In: The VLDB Journal 7:2, 1998, pp. 79-95

Advanced Data Processing in KRISYS: Modeling Concepts, Implementation Techniques, and Client/Server Issues

Stefan Deßloch[1], Theo Härder, Nelson Mattos[1], Bernhard Mitschang[2], Joachim Thomas

Department of Computer Science
University of Kaiserslautern
67653 Kaiserslautern, Germany
e-mail: {haerder, thomas}@informatik.uni-kl.de

Abstract

The increasing power of modern computers steadily opens up new application domains for advanced data processing such as engineering and knowledge-based applications. To meet their requirements, concepts for advanced data management have been investigated during the last decade, especially in the field of object-orientation.

Over the last couple of years, the database group at the University of Kaiserslautern has been developing such an advanced database system, the KRISYS prototype. In this article, we report on the results and experiences obtained in the course of this project. The primary objective for the first version of KRISYS was to provide semantic features, such as an expressive data model, a set-oriented query language, deductive as well as active capabilities.

The first KRISYS prototype became completely operational in 1989. To evaluate its features and to stabilize its functionality, we started to develop several applications with the system. These experiences marked the starting point for an overall redesign of KRISYS. Major goals were to tune KRISYS and its query processing facilities to a suitable client/server environment as well as to provide elaborate mechanisms for consistency control comprising semantic integrity constraints, multi-user synchronization, and failure recovery. The essential aspects of the resulting client/server architecture are embodied by the client-side data management needed to effectively support advanced applications and to gain the required system performance for interactive work.

The project stages of KRISYS properly reflect the essential developments that have taken place in the research on advanced database systems over the last years. Hence, the subsequent discussions will bring up a number of important aspects w.r.t. advanced data processing that are of significant general importance as well as of general applicability to database systems.

Key words: Object-Oriented Modeling Concepts - Consistency Control - Query Processing - Run-Time Optimization - Client/Server Architectures.

1. IBM Database Technology Institute, Santa Teresa Laboratory, 555 Bailey Ave., San Jose, CA, 95161 USA, e-mail: {dessloch, mattos}@us.ibm.com.
2. Technical University of Munich, Computer Science Dept., Orleansstr. 34, 81667 München, Germany, e-mail: mitsch@informatik.tu-muenchen.de.

I Introduction

1. Advanced Data Management

The increasing power of modern computers steadily opens up new application domains for data processing. As a consequence, the variety and sophistication of database applications is growing rapidly. To meet the requirements of those applications, concepts for advanced data management have been investigated over the last decade, especially in the field of object-orientation. For these developments, mainly two phases can be observed, according to the objectives the corresponding research activities were focused on.

The first stage was primarily concerned with providing powerful modeling features for advanced applications. Knowledge-based systems [In84, JGJSE95] as well as prototypical database implementations [Ma91, KL89] proposed concepts that allowed to adequately model advanced applications. These concepts comprise

- expressive object-oriented data models featuring possibly multiple abstraction concepts,
- powerful query languages tailored to the particularities of the underlying object models,
- constraints and event-based processing, as well as
- deductive capabilities.

While these concepts were being shaped, several data model and language standards arose, prominent examples of which are SQL3 [ISO96], ODMG [ODMG96], or STEP/EXPRESS [ISO94]. Simultaneously, it became obvious that providing optimum support for advanced applications is not only restricted to offering expressive semantic concepts, but that it also requires their effective and efficient implementation. For this reason, research turned its attention to performance aspects of advanced database processing.

This shift of activities, which has taken place in recent years, marks the second phase of research in the scope of advanced data management. During this period, investigations concentrated on processing issues covering aspects as, for example:

- object-oriented query processing,
- client/server processing,
- main-memory query processing, or
- parallelism.

While client/server environments have been widely accepted as being the ideal architectural setting for processing advanced applications [HR85, DFMV90, HMNR95], there are a number of approaches proposing different solutions to performance-critical topics in advanced database processing [HS93, Gr94, LLPS91, CR94]. What is missing, however, is the integration of these isolated approaches into a common implementational framework.

The advanced database system KRISYS (Knowledge Representation and Inference System), which is at the focus of this article, has been developed at the University of Kaiserslautern over the last eight years following exactly this evolutionary pattern. Subsequently, we will give a brief overview of the history of KRISYS.

2. Evolution of the KRISYS Prototype

The overall goal of the KRISYS project is to combine adequate support for the design and operation of advanced applications with database facilities that allow to securely and persistently handle large amounts of data. Accordingly, KRISYS features a client/server architecture providing a client-based infrastructure for application development and processing as well as a centralized database system at the server side. While the design of both processing sites varied in the evolutionary process of KRISYS (cf., the following sections), the overall distribution of

tasks remained unchanged. At the client, applications are supported by an object-oriented data model (cf., Chapter II, Sect. 1) and a user interface featuring a set-oriented, declarative query language. The server, on the other hand, takes over all tasks related to general data management.

In the first project phase, research activities focused mainly on providing expressive semantic concepts that permit to adequately model the data and operations prevalent in advanced applications. Moreover, an essential objective for KRISYS has always been to incorporate uniform support for all phases of an application's life cycle, i.e., for application development as well as for application processing. In this context, schema evolution is an important feature, which has been incorporated into KRISYS from the very beginning of the project.

The first KRISYS prototype became completely operational in 1989. To evaluate its features and to stabilize its functionality, we started to develop several applications with the system. The following table gives an overview of the most important applications of KRISYS. We listed only those applications that cover 'real-world' problems in sufficient depth and generality. About the same number of applications were restricted in functionality, size, and depth and were developed either for demonstration purposes or to evaluate specific modeling concepts.

Application	Application Area / Problem Class
MED2 XPS-shell [Pu86]	Diagnosis
XPS for Trip Planning	Classification
Real-Estate Valuation Support	Decision Support in Finance
Intelligent CAD [DHMM89]	Architectural Design
TechMo	Mechanical Design
3D Objects	Spatial Reasoning and Integrity
Multi-Media Application	Object-Oriented Modeling of Multi-Media Data and Operations
Flexible Product Modeling for an Insurance Company [ST95]	Object-Oriented Analysis and Design, Schema Evolution

Since 1991, the first version of KRISYS has been successfully employed in a practical semester course on object-orientation and knowledge base management systems at our university as well as shipped to other universities for being used in teaching courses and research projects.

The experiences gained in the development of these applications as well as the feedback received from KRISYS users served as a broad and solid basis for evaluating the system from various points of view. While the object model and its features proved to be adequately expressive, the need for an elaborate concept for semantic integrity constraints became apparent. Regarding the conceptual and implementational foundations of processing in KRISYS, we learned that, for performance reasons, we needed a better adaptation of all related tasks to the client/server architecture of KRISYS. Since interactions with users and applications rely on the processing facilities at the client side, those improvements pertain primarily to the client-based functionality of KRISYS, in particular to query processing in KRISYS.

These experiences marked the starting point for a major redesign of KRISYS. The new KRISYS prototype was successfully presented at the 1995 SIGMOD demonstration session [TDM95]. For that purpose we extended the system by some specific explanation facilities in order to highlight important aspects of the internal processing, for example, a graphical view and explanation environment for query processing.

3. Related Work

As already mentioned, KRISYS features an expressive object model that is comparable to object-oriented data models as, for example, the ODMG approach. In addition and in contrast to most other approaches, the KRISYS model integrates active as well as deductive properties in the form of triggers and rules, all needed for a powerful semantic integrity and constraint management. Based on this enhanced object model, KRISYS supports the (evolutionary) process of application modeling and development in the same framework as the data management tasks. This is a distinct characteristic of the KRISYS approach that can only be found in knowledge base management systems as, for example, KEE [In84].

KRISYS provides a set-oriented declarative query language, whose data processing technology is built on well-founded relational query optimization and processing techniques. Hence, from an implementation point of view there is a great similarity to other relational and post-relational database systems as e.g. DB2/6000 [CMCD94], Starburst [LLPS91], POSTGRES [SK91], or EXODUS [CD87]. Even object-oriented database systems are now considering SQL-like query languages, thus evolving to similar directions [ODMG96].

Since KRISYS is conceived for client/server environments with most application-oriented processing being done at the client side, a client infrastructure for efficient data processing is indispensable. KRISYS supports main-memory query processing which asks for run-time optimization to dynamically exploit the client buffer contents at run time, thus minimizing client/server interaction. To the best of our knowledge we don't know of any existing database system employing a similar technology. However, if object-oriented database systems really want to support set-oriented and declarative (perhaps SQL-like) query languages, they have to devise similar concepts in order to provide acceptable performance [FMV94, Ki95, VD91].

Moreover, the KRISYS processing framework supports extensibility at different levels of query processing to cope with later extensions either of the query interface (shifting more application-oriented semantics into the scope of query processing) or of evaluation methods (such as improved join algorithms) [TD93]. Since its processing framework is founded on relational technology, its extensibility technology relies on the relational one as well [Gr93, HFLP89].

Another major design decision has been made to improve the interaction between client and server components. The object-server approach [DFMV90], which turned out to be a performance bottleneck in the first KRISYS implementation, was replaced by a query-server architecture that permits to delegate subqueries. In contrast to traditional query servers, this approach exploits the current buffer contents at the client and improves the balance of processing across the client-server architecture. Moreover, it supports set-oriented retrieval of objects from the server, yet avoids drawbacks encountered with page-server approaches [LLOW91], whose effectiveness strongly relies on appropriate object-clustering mechanisms [DFMV90].

In the database literature, there are only few publications discussing theoretical or implementational aspects of descriptive buffer management. For example, [BJNS94] presents a polynomial-time algorithm for performing subsumptions, while [KB96] describes a query optimizer relying on query-like descriptions for exploiting locality of reference in the application buffer. Recently, however, descriptive buffer management is being paid more attention in conjunction with materialized views and *data warehousing* [Ro91, CR94, Lo95].

Just like in KRISYS, most of those approaches interleave query optimization and matching operations on the cache contents. However, they operate on the relational data model, so that subsumption tests are easier to perform as those required for the KRISYS object model called KOBRA or object-oriented data models in general, and only value-based comparisons must be considered (and not, e.g., transitive abstraction relationships). The commonalities of the approaches like [CR94] to the one of KRISYS lie in the functional organization of query processing and in the access and description facilities provided for cached sets of objects. The

main difference is the fact that [CR94] has to cope with materialized views whose base data may be updated outside the cache causing consistency problems. As the multi-user environment of KRISYS employs an adequate hierarchical lock protocol [RH97], the Context Manager is not faced with the decision on how and when to react to such external updates. As far as multi-user synchronization in object-oriented database management systems is concerned, similar work was performed in the ORION project [GK88]. However, our solution handles more complex situations, since the KRISYS object model allows for different abstraction hierarchies as well as multiple inheritance (multi-class membership) for classes and instances.

4. Overview of the Paper

The rest of this article is made up of three chapters. Chapter II deals with the conceptual foundations of KRISYS. The KRISYS object model, its query language, as well as its overall system architecture are being presented. Moreover, a basic understanding of the tasks of each architectural component is provided and the interaction of these components during query processing is demonstrated using a small example. These discussions mark the starting point for the implementational considerations as being described in Chapter III. Finally, Chapter IV gives a conclusion and an outlook to future work.

II Conceptual View of KRISYS

1. Object Model and Query Language of KRISYS

In this chapter, we summarize the main features supported by the object model of KRISYS, as adopted from the first version of KRISYS. The KOBRA (KRISYS Object Representation) object model featured by KRISYS is comparable to object-oriented data models [Ca91]. A KRISYS object is uniquely identified by a *name* (i.e., object identifier) and is made up of *attributes*, denoting descriptive or organizational characteristics of objects, and *methods* defining behavioral properties. All attributes or methods can be annotated with *aspects* providing additional information, e.g., default values or comments. KRISYS supports the abstraction concepts of classification, generalization, association, and aggregation [Ma91] whose semantics (e.g., inheritance along the classification and generalization relationships) is automatically enforced by the system. Objects are typically organized in hierarchies or lattices defined via those abstraction relationships. For generalization and classification, this means that both multiple inheritance and multiple instantiation (i.e., an object is a direct instance of more than one class) are supported. In addition, the object model of KRISYS provides various other features, as, e.g., constraint and rule management, not usually found in object-oriented data models [Ma91, De93].

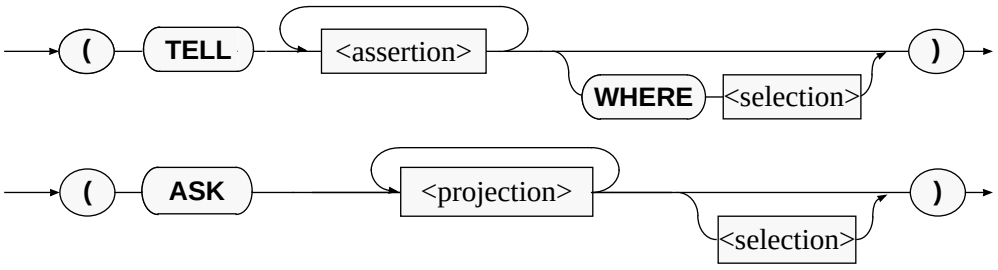


Fig. 1: Syntactic structure of KOALA statements

KOALA (KRISYS Object Abstraction Language) [Ma91], a descriptive, set-oriented language, constitutes the user and application interface of KRISYS. KOALA is computationally complete w.r.t. the KOBRA object model. KOALA features two operations, ASK and TELL (cf., Fig. 1), to perform set-oriented queries or modifications of the object base. Both statements allow to use variables (indicated by leading question marks) for referring to any part of an object, as, for example, entire objects, attributes, methods, aspects, or their values. Moreover, query variables may appear in each part of either ASK or TELL.

An ASK statement consists of two parts, a selection and a projection. The selection clause defines the scope of a query and conditions to be met by entities in order to qualify as query result. Compared to SQL-like query languages, the selection clause combines the tasks of the FROM clause and of the WHERE clause. This is due to the fact that the KOBRA object model treats metadata (i.e., information on classes) and "regular" data (i.e., instances) alike. Hence, specifying a class may serve to access the object itself as well as its extent, i.e., its instances.

Just like the ASK construct, the TELL statement consists of two clauses, a selection part for detailing the objects which are to be manipulated, and an assertion part describing the state to be achieved by the TELL at hand. This *state-orientation* introduces a major difference to relational query languages, as it allows the user to specify the net effects of modifications instead of forcing him/her to explicitly declare which actions must be taken to transform the current state of the object base into the new one. As outlined in [DLM90, De93, DLMT93], this feature is especially valuable if the query language is employed for processing constraints or rules, as it is the case for KOALA. Hence, depending on the current state of the object base, a TELL operation, translated to a relational setting, may correspond to possibly multiple INSERTs, UPDATEs, DELETEs, or no modification operations at all.

```
(1.0)(ASK ((?X)(?Y))                                          } projection clause
(1.1)      (AND (IS-INSTANCE ?X rooms *)
(1.2)           (EQUAL South (SLOTVALUE orientation ?X))
(1.3)           (IS-INSTANCE ?Y furnishings *)                } qualification clause
(1.4)           (> (SLOTVALUE price ?Y) 1000)
(1.5)           (MESSAGE is-suitable-for ?Y ?X)))
```

Fig. 2: Sample ASK statement.

In summary, KOALA offers a versatility which is comparable to relational query languages, but which is hardly to be found in the realm of object-oriented database systems. For the examples to follow, we assume an architectural application with an object base containing generalization hierarchies for *rooms* and *furnishings*. The ASK statement given in Fig. 2 selects all furnishings costing more than US$1,000, which are suitable for rooms located at the south side of the house to be planned. For reference purposes, we numbered the lines of the query. The projection clause (line (1.0)) states that the complete objects retrieved constitute the result of that query. The query refers to the abstraction concept of classification and reads as follows: Firstly, instances of *rooms* (direct as well as indirect ones, indicated by the asterisk behind the class name) are retrieved into query variable ?X (line (1.1)). This set of objects is further restricted by the condition that attribute *orientation* contain value 'South' (line (1.2)). In addition to instances of *rooms*, the query also refers to instances of *furnishings* which must have a price higher than US$1000, represented by the value of attribute *price* (lines (1.3), (1.4)). Finally, a method *is-suitable-for* is invoked to determine the feasibility of furnishings for certain rooms (line (1.5)). This is done by resorting to the MESSAGE predicate provided by KOALA.

2. KRISYS Architecture and Processing Model

We will now present the current KRISYS architecture, as shown in Fig. 3, to give an overview of the tasks of each system component and to exemplify the interaction of the components.

2.1 Overview of the Architecture

The server part, formed by the *PRIMA database kernel* [HMMS87], concentrates on efficient and reliable object-base management. At its interface, it features composite objects represented in terms of the MAD (Molecule-Atom Data) model [Mi88], and that are accessible via the associated query and manipulation language MQL. Thus, application-independent data management is warranted. The client side of KRISYS is partitioned into several components. The Working Memory, the client-based object buffer, is controlled by the Context Manager which is responsible not only for maintaining a declarative description of the Working Memory contents, but also for loading and unloading sets of objects into or from the application buffer. To transfer objects between server and client, the Context Manager interacts with the Mapping System, which transforms objects from the server data-model to KOBRA and vice versa. The Mapping System is also responsible for generating appropriate, application-specific mapping schemes for objects. The Constraint Manager performs all activities related to checking or processing the constraints of the object base. The KOALA Processing System, finally, provides the user (and application system) interface. Its task is to process and execute KOALA queries.

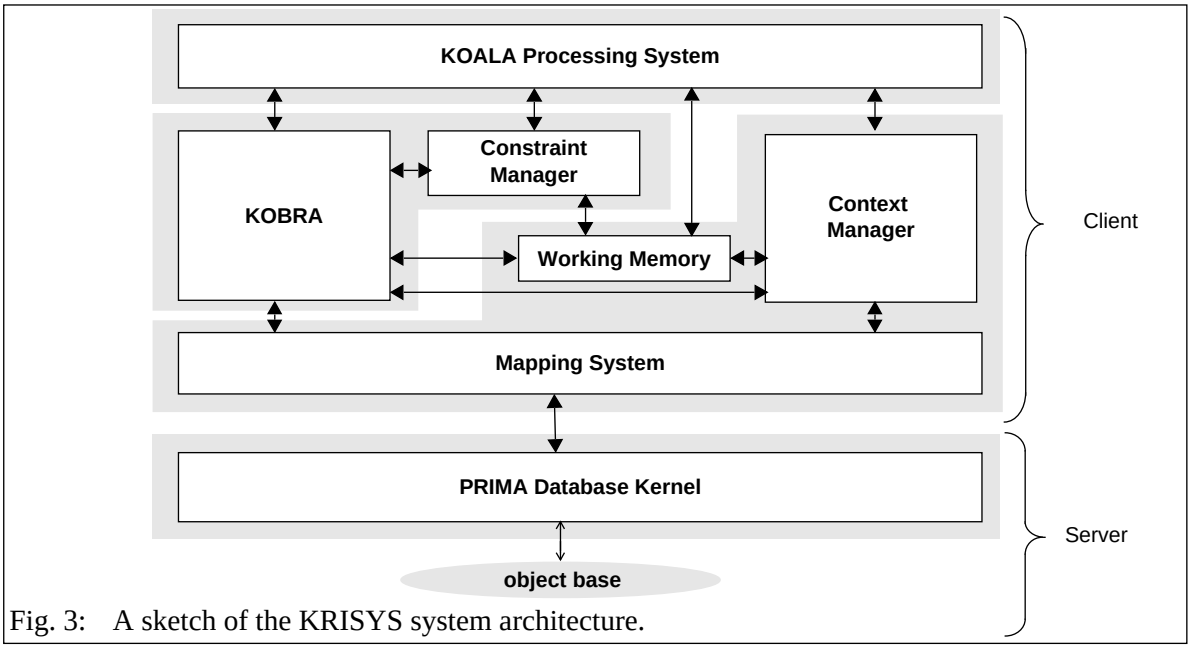


Fig. 3: A sketch of the KRISYS system architecture.

2.2 System Components

The *Mapping System* provides KOBRA objects as uniform representation for the client-based components of KRISYS. Thus, it isolates processing in the client from representational aspects of the server database system. Moreover, the Mapping System allows to generate optimized, application-dependent mapping schemes and to exploit them during application processing. Such a mapping permits, for example, to combine several interrelated classes in a single table or to split one class across several tables, in order to improve the performance of critical DML operations. The activities of the Mapping System can be divided into the following independent subtasks:

- generation of an optimized mapping for a specific application,
- transformation of delegated KOALA subqueries into queries of the PRIMA kernel, and

- adaptation of the mapping in case of changes in the object-base structure (schema evolution).

The *Working Memory* is the KRISYS object buffer, and its general task is to support locality of reference during query processing. In order to accomplish this task, the Working Memory

- provides data structures and operations for representing and effectively manipulating objects in a format directly reflecting the semantics of the object model,
- allows efficient set-oriented processing of objects by the KOALA Processing System through so-called *Access Structures* (AS), combining scans with main-memory index facilities, and
- supports pointer-like navigational access or traversal of objects in abstraction hierarchies to optimize the processing of model-inherent constraints.

Transformations of the object format take place whenever objects are transferred from the server and stored in the Working Memory. Such transformations include swizzling pointers representing inter-object relationships, constructing appropriate Access Structures, etc. While the Working Memory provides basic functions for accessing objects in main memory, buffer management is performed by the Context Manager introduced later in this paper.

The *KOALA Processing System* is responsible for performing statements formulated in the query language KOALA [Th96]. The overall steps of query processing proceed in a similar fashion as those in relational database systems [HFLP89]: first, an algebra graph is generated and subsequently optimized, i.e., rewritten; thereafter, an execution plan is constructed; finally, executable code is assembled, and the query is actually evaluated. We will discuss these steps more elaborately in Chapter III, Sect. 2.

The evaluation of a query must exploit the Working Memory contents as far as possible. Therefore, the KOALA Processing System closely interacts with the Context Manager and with the Constraint Manager to identify those parts of the query that should be performed at the client side and those that are to be delegated to the server. As we will see in our discussion of implementational aspects of query processing (cf., Chapter III), this decision is reflected by different types of plan operators in the execution plan (e.g. 'Buffer-SELECT' and 'DB-SELECT').

During the generation of an execution plan, the KOALA Processing System has to find out which parts of the query may be executed on the buffer contents, because the required objects are already present in the Working Memory. A buffer description based on object identifiers is not sufficient for accomplishing this task [De93, DLMT93]. Instead, a declarative description is required. It is the major task of the *Context Manager* to maintain such declarative buffer descriptions. They can be incrementally constructed from the subqueries that are delegated to the server, since the selection conditions of these queries perfectly describe the results (i.e., the contexts) that are installed in the Working Memory. To provide the required information for the KOALA Processing System, the Context Manager performs special context-inference operations comparing data requests of queries to the contexts of the buffer and producing a declarative description of those object sets to be fetched from the server database system.

The *KOBRA* component realizes the semantics of the KOBRA object model and its associated operational facilities [Ma91]. KOBRA provides operations for accessing single objects based on their object names. This includes functions, e.g., for creating or deleting objects, for reading or changing attribute values, for establishing or removing abstraction relationships, or for executing methods. Objects may be interrelated by one or more of the abstraction concepts or by virtue of application-defined relationships. Thus, networks of objects can be constructed. While KOALA provides transitive-closure operations for surveying such networks in a set-oriented fashion, the functional capabilities of the KOBRA component are restricted to navigational, object-wise operations only. They are accomplished by exploiting object references occurring as values of attributes that establish any of the object relationships mentioned above. KOBRA provides the conceptual and implementational basis for all activities to be carried out

in main memory at the client side. Consequently, KOALA and the activities of the KOALA Processing System are mapped to the operational primitives of the KOBRA component.

The task of maintaining object-base consistency according to the given constraints is fulfilled by the *Constraint Manager* [De93]. Based on events reported by the KOALA Processing System or the KOBRA component (e.g., atomic write/read operations, begin/end of composed activities, etc.), the Constraint Manager initiates actions to ensure consistency, or stores the events for later, deferred activation. Moreover, the creation, deletion, or modification of constraints is reported to this component. Additionally, the Constraint Manager provides information about certain types of constraints to the KOALA Processing System which are necessary for rewriting purposes during query optimization.

2.3 Interaction of System Components During Query Processing

To illustrate the interactions and dependencies between the different system components of KRISYS, we sketch the evaluation of a simple example query. We refer to the query presented in Fig. 2. Details of this discussion can be found in [De91, De93].

After having been submitted to the KOALA Processing System (Fig. 4 ①), the statement is transformed into an algebra graph, on which algebraic optimizations are performed (Fig. 4 ②). These involve query rewrites commonly applied in relational database systems, such as subquery-to-join transformation, selection-push-down, etc.

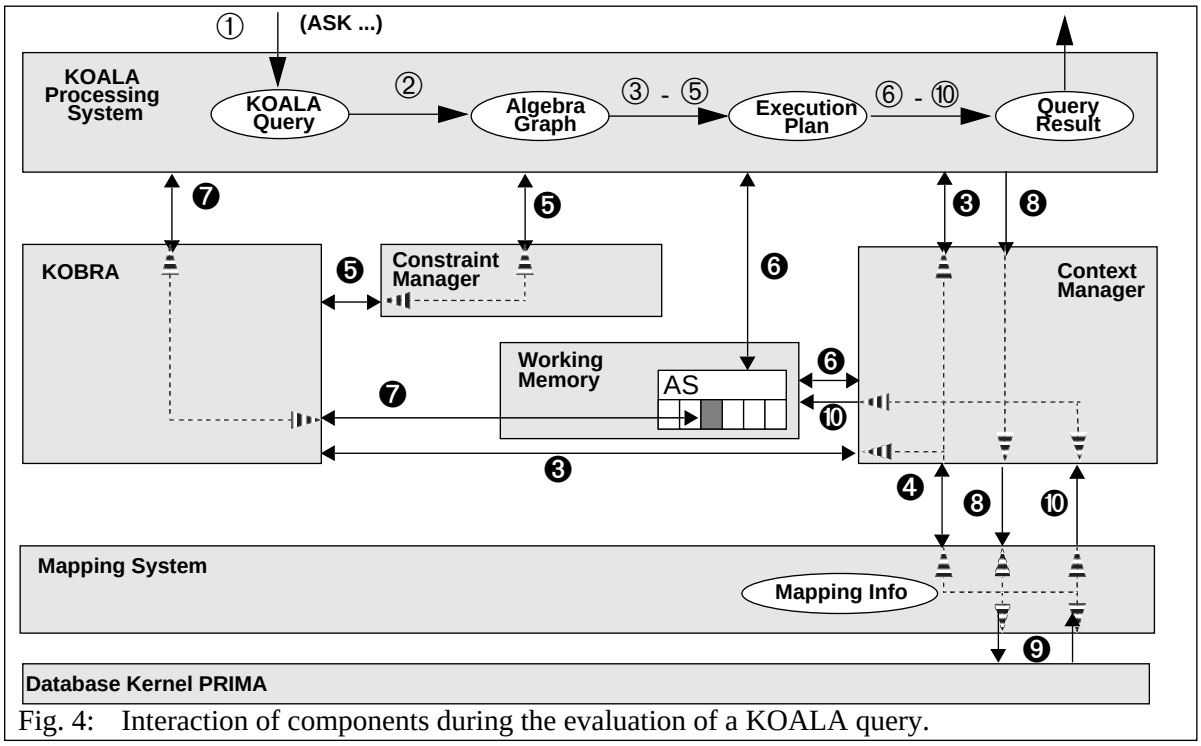


Fig. 4: Interaction of components during the evaluation of a KOALA query.

In the next step, an execution plan is generated. At this stage of processing, the KOALA Processing System interacts with the Context Manager (Fig. 4 ❸) to determine which parts of the query can be executed on the Working Memory contents and which have to be delegated (Fig. 4 ③). Not all constituents of the query are considered for delegation. For example, all operations involving method calls, like the join operation of rooms and furnishings resulting from the activation of method *is-suitable-for*, can only be performed at the client side. To provide the required objects, the Context Manager analyzes the descriptions of the contexts currently installed in the Working Memory. There may, for example, be no context containing rooms, so that an appropriate answer is given to the KOALA Processing System, which will then consider the delegation of the corresponding subquery. In many cases, however, there will

be contexts that somehow overlap with the sets of objects requested by subqueries. For this purpose, the Context Manager supports inference capabilities that allow to determine a declarative description of those objects that are still missing and consequently have to be fetched from the server. For our example, we assume that the Context Manager detects that the selection subquery involving furnishings can completely be supported by an existing context.

Using the information provided by the Context Manager, the KOALA Processing System produces an appropriate execution plan. For this task, this component additionally needs an estimation of execution costs (Fig. 4 ④). Here, the mapping scheme chosen for the current application plays a crucial role. To provide the required cost estimations, the Context Manager therefore enriches its description of execution alternatives with cost information provided by the Mapping System, before passing it to the KOALA Processing System (Fig. 4 ❹). Depending on this cost information, the KOALA Processing System determines which inferences drawn by the Context Manager to exploit in order to guarantee optimum query execution.

Additionally, the KOALA Processing System must interact with the Constraint Manager (Fig. 4 ❺)[3], because the evaluation of predicates in the selections to be delegated might involve the activation of constraints, which cannot be performed by the server. For example, the attribute *price* of *furnishings* may be referred to in a constraint relating it to the additional features of the furnishing. Depending on how the object-base designer has chosen to represent this constraint (e.g., defining the price of *furnishing* as a virtual attribute, whose value is computed on demand), additional rewrite operations may be necessary (Fig. 4 ⑤).

Let us assume that the subqueries chosen for delegation do not require to activate constraints, so that the execution plan generated by the KOALA Processing System is confirmed and can be compiled and executed (Fig. 4 ⑥ - ⑩). The execution of Working Memory plan operators is based on Access Structures containing sets of objects (Fig. 4 ❻). Each operator can be understood as producing a temporal Access Structure to be consumed by its successor. The functionality required to implement the operations performed on each element of the Access Structure during the execution of a plan operator (e.g., accessing the attribute 'price' of the instances of *furnishings*) is provided by the KOBRA component (Fig. 4 ❼). Working Memory operators appearing as leaves of the plan-operator graph rely on contexts residing in the application buffer. To this end, the Context Manager provides initial access to the associated contexts organized in particular Access Structures managed by the Context Manager. In our example, an Access Structure containing the furnishings is provided.

The execution of database plan-operators is performed in several steps. First, the Mapping System is consulted to produce an equivalent server DML operation based on the current mapping scheme (Fig. 4 ❽). This DML operation is sent to the server and executed (Fig. 4 ❾). The result of the query is then returned to the Mapping System, which transforms it into the Working Memory representation (i.e., KOBRA objects). Finally, the resulting objects are inserted into the Working Memory and collected in a new Access Structure (Fig. 4 ❿). This last step is performed by the Context Manager, which registers the result of the delegated subquery as a new context and provides it as an Access Structure to the KOALA Processing System.

Plan execution is continued in the above described manner and completed by returning the result of the query to the user or application.

3. Please note that points ③ - ⑤ are not necessarily executed in the sequential order chosen above for illustrating the interactions.

III Implementational View to Advanced Data Processing in KRISYS

1. Working Memory

As described above, operations on objects sparked by applications are carried out through the KOALA Processing System, the Constraint Manager, the Context Manager, and KOBRA. All client-based activities arising in this fashion are performed in the Working Memory. This application buffer therefore must provide adequate efficient support functions for its associated system components.

1.1 Efficient Navigational Access to Objects

An important processing requirement is the fast localization of objects in the Working Memory based on their identifiers. This is achieved through a hash table relating object identifiers to the current location of the objects in the buffer. Efficient access to object-internal information (i.e., attributes, aspects, etc.) is supported by means of a pointer-based representation. Access usually occurs repeatedly to different attributes of the same object, or to different aspects of the previously accessed attributes, and can therefore be seen as a kind of ‘navigation within the object’. For example, an update operation involves to localize the object in the Working Memory, to modify an attribute within the object, and to additionally access aspect information associated with the attribute, in order to record events and notify the constraints affected by the update.

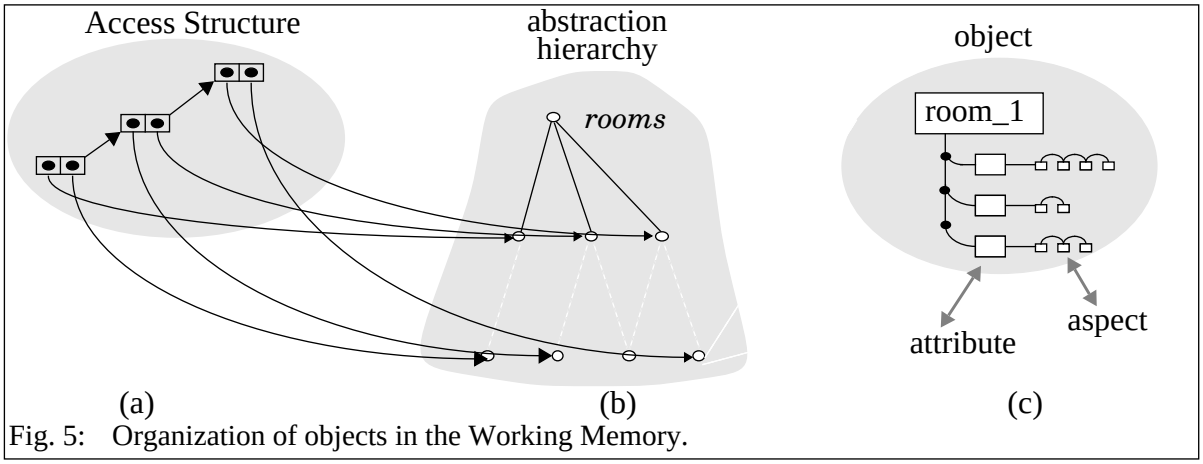


Fig. 5: Organization of objects in the Working Memory.

The different internal representational levels (object, attribute, and aspect) are directly reflected in the Working Memory representation (Fig. 5 (c)), and are linked via main-memory pointers. The pointers allow to efficiently retrieve attribute and aspect information stored in the object. In Fig. 5 (c), we have sketched this representation for object *room_1*.

This representation has the additional advantage to integrate structurally heterogeneous objects in a single, uniform data structure for accessing object information. For example, even objects belonging to the same class may have varying structures because they belong to structurally different subclasses. Moreover, some objects might be instances of multiple classes, or some attributes may have been defined only for individual objects. Due to the above representation scheme, information about objects can be retrieved and modified on a uniform basis without access to meta-information (e.g., class descriptions) in order to interpret the data structures. Moreover, the Working Memory provides functions for creating and deleting data structures for objects, attributes, and aspects, as well as for read/write access.

1.2 Fast Navigation across Abstraction Hierarchies

It is important to speed up the retrieval of objects via abstraction relationships (e.g., all transitive instances of a class) and to provide means for efficiently guaranteeing model-inherent integrity constraints. For example, the creation of a new attribute in a class requires to traverse the class hierarchy to perform inheritance. Consequently, the abstraction relationships between objects are materialized as main-memory pointers. This materialization is depicted in Fig. 5 (b) for the generalization/classification hierarchy of *rooms*.

Besides operations for establishing/deleting abstraction links among objects, the Working Memory offers additional functionality to traverse abstraction hierarchies and perform operations on the traversed objects. These functions, which are mainly used for maintaining model-inherent integrity, can be supplied with parameters that determine the relationships to be followed, specify a search strategy for the traversal, or denote operations to be performed at each node during the traversal. For example, attribute inheritance was easily implemented as a breadth-first traversal following the *subclass-of* and *instance-of* relationships.

1.3 Set-Oriented Processing of Objects in the Working Memory

For set-oriented processing of objects, the Working Memory allows the KOALA Processing System to create, maintain, and exploit collections of objects organized as Access Structures (cf. Fig. 5 (a)). In order to be suitable for the purposes of the KOALA Processing System, an Access Structure must contain items that match the internal format used during query processing, i. e., tuples, which will be described in detail in Sect. 2.3 of the current chapter.

As shown in Fig. 5 (a), these tuples do not contain copies of Working Memory objects, but are associated with the objects via main-memory pointers. In this example, the Access Structure comprises pairs of objects resulting from a join. This ensures that during query processing no redundancies are introduced by the KOALA Processing System. Intermediate results are produced by employing a sophisticated concept for sharing object information even at the attribute and aspect level, using multiple pointers to the same information.

The Working Memory provides the following operations for exploiting Access Structures.

- Creation and deletion of Access Structures.
- Opening and closing cursors for Access Structures, which allow to scan Access Structures in forward or backward direction. Multiple cursors can be defined for the same Access Structure, so that an intermediate query result can be exploited by several ‘threads’ of the query execution simultaneously.
- Functions for reading, inserting, removing, and replacing tuples of Access Structures relative to the cursor position.

In their basic form, Access Structures are organized as lists of tuples. In addition, they can also be implemented as search trees or hash tables, thereby serving as main-memory indices. To this end, additional information must be provided upon creation of Access Structures, describing the key attributes of objects to be indexed. During query processing, such index structures may be installed dynamically and temporarily, e.g., in the scope of a single query or transaction. Moreover, associative access to the contents of Access Structures is supported through additional operations. With these facilities, the KOALA Processing System can fully exploit the contents of the Working Memory during query processing.

In summary, the Working Memory directly and effectively covers the basic requirements related to the processing of object information, thereby providing a suitable foundation for query processing in the client component of KRISYS.

2. Query Processing

Query Processing is performed by the KOALA Processing System [Th96]. To guarantee a semantically clear and streamlined system design, we partitioned its overall tasks into a processing framework and a part responsible for object-model semantics. While the processing framework is based on an algebraic model that allows conventional (relational) algebraic optimizations to be used to a large extent, object-model semantics is founded on the functionality provided by KOBRA. In the following, we will discuss those issues in detail.

2.1 Object-Model Semantics

Except for the notion of object structures, the processing framework of the KOALA Processing System is completely independent of object-model semantics which is introduced via *base predicates*. Base predicates represent an intermediate level between KOALA and KOBRA. Fig. 6 depicts the different representational levels and their processing characteristics. While KOALA expressions are declarative, state-oriented, and set-oriented, base predicates operate object-wise, however still being declarative and state-oriented. Since base predicates resemble assertions on single objects, they can be easily mapped to the procedural level of KOBRA.

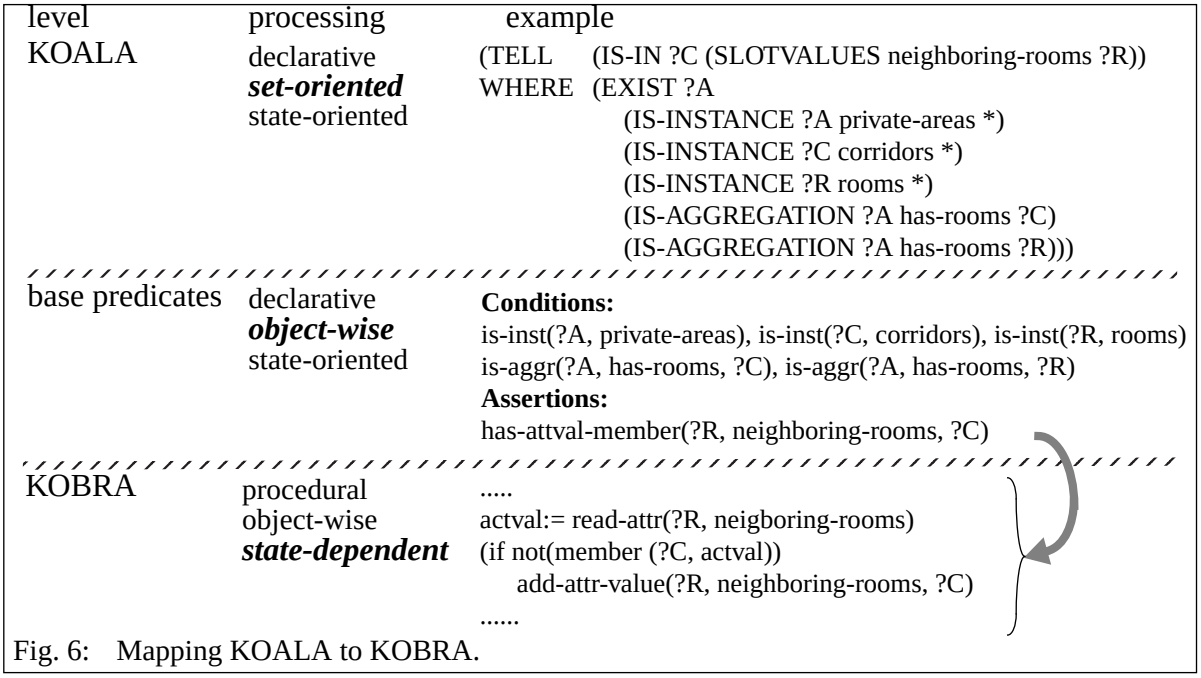


Fig. 6: Mapping KOALA to KOBRA.

To the right side of Fig. 6, we sketched how an example KOALA statement is firstly translated to the level of base predicates and then to the KOBRA level. We use a TELL statement asserting that all corridors (being instances of that class and bound to query variable ?C) are adjacent to any room ?R lying in the same private area ?A. In the selection part (WHERE clause), the existence of private-areas, corridors, and rooms, as well as of aggregation relationship *has-rooms* is assumed. Let us have a look at the way the assertion is translated to base predicates. In the state to be achieved, each qualifying corridor ?C must be a value of attribute *neighboring-rooms* of any adjacent room of that private area. Consequently, the assertion is translated into a piece of code at the KOBRA level that reads the value of attribute *neighboring-rooms* and adds the current value of ?C to the attribute value if it is not yet included.

2.2 Processing Framework

As mentioned before, the overall steps of query processing proceed in a similar fashion as those in relational database systems [HFLP89]. We will discuss them more concisely in the following.

Due to the client/server environment in which query processing is performed, determining the evaluation site of each operator is a crucial issue (step ❷ in Fig. 7). By delegating operations to the server, the amount of data to be transferred to the Working Memory can be reduced significantly. This also results in less objects to be installed in the Working Memory allowing a better exploitation of its storage capacities. Deciding on the evaluation site of each operator is based upon two criteria. Firstly, those algebra operators must be assigned to the client that are either too complex to be evaluated by the server component or that cannot be transformed into appropriate queries due to the current mapping to the server database system.[4] Secondly, for performance reasons, the KOALA Processing System must exploit the contents of the Working Memory (including indices, sort orders, etc.).

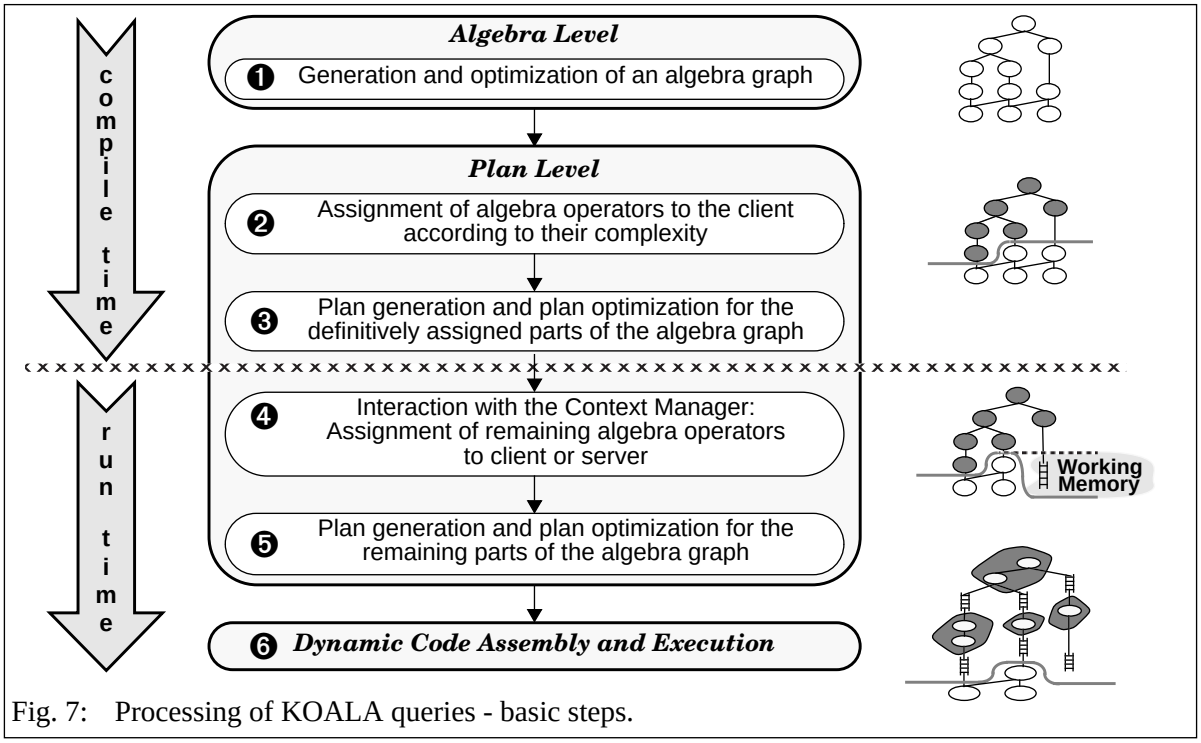


Fig. 7: Processing of KOALA queries - basic steps.

The first criterion can be tested at compile time so that a preliminary borderline between client-based and server-based operations can be drawn (cf., Fig. 7, right side). Depending on the contents of the Working Memory at run time, the operators below the borderline may be assigned to client or server. Hence, plan-level manipulations can be definitively completed only at run time, yet, to save run-time effort, preliminary plan optimizations may be performed for those operators definitively assigned to the client (step ❸ in Fig. 7).

At run time, the KOALA Processing System interacts with the Context Manager to compare the actual contents of the Working Memory to the information referred to by the query at hand. If the input to an operator already resides in the Working Memory (as an Access Structure), the producing subgraph[5] is pruned and replaced by a pointer to the appropriate Access Structure. While this applies to operators above as well as below the preliminary borderline, for the latter it also implies that these operators are assigned to the client, i.e., the borderline is moved downward, and less operators must be delegated to the server (sketched in Fig. 7, right side). For those subgraphs not yet assigned to client or server, two further situations may arise. If the Working Memory does not contain any input for a subgraph, the whole subgraph must be evaluated at the server[6], and the border between client and server processing remains where it has been put at compile time.

4. For simplicity, we shall not consider this aspect in this paper.
5. Consisting of one or more plan operators.

The second situation occurs if only part of the required input is residing in the Working Memory, and the rest must be fetched from the server. In this case, basically two processing strategies are possible: either to completely delegate the query to the server, requiring to previously write back the potentially updated objects installed at the client side, or to complement the Working Memory contents such that the query can be performed at the client side. To solve this optimization problem, the KOALA Processing System interacts with the Context Manager (step ❹ in Fig. 7). This interaction and its outcome for our example query will be detailed in Sect. 2.3. Fig. 9 (b) shows the resulting plan-operator graph[7] assuming that no instance of *area* is installed in the Working Memory. Hence, the corresponding selection must be executed in the server and is transformed into a server plan-operator (DB-SELECT). Since the subsequent UNNEST operator refers to object structures of the object model, it must be carried out at the client side. Consequently, all its successors must be executed there as well. Since making assertions over the object base may involve the full functionality of the object model, algebra operator ASSERT is always transformed into a client-based plan-operator (of the same name).

The resulting plan-operator graph can be further optimized (step ❺ in Fig. 7).

2.3 Algebra Level

Algebra operators work on data streams consisting of sets of n-tuples which they accept as input and also produce as output. A data stream can be seen as a table made up of n columns bound to query variables. A table is represented as an Access Structure in the Working Memory, and each n-tuple (table row) represents an associated entry comprising n elements. Each element, in turn, features object level, attribute level, and aspect level. Depending on the operations to be performed, this nested structure of column elements may be resolved. This is achieved by unnesting the elements on the attribute level and/or the aspect level. Fig. 8 depicts an example table consisting of 2-tuples and demonstrates the effects of unnesting/nesting the first column on the attribute level[8].

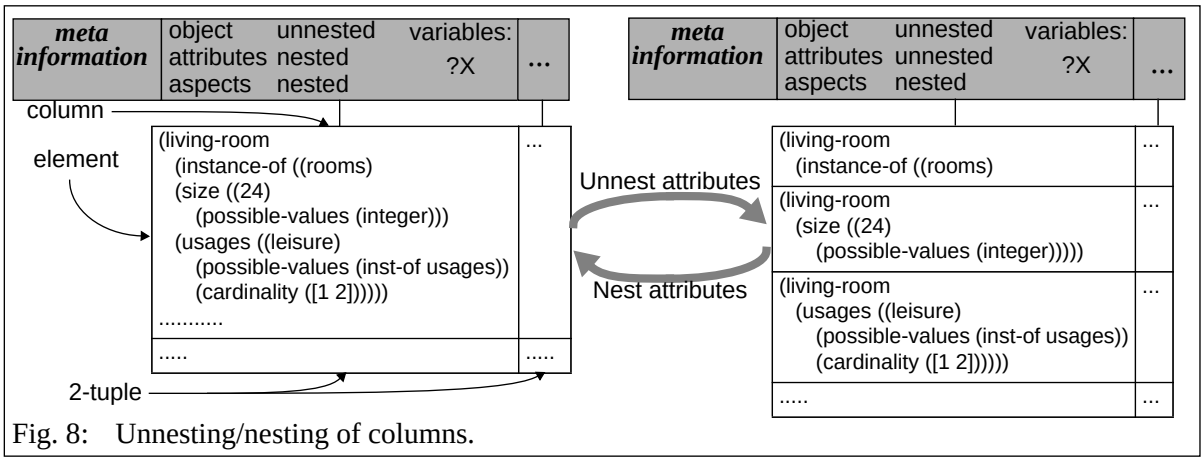


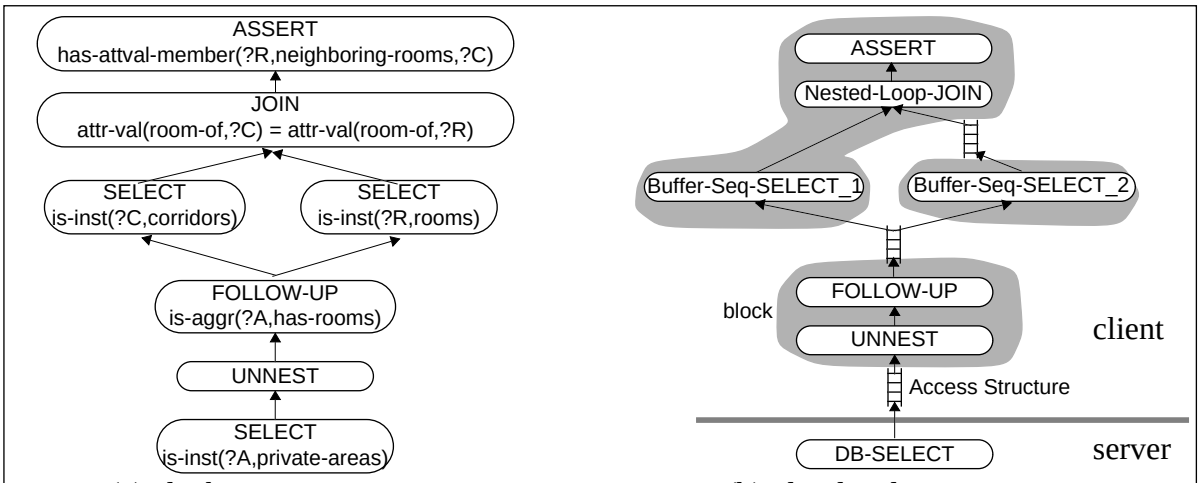
Fig. 8: Unnesting/nesting of columns.

The KOALA algebra consists of three kinds of operators. Firstly, there are operators that are responsible for handling columns or object structures (e.g., COL-COPY, COL-PROJECT, COL-UNION, NEST, UNNEST). The operators of the second kind provide functionality comparable to conventional relational algebras, e.g., EXIST, FORALL, JOIN, PRODUCT, or SELECT. The third kind is responsible for modifications of the object base. As described above, the KOALA algebra employs state-oriented base predicates for realizing object-model

6. Note that it has already been checked at compile time that all operators below the borderline can be evaluated at the server.
7. For simplicity, we did not repeat the base predicates for the plan operators.
8. Nesting and unnesting of attributes is used, for example, during projections.

semantics. Consequently, the algebra level need not consider the actual state of the object base, and requires only a single operator, ASSERT, to carry out modifications. In relational algebras, however, where state-orientation is not known, several operators are required to carry out changes in the database (e.g., UPDATE, INSERT, DELETE).

To illustrate how a query is translated into an algebraic representation, we refer to the sample TELL statement and the way it is decomposed into base predicates shown in Fig. 6. First, an algebra graph is constructed (cf. step ❶ from Fig. 7). It is shown in Fig. 9 (a). Firstly, instances of *area*s are selected. Since relevant rooms and corridors must be components of some private area, the corresponding object identifiers can be retrieved from attribute *has-rooms* of each selected area. To access this attribute, each area object must be unnested on the attribute level. Thereafter, all objects referenced by attribute *has-rooms* of a given area can be retrieved. Since such an evaluation of object references is quite a frequent operation, the KOALA algebra provides a special operator, FOLLOW-UP. After the FOLLOW-UP operation has been performed, rooms and corridors are selected separately. Those belonging to the same private area are joined and provided as input to the assertion part of the TELL statement.



(a) algebraic representation (b) plan-level representation

Fig. 9: Example TELL statement.

2.4 Plan Level

Our plan-operator approach involves the concepts shown in Fig. 10. We briefly recapitulate the prominent features of the plan level; for details we refer to [TD93, TGHM95, Th96].

Plan-operator templates realize a *simple processing paradigm for plan operators*, as well as *extensibility at the plan-operator level*. Object-model semantics is introduced into plan-operator processing via base predicates supplied as parameters to the plan operators. By combining plan-operator functionality with object-model semantics in such a fashion, *extensibility of the query language* is possible without affecting existing plan operators. Plan-operator subgraphs are combined to units of execution, called *blocks* (cf. Fig. 9 (b)). Data streams between blocks are materialized in the Working Memory and mapped to Access Structures, thus ensuring *efficient data flow between blocks*. To minimize the amount of intermediate Access Structures, blocks are constructed such that intra-block processing works in a pipelining mode, i.e., tuple-wise, without the need for intermediate result materialization [TGHM95, Th96]. The concept of *LAS* (Logical Access Structures) provides an adequate data structure for this kind of internal data flow. Structurally, LAS correspond to their “physical” counterparts, yet they abstract from the internal organization of Access Structures that is visible at the interface of Access Structures, i.e., whether the Access Structure is a simple list, a tree, or a hash table. Instead, LAS possess a uniform interface tailored for sequential processing by supporting an *open-next-close* protocol consisting of a set of navigational operations.

Finally, the way in which blocks, Access Structures, and LAS are combined warrants *efficient dynamic query optimization* and the construction of *flexible units of execution* even at run time. These characteristics were achieved by a modular design and realization of the plan level.

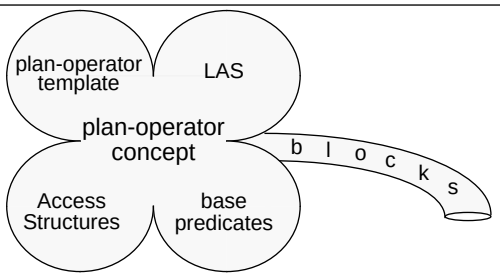


Fig. 10: Constituents of our plan-operator concept.

2.5 Dynamic Code Assembly and Execution

The tasks of dynamic code assembly and execution complete overall query processing (step ❻ in Fig. 7). The plan graph of a query is divided into *blocks* (sketched in Fig. 7, right side) which are the units of execution in our approach [TGHM95, Th96]. Just like plan operators, blocks accept one or more input streams and produce a single output stream. Blocks are constructed based on the processing characteristics of plan operators to minimize the amount of materialized intermediate results during query processing. For this purpose, a set of rules has been defined which exploits both the structural properties of a given plan-operator graph and the processing characteristics of the operators contained. Fig. 9 (b) shows the blocks derived for our example query. The UNNEST and FOLLOW-UP operators work in a pipelined fashion, and are therefore combined into a single block. The same holds for the Buffer-Seq-SELECT_1 (alternatively Buffer-Seq-SELECT_2), NESTED-LOOP-JOIN and ASSERT operators.[9]

Evaluating a query means executing the corresponding blocks. The most straightforward way is to perform blocks in a sequential order defined by the inter-relationships of the block-structured graph. Additionally, our query-processing approach also permits parallel execution of blocks [TMMD93, Th96]; however, this topic is not discussed in this paper.

Opposed to conventional query-processing systems requiring *strict compilation*, we assemble executable code by putting together precompiled functions, yet we may also compile a query, e.g., for complex queries or large amounts of data to be processed. We call this approach *dynamic code assembly* [TGHM95, Th96], allowing to assemble executable code from fully compiled code fragments employing data structures containing function pointers.

3. Context Management

It is the task of the Context Manager to provide the KOALA Processing System with information about Working Memory contents during query optimization and plan generation. Due to the declarative query interface to the server component, the Context Manager can maintain a description of the buffer contents in a declarative form as well. Consequently, the Context Manager perceives the Working Memory as a collection of *contexts*. A context represents a set of objects being the complete extension of a logical condition, the *context description*.

Contexts directly correspond to the results of (sub-)queries that have been delegated to the server and whose results have been brought into the Working Memory. For each set of query results received from the server, the Context Manager keeps the query condition as a context

9. Note that, for this block being able to operate as a pipeline, the complete results of Buffer-Seq-SELECT_2 must be computed previously. Only in this case, the NESTED-LOOP-JOIN can directly process any new result being piped from Buffer-Seq-SELECT_1.

description and maintains an Access Structure that contains the set of result objects.[10] The language for context descriptions is therefore equivalent to the subset of KOALA that can appear as a condition of DB-SELECT plan operators. In the following, we will illustrate the main activities performed by the Context Manager in coordination with the KOALA Processing System using the example query already introduced above (cf., Fig. 2).

3.1 Context Description

Let us assume, that all instances of *areas* having more than three rooms have been retrieved from the server by a previous query. The Context Manager has registered the following context description.

```
((?X)                                          ← projection (P)
    (IS-INSTANCE ?X areas *)                   ← variable definition (V)
    (> (SLOTVALUE no-of-rooms ?X) 3))          ← selection (S)
```

The description consists of three parts. The *variable definition part* (V) characterizes the domain of the context in terms of predicates referring to the abstraction concepts. The *selection part* (S) states further conditions applying to the context. Finally, the *projection part* (P) lists those attributes that have been brought into the Working Memory.

3.2 Context Comparison

When consulting the Context Manager, the KOALA Processing System submits a description of a 'wanted' context W, resembling the subquery currently under consideration. The Context Manager compares W with a 'given' context G, i.e., with a context available in the Working Memory. For our example query, the KOALA Processing System will ask the Context Manager about contexts available for supporting the selection on *private-areas*. The result of the involved context comparison is depicted in Fig. 11.

First of all, the projection parts of the contexts are compared. Since both projection parts preserve complete object structures, they turn out to be equivalent. Next, the variable definitions are compared. To determine the result, the Context Manager will at this point have to inspect the abstraction relationships defined in the object base. Since *areas* is known to be a superclass of *private-areas*, the result of the comparison is the set inclusion $V(G) \supseteq V(W)$. Finally, the selection parts are compared. Since no additional selection is defined for W (i.e., all *private-areas* are contained in the context), the comparison results in the set containment $S(G) \subseteq S(W)$.[11] The comparison of predicates in both the selection and the variable-definition parts relies on the interpretation of set relationships as logical connectives, where set containment is equivalent to logical implication.

To obtain the overall relationship between G and W, the individual comparison results for P, V, and S are combined. In our example, the relationship 'G overlaps with W' (denoted by 'O') is achieved, because we have obtained two 'inverse' set inclusions in V and S. The 'overlap' result implies that we can exploit the context existing in the Working Memory for answering the query. However, we still have to access the server for those objects not covered by the context. Therefore, the Context Manager also produces descriptions how to filter the existing context for the required result set (i.e., how to obtain $G \cap W$ from G), and how to retrieve the remaining objects from the server (i.e., how to retrieve $W \setminus G$). These results are passed to the KOALA Processing System for modifying the query plan accordingly. Moreover, a pointer to the Access Structure containing G is passed on to make it accessible for the KOALA Processing System.

10. The Access Structure can later be handed to the KOALA Processing System for accessing the context.
11. If several predicates are involved in a selection (or a variable-definition) part, each predicate of G must be compared with each one of W. For complex selection conditions (involving disjunctions, etc.), a disjunctive variant of the algorithm is supplied in addition to the above described (conjunctive) version.

Let us assume that in our example, the KOALA Processing System chooses not to consider the private areas already in the Working Memory but to fetch all instances of *private-area* from the server. Before executing the query, however, the Context Manager must write back to the server all modified objects relevant for this server operation. Analogously, the Context Manager is asked about contexts for rooms and corridors, the other classes involved in the example query. For reasons of simplicity, we assume that these subqueries can be fully supported by contexts at the client component. The resulting query execution plan is depicted in Fig. 9 (b).

G(iven)	Rel.	W(anted)
P: (?X)	≡	P: (?X)
V: (IS-INSTANCE ?X areas *)	⊇	V: (IS-INSTANCE ?X private-areas *)
S: (> (SLOTVALUE no-of-rooms ?X) 3)	⊆	S: 'true'
context G	'O'	context W

$G \cap W$:

```
SELECT (IS-INSTANCE ?X private-areas *) FROM G
```

$W \setminus G$:

```
LOAD (?X)
  (AND  (IS-INSTANCE ?X private-areas *)
        (NOT (> (SLOTVALUE no-of-rooms ?X) 3))
```

Fig. 11: Context comparison (example).

The above algorithm, which is outlined in detail in [De93], exhibits polynomial time complexity w.r.t. the number of predicates involved in the comparison. It is important to note that, although we retrieve only objects in W \ G from the server, we might well retrieve objects that are already in the Working Memory. For example, other contexts might be present that overlap with W, but are not exploited because the 'amount of overlap' is not promising enough. The Working Memory is capable of handling this situation simply by ignoring newly fetched versions of already installed objects (i.e., no additional copies are introduced into the Working Memory).

3.3 Additional Tasks of the Context Manager

Although maintaining and comparing context descriptions are the central task of the Context Manager, additional activities are performed by this component to achieve consistent buffer management based on the notion of contexts.

For instance, the Context Manager is involved in the process of *update propagation* to the server. Before a query is delegated there, updates that have occurred on Working Memory objects must be propagated to the server. Otherwise, inconsistencies between server database and Working Memory may arise. Using the Context Manager, we can realize a partial update delegation approach, i.e., only those updates (or a relatively small superset) that are needed to guarantee a correct query result are propagated.

Additional activities of the Context Manager relate to keeping context extensions 'up-to-date' after modifications, and to discarding contexts from the Working Memory. A detailed discussion of these tasks can be found in [De93].

4. Consistency Control

Consistency control in a DBMS is needed for various aspects: semantic (logical) integrity control or constraint management, multi-user synchronization, and physical database integrity. In advanced DBMS, these tasks become more challenging, because the rich modeling concepts provide more complex operations on the more complex data structures. Their consistency has to be preserved in case of modification operations and multi-user access, thereby observing the

well-known transaction concept whose key properties are atomicity, consistency, isolated execution, and durability - the so-called ACID properties [HR83]. Hence ACID transactions can be seen as dynamic control structures isolating the operations of multiple users and making failures transparent to them. These properties, in turn, are guaranteed by the components responsible for consistency control: constraint management, concurrency control, as well as logging and recovery. Each of our solutions to implement these components is - in its very nature - complex and multi-facetted. Due to space restrictions, we can therefore only outline our conceptual approaches and their implementations.

4.1 Constraint management

KRISYS supports a layered approach for representing constraint characteristics at different abstraction levels. The central part of a constraint is its condition, i.e., a logical formula that has to be valid in a consistent state of the object base. At the operational level, the constraint is described in terms of adjustments telling the system how to correct inconsistencies. For a single constraint, alternative adjustments can be specified, which are selected and executed according to various criteria. At the realization level, the 'implementation' of a constraint is described in terms of event patterns, whose occurrence will lead to the execution of particular checking and adjustment operations. Among other things, the constraint designer may choose either a data-driven realization (i.e., where violating actions trigger corrections), or a demand-driven semantics (i.e., 'dependent' information is recomputed each time it is needed).

In KRISYS, constraints are represented as objects. In other words, an object-base designer defines constraints by creating instances of the classes *constraint* and *adjustment*. Characteristics of constraints, such as the constraint condition and the adjustments state, are represented as attribute values of these objects. In addition, abstraction concepts may be used to further organize the constraints defined for an object base. This is especially important for applications such as design environments where a large number of constraints are defined, probably in a dynamic way and for individual objects.

For an elaborate description of the modeling and processing concepts devised for integrity constraints we refer to [De93].

4.2 Synchronization based on Abstraction Relationships

Concurrency control and failure transparency are the most important properties to be provided for system operation in multi-user mode. Both functions give far-reaching guarantees, and, in order to avoid deterioration of system performance, they need sophisticated and adjusted solutions. First of all, they must support the existing object-base structures and operations in an efficient and effective way to preserve the performance gained in single-user mode as far as possible. Selection or modification of data sets using the spectrum of abstraction concepts or access to such data sets qualified by predicates - comparable to the object-base modeling operations - are the most natural form of advanced data processing. Hence, we tried to capture the properties and processing principles of these structures when we designed the supporting functions of our transaction concept.

The most important techniques in real-world database applications used to control concurrency are based on locking. Especially, the Granular Lock Protocol (GLP, [GLPT76]) and its variants, widely applied in commercial database systems, embody an effective way to control locking overhead and transaction parallelism by providing multiple lock granularities. This hierarchical lock protocol is tailored to database structures forming a single directed acyclic graph (DAG). Its operational power and effectivity stems from the use of intention locks and implicit locks in subhierarchies where all objects in a subhierarchy are covered by a single lock at the root.

Unfortunately, such a protocol is unable to solve all data access conflicts occurring in a KRISYS object base. Note, we have set-oriented operations exploiting the semantics of abstraction relationships, and we have three of them (classification/generalization, association, and aggregation). Each of these abstraction relationships is directed from the root object GLOBAL[12] to all objects participating in them. Furthermore, each object may be a member of multiple abstraction relationships of the same or different type. Hence, concurrency control is more complicated, since multiple inheritance may be used for modeling classification/generalization relationships. In such cases, multiple paths may lead from a superclass to one of its subclasses/instances. If a subhierarchy starting from such a superclass (e.g., $class_k$ in Fig. 12) must be locked, how can we know that all these objects are safely isolated? Of course, limited solutions were proposed, e.g., for ORION [GK88]. The application of the ORION lock protocol restricts multiple inheritance to classes and requires single-class membership for instances. Processing conflicts are solved by explicit locks on all classes in the subhierarchy to be accessed. Obviously, this approach works only well when the number of related classes is small.

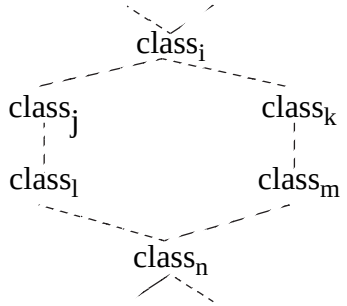


Fig. 12: Multiple inheritance for classification/generalization.

In KRISYS, however, multi-class membership may occur for instances as well. Since explicit locks for all classes and, in particular, for all instances in a subhierarchy may be extremely expensive, the procedure chosen in ORION is not advisable for KRISYS classification/generalization hierarchies. On the other hand, locking only a single path in a DAG may cause the so-called “bastard problem”, i.e., a particular object (instance or class) can be reached via multiple parents in the classification/generalization DAG. In addition, objects can participate in aggregation and association relationships, which makes the synchronization task even more complex. For this general situation where objects can be associated to multiple classes and can participate in multiple abstraction relationships, we developed a locking protocol called LARS. It is based on the application of Locks using Abstraction Relationship Semantics in the following way:

- The object-base structures are logically partitioned into three DAGs for classification/generalization, association, and aggregation.
- A hierarchical lock protocol using tailored locks is provided for each of the logical partitions.
- When locking an object, the paths to be accessed and isolated must be determined dynamically. To do so, lazy locking (that is, not before the object is actually accessed) is used. Explicit locks are only acquired for objects having multiple parents, whereas single-parent objects are implicitly locked.

By using intention locks as well as different lock modes for the different access hierarchies, the LARS protocol keeps the flexibility of the GLP. The details of the LARS protocol can be found in [RH97].

12. This object constitutes the root of all abstraction hierarchies existing in an object base.

4.3 A WAL-based and Object-oriented Recovery Strategy

To provide physical integrity of the database and failure transparency for the user in the framework of a transaction concept, the advanced database system automatically has to recover from all transaction, system, and media failures. Therefore, logging must be applied for all object-base modifications during normal system operation, in order to perform the required recovery actions in case of a failure. Since logging and recovery are typically oriented towards the physical representation of objects, little object-model related semantics can be exploited by these algorithms. As a consequence, all concepts and approaches known from database systems [HR83] can be applied. Therefore, we only sketch the main properties of our solution.

We have designed a recovery algorithm called WALORS [RB96] which is able to run in a STEAL/NOSTEAL and FORCE/NOFORCE system environment. As its name suggests, our algorithm is WAL-based and uses an Object-oriented Recovery Strategy. WAL (Write Ahead Log) is applied when necessary; hence, it allows update-in-place propagation of modified objects, and it supports arbitrary buffer replacement algorithms. WALORS mainly deals with objects instead of pages, that is, logging is performed at the object level in form of physical entries guaranteeing minimal log space and log I/O. In turn, entry logging allows fine-granularity locking at the object level or even at the attribute level.

To check and correlate the state of objects with their corresponding log records, WALORS stores a Log Sequence Number (LSN) in every object of the object base. In contrast to other recovery strategies using page LSNs, WALORS is more flexible during recovery, since it can be very precise when analyzing the log records to be applied to the (indefinite) state of the objects and deciding whether or not to undo or redo object updates. Accordingly, it does not need to perform “repeating history” and is prepared to run selective undo as well as selective redo passes in case of crash recovery. In addition to the standard requirements of failure recovery, WALORS supports partial rollbacks of transactions by means of a checkpoint concept.

The properties of WALORS, in particular fine-granularity logging together with the full set of recovery functions, as well as access synchronization by object-granularity locking, provide a flexible basis for the KRISYS multi-user environment.

4.4 Enhanced Transaction Model

ACID transactions encapsulate the DB-related work and guarantee isolated execution of the data accessed by them. Since advanced applications may incorporate long-running activities, atomicity of all operations inside the transaction is neither useful nor desirable, since each failure would automatically undo all work accomplished in the current transaction. A first refinement deviating from this ‘all-or-nothing’ kind of transaction processing was the provision of checkpoints to prevent loss of work in case of failures. In order to gain more control inside a transaction, we have designed a nested transaction model adjusted to the particular needs of the processing in KRISYS [HR93, RH95]. Such enhanced transactions consist of hierarchically nested subtransactions which are the unit of atomicity and isolated execution.

We are currently implementing the nested transaction model in the KRISYS client/server architecture [Re97]. Based on the flexibility of nested transactions, we enable fine-grained recovery and intra-transaction parallelism to enhance system performance. For example, method invocations may be assigned to subtransactions thereby encapsulating their execution and guaranteeing isolated recovery. Since such method invocations may be recursively triggered, e. g. for constraint adjustments, flexible and fine-grained failure control can be achieved. Furthermore, nested transactions provide means to organize parallel threads on shared data. When supported by suitable adjustments of the locking and deadlock detection protocols, they enable parallel query processing within a transaction in isolation and, as a consequence, automatic fine-grained

recovery in case of deadlocks. In this context, we have already integrated selected protocols into the KRISYS testbed and conducted first performance measurements [RHGL97].

IV Conclusions and Outlook

In the previous chapters, we reported on how the KRISYS system evolved over almost a decade. The central objective of KRISYS has always been to support advanced application processing in an adequate fashion. While during the first development phase of KRISYS provision for expressive semantic concepts (such as an expressive data model, a set-oriented query language, semantic integrity constraints, deductive as well as active capabilities) was at the center of interest, the second stage is characterized by the endeavor to improve processing, architectural aspects, as well as to establish multi-user mode governed by an adequate transaction model, since the practical applicability of database systems is not only determined by their functionality, but also by their performance.

Among the most important issues addressed by the KRISYS framework to advanced data processing are the object model together with the various abstraction concepts and the query language. As already indicated, we have validated all concepts of the object model in various practical applications. Our realization was tailored to a workstation/server environment; its impact on overall data processing affected the following major design and implementation issues:

- Descriptive Buffer Management
  The Context Manager guarantees that the Working Memory contents (at the client side) is exploited for query processing, thus reducing data transfer between workstation and server to a minimum.
- Main-memory-based Query Processing
  The processing framework of KRISYS founds on the KOBRA knowledge model and benefits from well-known query-processing techniques, especially from the areas of relational, main-memory, and object-oriented database systems [HFLP89, IEEE92, Ca91, MPTW94]. That is, requests in the query language KOALA are evaluated following an algebraic approach that was designed to be sufficiently flexible to adapt to language extensions. The plan-operator concept has been adjusted to main-memory query processing; it allows for run-time optimizations and shows extensibility as well.
- Consistency Control
  Semantic integrity control (constraint management), multi-user synchronization, as well as logging and recovery have been designed according to the data model's complex data structures and powerful operations, and finally integrated into the system.

The availability of main-memory query processing opens up a range of further research activities we are currently working on or which will be part of our future work:

- Thorough evaluation of the Context Manager to experimentally investigate the interplay between the data referenced by queries and the costs and benefits of context maintenance.
- Evaluation of the Constraint Manager starting from the basic functionality linking query processing and constraint management, i. e., event management, constraint scheduling, and constraint enforcement.
- Consideration of non-algebraic optimization, i. e., establishing a cost model for query processing taking into account features of our advanced data model (method calls, transitive closure operations like inheritance, etc.), context management, and specialized client/server mapping.

- Completion of a nested transaction facility to support fine-grained recovery and intra-transaction parallelism at both query-execution level and method-invocation level in order to enhance system performance.

The applicability of our approach as well as of the mechanisms necessary for implementing it are not restricted to KRISYS but are generally valid for (advanced) DBMS requiring client-based query processing. Therefore, we see our processing framework and its implementation as a valuable contribution to current research in advanced DBMS.

V References

BJNS94 Buchheit, M., Jeusfeld, M., Nutt, W., Staudt. M.: Subsumption Between Queries to Object-Oriented Databases, Advances in Database Technology - EDBT '94, Jarke, M., Bubenko, J. (eds.), LNCS 779, Springer, 1994, 15-22.

Ca91 Cattell, R. (ed.): Next Generation Database Systems, Special issue of Comm. ACM 34:10, 1991.

CD87 Carey, M., DeWitt, D.J.: An Overview of the EXODUS Project, Data Engineering Bulletin 10:2, 1987, 47-54.

CMCD94 Chen, J., Mattos, N., Chamberlin, D., DeMichiel, L.: Extending Relational Database Technology for New Applications, IBM Systems Journal 33:2, 1994, 264-279.

CR94 Chen, C.M., Roussopoulos, N.: The Implementation and Performance Evaluation of the ADMS Query Optimizer: Integrating Query Result Caching and Matching, Advances in Database Technology - EDBT '94, Jarke, M., Bubenko, J. (eds.), LNCS 779, Springer, 1994, 323-336.

De91 Deßloch, S.: Handling Integrity in a KBMS Architecture for Workstation/Server Environments, in: Proc. GI-Fachtagung "Datenbanksysteme in Büro, Technik und Wissenschaft", H.-J. Appelrath (ed.), IFB 270, Springer, 1991, 89-108.

De93 Deßloch, S.: Semantic Integrity in Advanced Database Management Systems, Doctoral Thesis, Computer Science Dept., University of Kaiserslautern, 1993.

DFMV90 DeWitt, D., Futtersack, P., Maier, D., Velez, F.: A Study of Three Alternative Workstation-Server-Architectures for Object-Oriented Database Systems, D. McLeod, R. Sacks-Davis, H.-J. Schek (eds.), Proc. 16th VLDB Conf., Brisbane, Australia. Morgan Kaufmann Publishers, Palo Alto, CA, 1990, 107-121.

DHMM89 Deßloch, S., Härder, T., Mattos, N., Mitschang, B.: KRISYS: KBMS Support for Better CAD Systems, Proc. Int. Conf. on Data and Knowledge Systems for Manufacturing and Engineering, Gaithersburg, MD., IEEE Computer Society Press, Washington, D.C., 1989, 172-182.

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.

DLMT93 Deßloch, S., Leick, F.J., Mattos, N., Thomas, J.: The KRISYS Project - A Summary of What We have Learned so far, Proc. GI-Fachtagung "Datenbanksysteme in Büro, Technik und Wissenschaft", Stucky, W., Oberweis, A. (eds.), Springer (Informatik Aktuell), 1993, 124-143.

FMV94 Freytag, J.C., Maier, D., Vossen, G. (eds.): Query Processing in Object-Oriented, Complex-Object, and Nested Relation Databases, Morgan Kaufmann, 1994.

GK88 Garza, J.F., Kim, W.: Transaction Management in an Object-oriented Database System, H. Boral, P.-A. Larson (eds), Proc. ACM SIGMOD, Chicago, Ill. SIGMOD Record 17(3), 1988, 37-45.

GLPT76 Gray, J.N., Lorie, R.A., Putzolu, G.R., Traiger, I.: Granularity of Locks and Degrees of Consistency in a Shared Data Base, in: G.M. Nijssen (ed), Proc. IFIP Working Conf. on Modelling in DBMSs, North-Holland, 1976, 365-394.

Gr93 Graefe, G.: Query Evaluation Techniques for Large Databases. ACM Comp. Surveys 25: 2, 1993, 73-170.

Gr94 Graefe, G.: Volcano, an Extensible and Parallel Query Evaluation System, IEEE Trans. Knowledge and Data Engineering 6:1, 1994, 120-135.

HFLP89 Haas, L. Freytag, J., Lohman, G., Pirahesh, H.: Extensible Query Processing in Starburst, J. Clifford, B. G. Lindsay, D. Maier (eds), Proc. ACM SIGMOD, Portland, Ore. SIGMOD Record 18(2), 1989, 377-388.

HMMS87 Härder, T., Meyer-Wegener, K., Mitschang, B., Sikeler, A.: PRIMA - A DBMS Prototype Supporting Engineering Applications, P. M. Stocker, W. Kent (eds), Proc. 13th VLDB Conf., Brighton, UK, Morgan Kaufmann Publishers, Palo Alto, CA, 1987, 433-442.

HMNR95 Härder, T., Mitschang, B., Nink, U., Ritter, N.: Workstation/Server Architectures for DB-based Engineering Applications (in German), Informatik - Forschung und Entwicklung 10:2, 1995, 55-72.

HR83 Härder, T., Reuter, A.: Principles of Transaction Oriented Database Recovery, ACM Comp. Surveys 15:4, 1983, 287-317.

HR85 Härder, T., Reuter, A.: Architektur von Datenbanksystemen für Non-Standard-Anwendungen (in German), A. Blaser, P. Ristor (eds), Proc. GI-Fachtagung "Datenbanksysteme in Büro, Technik und Wissenschaft", Karlsruhe, IFB 94, Springer, 1985, 253-286.

HR93 Härder, T., Rothermel, K.: Concurrency Control Issues in Nested Transactions, VLDB-Journal 2:1, 1993, 39-74.

HS93 Hong, W., Stonebraker, M.: Optimization of Parallel Query Execution Plans in XPRS, Distributed and Parallel Databases 1, 1993, 9-32.

IEEE92 Eich, M. (ed.): IEEE Trans. Knowledge and Data Engineering 4:6, Special Issue on Main-Memory Databases, 1992.

In84 IntelliCorp Inc.: The Knowledge Engineering Environment, IntelliCorp, Menlo Park, CA, 1984.

ISO94 ISO 10303 - Industrial automation systems and integration - Product data representation and exchange - Part 1: "Overview and fundamental principles", International Standard, 1st edition, 1994.

ISO96 ISO/IEC CD 9075 Committee Draft, Database Language SQL, Jim Melton (ed.), 1996.

JGJSE95 Jarke, M., Gallersdörfer, R.,Jeusfeld, M., Staudt, M., Eherer, S.: ConceptBase - a deductive object base for meta data management, Journal of Intelligent Information Systems 4:2, Special Issue on Advances in Deductive Object-Oriented Databases, 1995, 167-192.

KB96 Keller, A., Basu, J.: A Predicate-based Caching Scheme for Client-Server Database Architectures, VLDB Journal 5:1, 1996, 35-47.

Ki95 Kim W. (ed.): Modern Database Systems: The Object Model, Interoperability, and Beyond, ACM Press, 1995.

KL89 Kifer, M., Lausen, G.: F-Logic, a Higher-Order Language for Reasoning about Objects, J. Clifford, B. G. Lindsay, D. Maier (eds), Inheritance and Schema, Proc. ACM SIGMOD, Portland, Ore. SIGMOD Record 18(2), 1989, 134-146.

LLOW91 Lamb, C., Landis, G., Orenstein, J., Weinreb, D.: The ObjectStore Database System, in: [Ca91], 50-63.

LLPS91 Lohman, G. Lindsay, B., Pirahesh, H., Schiefer, B.: Extensions to Starburst: Objects, Types, Functions, and Rules, Comm. ACM 34:10, 1991, 94-109.

Lo95 Lomet, D. (ed.): Bulletin of the Technical Committee on Data Engineering 18:2, Special Issue on Materialized Views and Data Warehousing, 1995.

Ma91 Mattos, N.: An Approach to Knowledge Base Management, LNCS 513, Springer, 1991.

Mi88 Mitschang, B.: A Molecule-Atom Data Model for Non-Standard Applications - Requirements, Data Model Design, and Implementation Concepts (in German), IFB 185, Springer, 1988.

MPTW94 Mohan, C., Pirahesh, H., Tang, W., Wang, Y.: Parallelism in Relational Database Management Systems, IBM System Journal 33:2, 1994, 349-371.

ODMG96 Cattell, R. (ed.): The Object Database Standard: ODMG-93, Release 1.2, Morgan Kaufmann., 1996.

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

Re97 Rezende, F.: Transaction Services for Knowledge Base Management Systems - Modeling Aspects, Architectural Issues, and Realization Techniques, Doctoral Thesis, University of Kaiserslautern, 1997.

RB96 Rezende, F., Baier, T.: WALORS - A WAL-Based and Object-Oriented Recovery Strategy, R. Wagner, H. Thoma (eds), Proc. 7th Int. Conf. on Database and Expert Systems Applications (DEXA'96), Zürich, Switzerland, LNCS 1134. Springer, Heidelberg Berlin New York, 1996, 116-129.

RHGL97 Rezende, F., Härder, T., Gloeckner, A., Lutze, J.: Detection Arcs for Deadlock Management in Nested Transactions and their Performance, C. Small, P. Douglas, R. G. Johnson, P. J. H. King (eds), Proc. 15th British National Conference on Databases (BNCOD'97), London, U.K., LNCS 1271, Springer, Heidelberg Berlin New York, July 1997, 54-68.

RH95 Rezende, F., Härder, T: Concurrency Control in Nested Transactions with Enhanced Lock Modes for KBMSs, Proc. 6th Int. Conf. and Workshop on Database and Expert Systems Applications, London, LNCS 978, Springer, 1995, 604-613.

RH97 Rezende, F., Härder, T.: Exploiting Abstraction Relationships' Semantics for Transaction Synchronization in KBMSs, Journal of Data and Knowledge Engineering (DKE) 22:3, 1997, 233-259.

Ro91 Roussopoulos, N.: An Incremental Access Method for ViewCache: Concept, Algorithms, and Cost Analysis, ACM Transactions on Database Systems 16:3, 1991, 535-563.

SK91 Stonebraker, M., Kemnitz, G.: The POSTGRES Next-Generation Database Management System, Comm. ACM 34:10, 1991, 78-93.

ST95 Sauter, G., Thomas, J.: An object-oriented approach to structuring business information processing (in German), G. Lausen (ed), Proc. GI-Tagung "Datenbanksysteme in Büro, Technik und Wissenschaft", Dresden, Germany. Springer, Heidelberg Berlin New York, 1995, 348-357.

TD93 Thomas, J., Deßloch, S.: A Plan-Operator Concept for Client-Based Knowledge Processing, R. Agrawal, S. Backer, D. A. Bell (eds), Proc. 19th VLDB Conf., Dublin, Ireland. Morgan Kaufmann Publishers, Palo Alto, CA, 1993, 555-566.

TDM95 Thomas, J., Deßloch, S., Mattos, N. M.: Design and Implementation of Advanced Knowledge Processing in the KBMS KRISYS, M. J. Carey, D. A. Schneider (eds), Proc. ACM SIGMOD, Exhibits Program, San Jose, CA, SIGMOD Record 24(2), 1995.

TGHM95 Thomas, J., Gerbes, T., Härder, T., Mitschang, B.: Implementing Dynamic Code Assembly for Client-Based Query Processing, T. W. Ling, Y. Masunaga (eds), Proc. 4th Int. Conf. on Database Systems for Advanced Applications, Singapore. World Scientific Press, Singapore, 1995, 264-272.

Th96 Thomas, J.: An Approach to Query Processing in Advanced Database Systems, Doctoral Thesis, Computer Science Dept., University of Kaiserslautern, 1996.

TMMD93 Thomas, J., Mitschang, B., Mattos, N., Deßloch, S.: Enhancing Knowledge Processing in Client/Server Environments, B. K. Bhargava, T. W. Finin, Y. Yesha (eds), Proc. 2nd Int. Conf. on Information and Knowledge Management (CIKM'93), Washington, D.C. ACM Press, New York, 1993, 324-334.

VD91 Vandenberg, S.L., DeWitt, D.J.: Algebraic Support for Complex Objects with Arrays, Identity, and Inheritance, J. Clifford, R. King (eds), Proc. ACM SIGMOD, Denver, Colo. SIGMOD Record 20(2), 1991, 158-167.