A prime candidate for an optimization to be performed in this phase is the optimization of the mapping scheme used to map KOBRA to MAD because it greatly influences the performance of the operations of the application. We observed that the mapping described in Section 2.4 (Figure 3) was ideal for the design phase, since any design operation, even structural changes, did not provoke any changes in the MAD schema, but only the insertion, deletion or modification of tuples (instances) in one (or several) atom types. However, this kind of mapping caused a significant overhead for select operations (which were prevailing in the operation phase) since each object had to be constructed from its components (attributes and their aspects), leading to a rather complex query even if the object was selected by means of its object identifier. In the optimization phase, the system should therefore be switched to a more specific mapping, which reflects the application domain and does not exhibit the above described drawbacks.

However, changes on the mapping between KOBRA and MAD are not the only optimization actions that can take place after finishing the design phase. We also concluded that it was then possible and necessary (for performance reasons) to compile and therefore optimize all operational concepts used in the application, thereby going out of an interpretative mode into a compiled and consequently more efficient mode. For example, queries contained within methods should be optimized and compiled. Integrity constraints should be transformed into an internal format and necessary integrity checks for update operations should be generated and placed into the code of methods which perform the corresponding updates. Rule processing should be supported by means of building dependency graphs for each ruleset as well as improved by compiling and optimizing the rules themselves.

KRISYS clearly failed to fulfil the requirements formulated in this lesson because none of the above described means for optimizations were supported.

Lesson 18: The structure of the KB and the processing characteristics of each application are application dependent and determine an optimal mapping scheme between KOBRA and MAD [Ma91].

A thorough analysis of our applications has shown that the structure of their KBs differed very much. They differed for example in the number of hierarchies, in the kind of abstraction hierarchies (generalization, association, aggregation), their height, their shape (only trees or even graphs), in the distribution of the instances in the hierarchy, etc. [Sch91, Ma91]. Furthermore, we observed that the processing characteristics of the applications were also very different. They showed differences in the programming paradigm they were using (object-oriented, data-driven, rule-based, hybrid), in the exploitation of context definitions, in the kinds of queries, frequency of updates, existence of schema evolution, etc. We finally observed that all the characteristics strongly influenced the shape of an optimal mapping scheme between KOBRA and MAD.

Lesson 19: KBMS cannot be based on a fixed mapping scheme, but have to be able to handle distinct, application-dependent mappings [Ma91].

Based on the above lessons, we could then conclude that the mapping between KOBRA and MAD should be adapted to the requirements of each specific application after the completion of the design phase, when such requirements are well defined because the KB structure as well as the processing characteristics of an application are application dependent. Naturally, in order to exploit specific mapping schemes, the MAD schema and the corresponding transformation process of KOBRA objects to MAD atoms must not be fixed to a certain mapping scheme, but have to be adaptable to distinct, application-dependent mappings. Therefore, the optimization phase has to support the generation of an application-specific mapping and allow the transformation of the KB into the new format.

Lesson 20: Most of the work done during in the generation of an appropriate mapping can be done by the system during the optimization phase.

The main goal of the optimization phase is the generation of an application-oriented and consequently efficient mapping scheme for a certain application. Once an appropriate mapping has been found the transformation of the KB into the derived MAD schema can be performed automatically by the system. We had experienced in our attempts to find optimal mappings for our applications that this task involves the consideration of a lot of information and requires complex decisions to be made. However, most of the information needed for this task can be derived by the system because it is already somehow stored in or can be inferred from the KB, mainly from information about the KB structure (e.g., the number of hierarchies, their height, their shape, etc.). Also, some information concerning the processing characteristics of the application can be derived by the system from the rules in rulesets, from the kinds of demons, their activation time, etc. Only information not represented in the KB like the number of expected instances per class, elements per set, the relevance of membership stipulations for access purposes, etc., has to be specified by the knowledge engineer. Therefore, the system is able to perform most of the work necessary to generate an optimal mapping. However, because of the large amount of relevant information and the enormous number of potential mappings that can be generated, one cannot expect that a fixed procedure can be followed by the system in order to derive the appropriate mapping for an application. The system has to make use of a number of existing heuristics to automatically derive an adjusted MAD schema for each application.

3.3 Overall Processing Support of KRISYS

General Architectural Considerations

Lesson 21: A layered system architecture with well-defined interfaces is useful and facilitates physical distribution of system components.

Considering the functionality provided by KRISYS, the system consists of several hierarchically ordered layers, thereby following general design rules known from existing DBMS [HR85]. The advantages of this architectural approach (ease of implementation, maintenance, and extension) became apparent during the implementation of KRISYS, especially because the system was continuously under extension and improvement . In the special case of KRISYS, the approach also enabled us to design the system in such a form that the layers of the system naturally reflected the three major classes of requirements of KBMS. All in all, it permitted us to realize well-defined interfaces providing modularity, data independence and extensibility in the various layers. It also facilitated the assignment of system components to workstation and server.

Lesson 22: 'Loose coupling' of workstation and server components with the support of locality of reference is necessary to reduce communication and transfer overhead.

As mentioned in Section 2, the KRISYS architecture was designed to fit into a workstation/server environment with decentralized and autonomous processors. In such an environment, the kind of interaction between the server and the client is a very important design issue [Hä89, Ma90a]. Regarding this issue, we learned that a close 'coupling' between server and workstation results in a huge amount of communication between the server and the workstation component combined with high transfer cost, since every user action can provoke 'thousands' of accesses to the KB. Furthermore, this kind of coupling results in a failure dependence, which is a very critical design issue, because the users may be affected by any kind of failure throughout their typically long-term activities. Therefore, we strongly believe that the only reasonable approach for a system like KRISYS is to rely on loosely coupled system components with interfaces that minimize communication traffic and KB accesses. We learned that this approach can only be successful if accompanied by the support of a high degree of locality of reference on the workstation side for performance reasons calling for an application buffer in the workstation, as will be discussed in lesson 25.

Lesson 23: In order to exploit the advantages of a workstation/server environment, knowledge model semantics and application-oriented processing should be shifted towards the workstation [De91, Ma91].

Considering the constructs underlying KOBRA at the workstation and the MAD model at the server, 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 conclude that this was a problem and try to come up with the idea of enhancing MAD with KOBRA semantics to equal both models. However, we have learned that efforts to equal MAD and KOBRA do not promise to offer solutions (see [De91, Ma91] for a detailed discussion on this topic). By placing the 'borderline' between workstation and server on the enhanced MAD interface, all the functionality of KOBRA would be completely delegated to the server, 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 the 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. Consequently, only basic, application independent tasks should be delegated to the server component. The server has to treat knowledge structures simply as a kind of network of complex objects to be consistently, reliably, and efficiently managed. It has to provide flexible means to select sets of objects and transfer them to the workstation or to write them back into the KB. In summary, the division of the system architecture of KRISYS with KOBRA and KOALA at the workstation side and the DBMS kernel at the server side proved to be correct.

Lesson 24: The NDBS-kernel approach is the appropriate architectural environment for KBMS.

The architecture of KRISYS follows the principles of the NDBS-kernel approach that has been proposed in [HR85]. The proposal claimed that NDBS should be constructed by a loose coupling of workstation and server, implementing the functionality required by a specific application class on the workstation and using the NDBS kernel for general data-management tasks. Although KRISYS is not restricted to an application class but rather offers a general platform for developing applications, it was yet another example of the advantages of the approach mentioned above, as can also be seen from lessons 21, 22, and 23. Thus, we can conclude that the affirmations in [HR85] could be demonstrated in practice by the KRISYS implementation.

Application Buffer

Lesson 25: An application buffer to support locality of reference is a key concept for efficient workstation/server processing.

In all applications developed we have observed locality of reference, i.e., there are many timely related accesses to the same object and even to the same attribute of an object [Ma90b]. This observation, which was not surprising during the operation phase, was also valid during the modeling phase of an application (subsequent modeling activities often involved the same hierarchy and even the same object) and the optimization phase. As already argued in lesson 21, this locality of reference must be supported to improve the performance of the application. For this reason, the integration of the working memory into the KRISYS architecture has proven to be the right decision. We performed several measurements with each of the developed applications and we observed that the use of the WM led to a significant reduction in application run-time by a factor 10 - 30 (i.e., from hours to some minutes). Therefore, it is absolutely necessary to support application processing in workstation/server environments efficiently.

Lesson 26: Pure hashing is inappropriate for accessing objects in the application buffer.

In our applications, parts of an object's structure (e.g., attributes or aspects) were usually accessed several times in the WM within a certain phase [Ma90b]. However, we observed that a pure hashing method for organizing the application buffer did not support these repeated references in a satisfactory way. Instead of being able to exploit the current position of an object for a faster relative access, KRISYS always had to compute the absolute position of the next relevant part of that object from scratch. Moreover, a hash method could neither support navigation through hierarchies nor set-oriented access of related objects. Because we also observed that these two kinds of operations were performed intensively in each application 'phase' (e.g., navigational access to support the inheritance of a new defined attribute to all subclasses and instances and set-oriented access to perform selections on instances of a class and its subclasses), we concluded that hash methods are not sufficient to appropriately support the kinds of accesses performed in the WM.

Lesson 27: An object representation in the buffer closer to the knowledge model and the support of navigational as well as set-oriented processing of buffer objects are essential for efficient processing.

In a first version of the application buffer, we had chosen an internal representation similar to the one used inside the DBMS kernel to speed up the load and unload tasks. No transformation was necessary when objects were brought into the WM or written back. However, since the knowledge model needed a different representation of the objects (both to support modeling activities and query processing [TMMD92]), the objects had to be converted each time they were accessed by a KOBRA or KOALA operation. Measurements showed that this conversion had consumed a lot of time (up to 30% of the workstation processing time). Thus, we concluded that objects should be represented in the application buffer in a form close to the knowledge model of KRISYS, resembling a direct representation of the hierarchies formed by the abstraction concepts (e.g., by means of main-memory links). The kind of representation of the objects stored in the application buffer is very important for the efficiency of the operations of the knowledge model and the processing of queries. Pursuing this observation and the observations of the previous lesson, we can then summarize our conclusions by saying that an application buffer should provide an internal representation of objects that is as close as possible to the knowledge model and that efficiently supports both navigational and set-oriented processing [La91].

Lesson 28: Query processing must fully exploit the application buffer.

We learned from the query processing in KOALA that in order to adequately support the processing of queries in the workstation, the application buffer has to provide means for the explicit representation of results of subqueries delegated to the DBMS kernel as well as intermediate results of the query evaluation (i.e., sets of objects or parts of them). Otherwise, the query processor has to maintain structures containing copies of the qualified objects or object parts leading to a redundant representation of objects in the workstation. Based on these access structures, the application buffer should provide (in addition of both kinds of accesses given in the previous lesson) a powerful, explicit cursor concept to scan over the objects, allowing value-based selections of objects as well as operations concerning an object set as a whole (e.g., sorting objects by some criteria or merging two object sets).

Lesson 29: Support of main-memory indices on the application buffer is needed to increase efficiency.

In our investigations, we also observed (even in the smallest applications) that the 'amount' of data that had to be processed and consequently had to be passed to the workstation during the operation phase was growing rapidly. Supported by modern workstations, which are equipped with several MB of main memory with strong growing tendency, this will cause the size of the application buffer, the number of objects stored within it, and the size of the intermediate results to grow rapidly as well. Thus, the application buffer has to provide a number of different main-memory indices [La91] to accelerate the processing of these large sets of objects by the query processor [Hä91, TMMD92] (e.g., a value-based scan over the objects) similar to main-memory DBMS.

Processing Contexts

Lesson 30: Processing contexts are a useful means for further reduction of server calls through prefetching.

Our measurements have shown that the definition of processing contexts and their exploitation to generate a set-oriented access to the server in order to prefetch the corresponding objects and store them within the application buffer was an adequate means for further improving the efficiency of application processing. The number of server calls could be reduced by a factor of 2-5 and the time consumed by the server component decreased by 15-30% resulting in a 10-20% acceleration of application processing [Ma90b].

Lesson 31: The system should support the KB designer in the definition of such processing contexts by analyzing the queries, operational concepts, and constraints.

The development of our applications also revealed insights regarding the practical handling of processing contexts. We observed that the concept was rarely used by the KB designers although they were the persons who knew best about the details of an application. Even for those experts, it was hard to 'locate' contexts within the application, i.e, objects required in certain phases of the application, as they were often hidden in the specification of queries, methods, problem solving strategies, integrity constraints, etc. For this reason, even when the KB designers specified such contexts, they proved to be inexact, i.e., did not contain all objects really needed in the corresponding phase of the application. A factor that contributed to this problem was that the choice of the most appropriate 'point in time' for the specification of contexts proved to be rather difficult. When defining the contexts during the modeling of the application, they had to be adapted each time the algorithm was changed because of necessary reformulations and redesigns. When defining the contexts after finishing the modeling phase, an additional deep analysis of all parts of the applications was required which was then hard to perform and also very time consuming. However, we realized that most of these contexts could be extracted from queries, rules, and demons, a task of which the system could easily take care. It became then clear that it is possible to support the KB designer in his/her task of defining contexts by providing the functionality to automatically analyze the specification of rules, demons, and queries in methods in order to be able to find out which objects are needed in each phase of an application.

Lesson 32: Internal contexts should be exploited automatically by the system.

Independent of processing contexts that can be derived from the phases of an application, the internal processing of KRISYS also revealed the existence of sets of objects which are needed for processing system operations. During the inheritance of a newly defined attribute, for example, all objects in the generalization hierarchy below the corresponding class were accessed by the system. Also each query and subquery implicitly defined a context, containing the query results. Another example was the evaluation of rules in which every ruleset (the set of corresponding rules) together with the rules themselves (which can be seen as a query) defined a context. The system should, of course, make use of this to automatically define processing contexts in order to prefetch the required objects and to reduce the number of server calls.

Query Processing

Lesson 33: Query processing should rely on an algebraic framework, allowing the exploitation of optimization techniques known from (relational) DBS.

In the first version of KRISYS, KOALA was implemented with the primary goal of making it available for use. The internal representation of KOALA queries permitted only local optimizations and did not allow the use of any standard optimization techniques known from the field of databases. However, a detailed study of these techniques revealed that they are very useful in the KRISYS architecture, if adapted to the issues of a server/workstation environment [Hä91, Ro92, TMMD92]. This made us conclude that KOALA should be reimplemented based on an algebraic approach and oriented at the framework of general query processing [JK84] that permits to apply some of the existing (especially relational) optimization techniques [TMMD92].

Lesson 34: For query processing, a set-oriented, declarative server interface is required, otherwise a preselection of transferred objects is not possible.

KOALA is a set-oriented query language relying on a declarative description of its referenced objects. Due to the expressive power of KOBRA, large 'pieces' of a KOALA query have to be executed in the workstation requiring a large number of relevant objects to be installed in working memory [De91]. Instead of requesting single objects upon reference to the server, a better and far more efficient approach is to load the objects into the WM prior to executing the corresponding parts of the query. For this purpose, a set-oriented, declarative interface to the server DBMS is essential. Only then, it is possible to preselect the objects requested by (some part of) a query giving way to the exploitation of a very efficient two-staged query evaluation (as outlined in [TMMD92]): parts of the query evaluation are performed in the workstation and parts in the server.

Lesson 35: A declarative description of the buffer contents is a prerequisite for optimizing queries in this framework.

Due to the application's locality of reference, some or even all objects needed to process a query in the workstation may already reside in the WM because they were previously referenced by another query. Those objects can directly be used for query processing and need not be loaded from the server. Therefore, the consideration of the contents of the application buffer is an important measure to increase the efficiency of query evaluation. To reach this end, KRISYS should maintain a declarative description of the objects currently being installed in the WM in groups corresponding to the way such groups are referenced by KOALA. Thus, the KOALA processing system could make full use of the contents of the WM. As a simple example consider a KOALA query where all schemas that are elements of a certain set are referenced via the IS-ELEMENT predicate, and the same set has been referenced in a previous query, so that the required schemas are already in the WM. If the WM contents were only described by a list of identifiers of the contained schemas, the query processing component would still have to evaluate the KOALA predicate (requiring access to the server) in order to determine the names of the requested schemas, before being able to decide whether they are already in the WM. However, if the WM contents were described by the IS-ELEMENT predicate (meaning that the elements are completely contained in the WM), this decision could be made by comparing the query predicate with the WM description, requiring no additional server access.

4. Conclusions and Outlook

The KBMS KRISYS - developed at the University of Kaiserslautern - has been operational since 1989. Since then a lot of applications coming from different application classes were modeled with KRISYS to evaluate its adequacy for application design and operation. The experiences we drew from these works were presented in this paper.

We have shown that many concepts offered by KRISYS proved to be well-suited to support the requirements of both the design phase and the operational phase of the applications.

- First of all, this is true for the knowledge representation framework offered by KRISYS. The KOBRA knowledge model, characterized by object-centered approach, allows a uniform representation of descriptive, procedural, and structural facets of an application domain by attributes. Especially, the abstraction concepts of classification, generalization, association, and aggregation are major features for organizing the knowledge base.
- The integration of regular and meta information into the knowledge base made up the second positive experience we had. The difference between structuring and use of a KB on a physical level and on an operational level is eliminated respectively because all information about the knowledge base including schema information is stored there and because the same set of operations applies for manipulating the knowledge base and the meta-information. We observed that this is a major prerequisite to support an incremental development process which is typical for all application classes. Therefore, KRISYS proved to offer integral system support throughout an application's life cycle.
- KOALA, the query language of KRISYS, supports KB access in a set-oriented and declarative fashion and was used in many applications because of its expressive power and its easy use. The latter is due to the fact that KOALA is a state-oriented language that enables the user to manipulate the knowledge base (either by ad-hoc queries or by rules) by only describing the state expected from the result of the operation. Thus, the specification of queries and rules can be done independently from the current state of the knowledge base, resulting in an easier modeling and in a reduction of the number of rules.
- Moreover, the system architecture showed a good support for the applications running on workstation/server environments. The appropriate distribution of tasks between workstation and server and the integration of an application buffer into the workstation enabled most of the semantics offered by KRISYS to be realized in the workstation. The server, hosting the DBMS, is only responsible for providing general management of the data that make up the KB. By integrating an application buffer (the working memory) into the workstation, the applications' locality of reference could be exploited, thus minimizing the commu-

nication between workstation and server. In addition, this architecture makes possible a two-staged processing scheme characterized by first loading the relevant knowledge into the working memory and subsequently processing it there.

We have also demonstrated that our evaluation revealed that some of the features offered by KRISYS are not yet sufficient to support the applications in an optimal way. This especially applies for the optimization of KOALA queries, its embedding into a programming language, for integrity management, for the internal structure of the working memory, and for the mapping scheme used by KRISYS. While we were aware of some of these shortcomings from the start (e.g., those related to the implementation of KOALA), there were several observations that became apparent only over time. For this reason, these experiences turned out to be the most important for us.

- KOALA must be integrated into an algebraic framework that provides the basis for applying query processing techniques from the field of (relational) databases, after adapting them to the 'problems' of a workstation/server environment.
- A new mechanism for supporting integrity constraints that is more flexible concerning the definition of constraints, their scope, and their activation is required to better fullfil the needs of applications.
- The working memory must provide a representation closer to the knowledge model, navigational as well as set-oriented accesses to its contents. All these measures aim at an efficient mapping of higher-level operations (e.g., KOBRA and KOALA operations) on the working memory without the need to perform any transformations on the knowledge residing there or to maintain copies of the objects or intermediate results.
- It is also worth increasing the functionality of the working memory to make its contents available for query processing through declarative descriptions that can directly be exploited by the query optimizer. By doing so, the transfer of information between workstation and server can be deduced by testing whether the knowledge already residing in the application buffer subsumes parts of the knowledge required by a query.
- For the different phases of an application, appropriate mapping schemes from the knowledge model of KRISYS to the data model of the database system must be supplied. These schemes must be based on a detailed analysis of the knowledge base underlying an application. A change in the mapping occurs each time a new design cycle has been concluded and the application is going into its operational phase.

Based on the experiences described in this paper, a new KRISYS architecture has been designed and is currently being implemented. In this new design, we tried to take into consideration all the lessons we learned from the old system. The current state of the implementation is such that the tasks relating to the working memory - except the maintenance of a declarative description of its contents - and the realization of the component responsible for providing and generating appropriate mappings between knowledge model and data model have been completed. The implementation of the algebraic framework for KOALA is currently being undertaken, as well as that of the new component for integrity control. We have also extended KRISYS to support advanced transaction facilities for cooperative design applications [IRLM92]. In addition, we are considering the exploitation of parallelism in the scope of query processing, since the workstation/server environment of KRISYS and the distribution of tasks between those components offer a good framework for this [Th92, TMMD92].

We believe that we have accumulated enough experiences and knowledge from the first version of KRISYS, that it is now time to 'start all over again' to build an even 'better' system that will hopefully show the benefits but not the problems described in this paper. These issues will however be the subject of forthcoming papers.

Acknowledgements

We would like to acknowledge the support of Prof. Theo Härder in giving us the opportunity and necessary support to carry out this research project. Additionally, we would like to thank in particular all the students that have participated in the KRISYS project, whose combined efforts resulted in the KRISYS implementation and in most of the results described in this paper: Ch. Altenhofen, S. Bauer, S. Damaschke, Ch. Differding, K. Duchow, R. Durben, A. Grasnickel, E.Hänsel, M. Höhfeld, S. Hörth, M. Keil, Ch. Knecht, S. Kraft, D. Langkafel, S. Lind, L. Mangels, M. Michels, H. Möllenkamp, F. Mohr, J. Reinert, F. Rezende, B. Rheinberger, N. Ritter, R. da Rocha, P. Schneider, D. Schulte, R. Stauffer, M. Strobel, B. Surjanto, Ch. Thomczyk, and R. Zimmer.

References

At89 Atkinson, M., et.al.: The Object-Oriented Database Manifesto, in Proc. First Int. Conf. on Deductive and Object-Oriented Databases, 1989, Kyoto, Japan, pp.40-57.

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.

Bo83 Borrmann, H.P.: MODIS - An Expert System for Supplying Repair Diagnosis of the Otto-Motor and its Aggregates (in german), Research Report No. 72/83, University of Kaiserslautern, Computer Science Department, Kaiserslautern, 1983.

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

DD91 Damaschke, S., Differding, Ch.: Validating the Modeling Concepts of the KBMS KRISYS using a Sample Application (in German), University of Kaiserslautern, Kaiserslautern - Germany, September 1991.

De90 Deßloch S.: Enforcing Integrity in the KBMS KRISYS, in: Proc. of the 2nd Workshop on Foundations of Models and Languages for Data and Objects, Aigen (Austria), September 1990, pp.123-138.

De91 Deßloch, S.: Handling Integrity in a KBMS Architecture for Workstation/Server Environments, in: Proc. GI-Fachtagung "Datenbanksysteme in Büro, Technik und Wissenschaft", Kaiserslautern, März 1991, Hrsg. H.-J. Appelrath, Informatik-Fachberichte 270, Springer-Verlag, S.89-108.

DFMV90 DeWitt, D.J., Futtersack, P., Maier, D., Velez, F.: A Study of Three Alternative Workstation Server Architectures for Object-Oriented Database Systems, in Proc. 16th VLDB Conf., Brisbane, Australia 1990.

DHMM89 Deßloch, S., Härder, T., Mattos, N., Mitschang, B.: KRISYS: KBMS Support for Better CAD Systems, in: Proc. 2nd Int. Conf. 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.

DHMS91 Deßloch, S., Hübel, C., Mattos, N., Sutter, B.: Handling Functional Constraints of Technical Modeling Systems in a KBMS Environment, in: International Journal of Systems Automation: Research and Applications (SARA), 1, 1991,pp. 347-367.

DK76 Davis, R., King, J.: An overview of production systems, in Elcock, E., Michie, D. (eds.): Machine Intelligence, Wiley, New York, pp. 300-332, 1976.

DK91 Duchow, K. & Keil, M.: A General Tool for the Development of Dialog Components for Knowledge-based Systems (in German), University of Kaiserslautern, Kaiserslautern - Germany, January 1991.

DLM90 Deßloch, S., Leick, F.J., Mattos, N.M.: A State-oriented Approach to the Specification of Rules and Queries in KBMS, ZRI-Report 4/90, University of Kaiserslautern, 1990.

Du91 Durben, R.: Modeling Technical Functional Constraints with KRISYS (in German), University of Kaiserslautern, Kaiserslautern - Germany, September 1991.

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.

Fr86 Frost, R. A.: Introduction to Knowledge Base Systems, Collins, London, 1986.

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.

Hä89 Härder, T.: Engineering Applications - a Challenge for the Next Generation of DBMS, invited talk at the 2nd German INGRES user conference, 1989.

Hä91 Hänsel, E.: Query Processing in the Knowledge Base Management System KRISYS (in German), Diploma Thesis Work, University of Kaiserslautern, Kaiserslautern - Germany, July 1991.

HM90 Härder, T., Mattos, N.: An Enhanced DBMS Architecture Supporting Intelligent CAD (invited lecture), in: Engineering Information in Data Bases and Knowledge Based Systems - TECHNO-DATA'90, (eds. Richter, D. Grabowski, H.), Akademie-Verlag, Berlin -Germany, 1990, pp. 28-50.

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.

HK87 Hull, R., King, R.: Semantic Database Modeling: Survey, Applications, and Research Issues, in ACM Computing Surveys, vol.19, no.3, September 1987, pp. 201-260.

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.: Architektur von Datenbanksystemen für Non-Standard-Anwendungen, in: Proc. GI-Proc. GI Conf. on Database Systems for Office, Engineering and Scientific Applications, p.253-286, Karlsruhe, Germany, März85, IFB 94, Springer Verlag,Heidelberg.

IRLM92 Iochpe, C., Rezende, F. F., Livi, M.A.C, Mattos, N.M.: Implementing a Design Management and Cooperation Model on the Basis of KRISYS, September 1992, submitted for publication.

JK84 Jarke, M., Koch, J.: Query Optimization in Database Systems, in: ACM Computing Surveys, vol.16, no.2, 1984, pp.111-151.

KDE90 IEEE Transactions on Knowledge & Data Engineering, special issue on database prototype systems, March 1990, vol.2, no 1.

KDG87 Küspert, K., Dadam, P., Günauer, J.: Cooperative Object Buffer Management in the Advanced Information Management Prototype, Proc. 13th VLDB Conf., Brighton, England, Sept. 1987, pp. 483-492.

KL89 Kifer, M., Lausen, G.: F-Logic, a Higher-Order Language for Reasoning about Objects, Inheritance and Schema, Proc. of the ACM SIGMOD Int. Conf. on Management of Data, 1989, pp. 134-146.

Kn92 Knapmeyer, Ch.: Optimization of the Database Mapping in the KBMS KRISYS Using Load Information (in German), Diploma Thesis Work, University of Kaiserslautern, Kaiserslautern - Germany, June 1992.

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

Kr90 Kraft, S.: An Analysis of Existing Systems for Knowledge Modeling (in German), Diploma Thesis Work, University of Kaiserslautern, Kaiserslautern - Germany, February 1990.

La91 Langkafel, D.: A Component for Graph-oriented Management of Knowledge Base Contents (in German), Diploma Thesis Work, University of Kaiserslautern, Kaiserslautern - Germany, June 1991.

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.

Ma89 Mattos, N.M.: An Approach to Knowledge Base Management - Requirements, Knowledge Representation, and Design Issues -, Doctoral Thesis, University of Kaiserslautern, Computer Science Department, Kaiserslautern,1989, also appeared as: Lecture Notes in Artificial Intelligence, Vol. 513 , Springer, 1991.

Ma90a 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.

Ma90b Mattos, N.: Performance Measurements and Analyses of Coupling Approaches of Database and Expert Systems and Consequences to their Integration, in: Proc. 1st Workshop 'Information Systems and Artificial Intelligence', Ulm - Germany, March 1990.

Ma91 Mattos, N.M.: KRISYS - a KBMS Supporting Development and Processing of Knowledge-based Applications in Workstation/Server Environments, ZRI-Bericht 5/91, Universität Kaiserslautern, submitted for publication.

MDL91 Mattos, N.M., Deßloch, S., Leick, F.-J.: A Knowledge-based Approach to Intelligent CAD for Architectural Design, in: Proc. IEA/AIE'91 - 4th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, Kauai, Hawaii, June 1991, S. 409-418.

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.

Mi89a Mitschang, B.: Extending the Relational Algebra to Capture Complex Objects, in: Proc. of the 15th VLDB Conf., Amsterdam, 1989, pp. 297-306.

Mi89b Michels, M.: The KBMS KRISYS from the Viewpoint of Expert Systems for Diagnosis (in German), University of Kaiserslautern, Kaiserslautern - Germany, March 1989.

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.

MMM92 Mattos, N.M., Meyer-Wegener, K., Mitschang, B.: Grand Tour of Concepts for Object-Orientation from a Database Point of View, to appear in: Data and Knowledge Engineering.

Mö90 Möllenkamp, H. T.: Ensuring Spatial Semantics of 3D Objects with Semantic Integrity Constraints (in German), University of Kaiserslautern, Kaiserslautern - Germany, August 1990.

Mö91 Möllenkamp, H.: Knowledge based Support for Real-Estate Valuation - Application Analysis, Conception and Prototypical Implementation (in German), Diploma Thesis Work, University of Kaiserslautern, Kaiserslautern - Germany, December 1991.

Pu86 Puppe, F.: Diagnostic Problem Solving with Expert Systems (in german), Doctoral Thesis, University of Kaiserslautern, Computer Science Department, Kaiserslautern 1986.

Pu88 Puppe, F.: Introduction to Expert Systems (in german), Springer, Berlin, 1988.

Re90 Reinert, J.: A Model for Representing Satic and Dynamic Aspects in Design (in German), Diploma Thesis Work, University of Kaiserslautern, Kaiserslautern - Germany, February 1990

Rh89 Rheinberger, B.: A XPS for trip planning as application of the KBMS KRISYS (in German), Diploma Thesis Work, University of Kaiserslautern, Kaiserslautern - Germany, March 1989.

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.

Ro92 Rocha, R. P. da: Transformation and Rewrite in the Query-Processing System of the KBMS KRISYS (in Portuguese), Master Thesis, CPGCC, UFRGS, Porto Alegre, Brasil, May 1992.

Sch89 Schulte, D.: Conception and Implementation of a Knowledge Base for the Representation of three-dimensional objects with the KBMS KRISYS (in German), University of Kaiserslautern, Kaiserslautern - Germany, October 1989.

Sch91 Schulte, D.: An Approach to Flexible Mapping of Knowledge Models to Data Models (in German), Diploma Thesis Work, University of Kaiserslautern, Kaiserslautern - Germany, June 1991.

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.

ST89 Schmidt, J.W., Thanos, C. (ed.): Foundations of Knowledge Base Management, TOIS, Springer-Verlag, 1989.

St92 Strobel, M.: Conception of a Component for Context Management in the KBMS KRISYS (in German), Diploma Thesis Work, University of Kaiserslautern, Kaiserslautern - Germany, June 1992.

Su91 Surjanto, B.: Conception and Implementation of a knowledge based system for the generation of an application oriented DB schema for a KRISYS KB (in German), Diploma Thesis Work, University of Kaiserslautern, Kaiserslautern - Germany, October 1991.

Th90 Thomczyk, Ch.: An Expert System for Design as Application of the Knowledge Base Management System KRISYS (in German), Diploma Thesis Work, University of Kaiserslautern, Kaiserslautern - Germany, January 1990.

Th92 Thomas, J.: An Approach to Parallelism in KRISYS, ZRI-Report 1/92, University of Kaiserslautern, Computer Science Department, March 1992.

TMMD92 Thomas, J., Mitschang, B.,Mattos, N., Deßloch, S.: Knowledge Processing in Workstation/Server Environments - the KRISYS Approach, September 1992 (submitted for publication).

Zi91 Zimmer, R.: Modeling a Multi-Media Application with the KBMS KRISYS (in German), University of Kaiserslautern, Kaiserslautern - Germany, July 1991.