appeared in: Proc. of the Int. Conf. on Data and Knowledge Systems for Manufacturing and Engineering
Gaithersburg - Maryland - Oct. 1989

KRISYS: KBMS Support for Better CAD Systems

S. Deßloch, T. Härder, N. Mattos, B. Mitschang

University Kaiserslautern, Department of Computer Science,
P.O. Box 3049, D-6750 Kaiserslautern, West Germany

Abstract

Here, we propose a so-called knowledge base management system based on the integration of database and artificial intelligence technology that provides adequate capabilities for the construction of advanced CAD systems. Our approach is evaluated in view of other systems and its suitability is demonstrated by means of some examples from the area of architectural design. Throughout the paper, we refine our view of advanced CAD and define some inherent characteristics of better CAD.

1. Introduction

Currently, many research investigations work for more appropriate design methodologies and more effective design tools. Their results, the so-called *advanced* computer-aided design systems are capable of providing an intelligent interface to the user:

- dealing with incomplete design information

  Object specifications may be incompletely defined by the user. The system supplements them by exploiting domain knowledge.

- exhibiting an active behavior

  The system is able to provide the design engineer with relevant solutions or hints to current design problems, complete object specifications, refined calculation and simulation results, and adequate diagnoses at all stages of the design process.

- guaranteeing area-spanning integrity constraints

  At each operation, the system takes into account all design aspects as well as all kinds of dependencies and restrictions that are involved in the design process (manufacturing, production, company guidelines, time considerations etc.) in order to guarantee a consistent design.

In this new environment, the design process is done iteratively as shown in figure 1. In contrast to conventional systems, we have no stand-alone tools converting the output data structures of previous design steps into internal data structures and vice-versa before they can accomplish their functions. Our approach divides the design process into design phases, which subsequently

- supplement a previously incomplete design object specification,
- add structure and geometry information,
- consider functional, physical, and technological constraints, and
- support improvements or refinements by means of feedback to each previous phase.

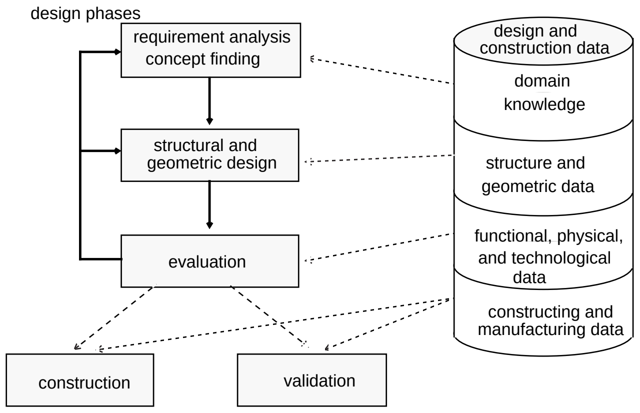

design phases
requirement analysis concept finding
structural and geometric design
evaluation
construction
validation
design and construction data
domain knowledge
structure and geometric data
functional, physical, and technological data
constructing and manufacturing data

Figure 1: Overall design process

Since all design phases refer to the same design objects, all aspects of these objects have to be represented in a unique and non-redundant manner, allowing for a consistent and uniform object handling. Obviously, the efficient and reliable management of such a product model (that includes all different aspects of design objects in one database schema) addresses the main functions of database management systems. On the other hand, artificial intelligence techniques (knowledge representation, reasoning, etc.) allow for both semantically enriched design object descriptions and for an active system behavior. Therefore, a practical approach to advanced CAD environments should incorporate the advantages of both database (DB) and artificial intelligence (AI) techniques. However, such an approach should by no means be based on an extension or coupling of existing systems. The lack of component integration in existing architectures is responsible in most cases for cumbersome handling and for quite ineffective performance [1]. For this reason, the approach that is described here is centered around a so-called *knowledge base management system* (KBMS) that integrates AI and DB techniques in an effective way.

Using the KBMS KRISYS [2], that was developed at our university, and some examples from an architectural design application, we demonstrate the applicability of our approach. KRISYS (Knowledge Representation and Inference SYStem) integrates concepts from the fields of AI and DB in order to provide the desirable support of knowledge modeling, manipulation, and management tasks. A mixed knowledge model offering a rich spectrum of constructs (e.g., object-centered representation, abstraction concepts, rules, demons) allows for an accurate representation of the application domain. The implementation of this knowledge model uses the enhanced DBS PRIMA to provide features such as knowledge persistence, efficient secondary storage management, fault tolerance, etc. After having introduced a sample design application, we show how the different modeling concepts offered by KRISYS could be conveniently exploited to accomplish the main design goals of advanced CAD. Simultaneously, we make apparent that other presumably conceivable environments do not fulfil all requirements and that the optimal solution could be achieved only by an effective integration of both DB and AI techniques. Finally, we conclude with a more comprehensive definition of our notion of advanced CAD, whereby we also indicate the main topics we are going to investigate in the near future.

2. A Short Overview of the Application

As an example of an advanced CAD-system supported by KRISYS, we use an architectural design application currently being implemented. The task of this system is to design one-story houses for families based on requirements and needs specified by the user. During a first design-phase, it questions the user as to his requirements, which may be formulated on a high semantic level (e.g., "I need a room to work in, which should be located at the south side of the house."). Furthermore, the specification of such requirements may be incomplete in the sense that they might not be sufficient to directly achieve a final design stage. For this reason, the system utilizes its knowledge (e.g. standard requirements, laws, etc.) to supplement the user-sketched blueprint, creating additional rooms and properties of rooms and constructing a functional description of the house. In a second phase, our application uses this description to generate an architectural sketch. The user may accept the design sketch or reject it, causing the system to generate alternative sketches. New requirements can also be added, or existing ones can be removed, leading to a (partial) repetition of the first design phase. Different design alternatives are kept and may be explained to the user until he accepts a certain sketch. The system also detects inconsistencies caused by incompatible requirements or violation of laws upon which the user is notified.

3. Using KRISYS to Model Advanced CAD-Systems

One group of features provided by KBMS to support better CAD-systems is related to knowledge modeling aspects, i.e., the constructs and mechanisms available for the description of the application world. What can be considered as one of the major goals of advanced CAD is the improvement of the overall design process by incorporating different design aspects (e.g. geometrical, functional, manufacturing) which are related to different processing or design phases but are all integrated into the same design object. The dependencies between these distinct representations have to be maintained and exploited by the system in order to achieve a 'manufacture-oriented' design. Therefore, significant parts of the semantics of the application world, which used to be buried in various programs responsible for different design phases, need to be represented explicitly in an integrated product model, where they are available for all system components. As a consequence, knowledge modeling aspects can be considered a key issue in the support of advanced CAD [3].

3.1 Modeling Aspects

Object structure

KOBRA (KRISYS OBject-Centered RepresentAtion), the knowledge model of KRISYS, provides an *object-centered representation*. That is, everything existing in the application domain is expressed as an object of the KOBRA model, the so-called *schema*, in which descriptive, operational, and organizational aspects of the real world are integrated. For this reason, an entity of the application world directly corresponds to an object of the knowledge base (KB). A schema, which is roughly analogous to a frame or unit in other representation systems, may contain *attributes* for the description of the real-world entity. The attributes may again be further described by *aspects* in order to characterize an object in more detail. In figure 2, we give an example of a schema representing a certain room of a house in our architectural application. Its properties are described by the attributes and attribute values: the size of the room is 16 square-meters, it is intended to be used for sleeping and is adjacent to two other rooms, namely 'room 2' and 'room 3'. The orientation and position of the room are not yet specified or determined. Via the aspect 'unit' for the 'size'-attribute, 'square-meters' is fixed as unit for the size of the room.

room 1		
orientation		-
position		-
size		16
	unit	square-meters
neighbors		room 2, room 4
usage		sleeping

Figure 2: Sample schema description

KB organization

For structuring the KB, KOBRA supports the abstraction concepts of classification, generalization, association, and aggregation [4,5,6,7]. These concepts are seen as special, predefined relationships between objects, defining the overall organization of a KB. Figure 3 gives a partial overview of our architectural KB considering the classification and generalization relationships. In this hierarchy, the object 'room 1' is seen as an instance of the class 'parents-bedrooms', which is a subclass of 'bedrooms', which is again a subclass of 'rooms', etc.

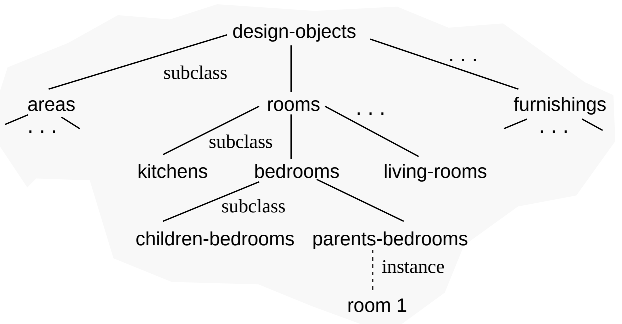


Figure 3: Generalization/classification hierarchy

Another view of the KB is given in figure 4, where aggregation relationships involving 'room 1' are presented. 'Room 1' is a component of the 'parent-area' and again has parts like 'bed 1' or 'wardrobe 1', representing pieces of furniture for the room.

The concept of association, which can be used to group heterogeneous objects together, is demonstrated in figure 5. The user of our application is able to specify the furnishings that he already possesses and are to be moved into the new house. This can be very important for the design of the rooms into which the furniture has to be placed. Therefore, two set objects are introduced to distinguish the belongings of the user from the furnishings proposed by the application. 'Bed 1' is an element of the set 'user's-furnishings',

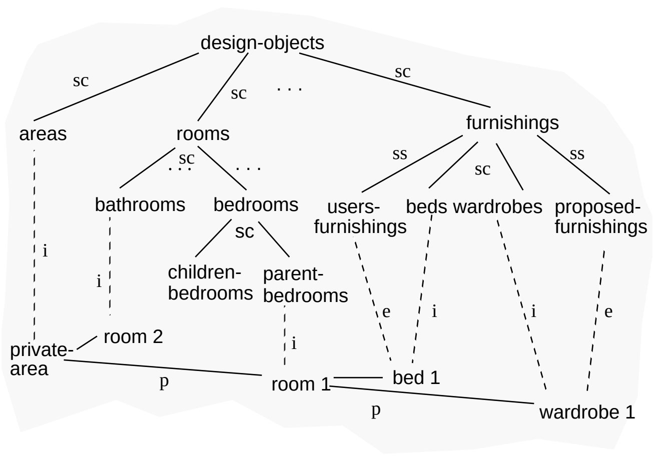
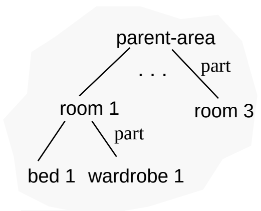


Figure 6: An integrated view of the KB

while 'wardrobe 1' is an element of 'proposed-furnishings'. Both sets are subsets of 'furnishings'.



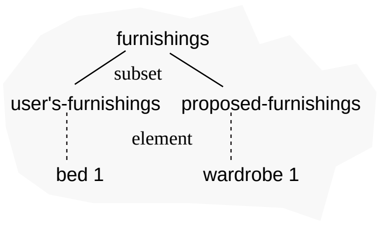


Figure 4: Aggregation hierarchy

Figure 5: Association hierarchy

The semantics of the abstraction concepts is guaranteed by the system via so-called *built-in reasoning facilities* [4]. E.g. inheritance, which is carried out according to the generalization/classification-relationships, is the reasoning as to the structure of an object. 'Room 1' in figure 3 inherits all properties of 'parents-bedrooms', 'bedrooms', 'rooms', and 'design-objects'. The aggregation-relationships are the basis for reasoning with so-called implied predicates. For example, the fact that 'room 1' has a size of 16 square-meters implies that the area in which it is contained must be at least 16 square-meters in size. This information is automatically deduced by KRISYS. The concept of association allows the definition of set properties. For the final costs involved with the new house, the prices of all furnishings proposed by the system have to be considered. The set property 'total-price' of 'proposed-furnishings' is defined as the sum of all the prices of its elements. Upon changes, like the insertion of a new piece of furniture into the set, the new total price is automatically recalculated by KRISYS. The KB structure defined by the abstraction concepts is not restricted to hierarchies. It is also possible to specify several classes for one instance or multiple superclasses for a class.

KRISYS supports an *integrated view* of the KB: first of all, there are no separate representations for classes, instances, sets, aggregates, etc. - all are represented as a schema. Also, the different meanings of an entity are integrated into one schema. A partial view of our example KB is presented in figure 6. The schema 'bed 1' represents, for example, three different 'roles' at the same time: it is an instance of 'beds', an element of 'users-furnishings' as well as a component of 'room 1'.

Supporting an integrated product model

An object-centered representation can be directly utilized for the definition of an integrated product model. It allows a natural description of the design object, integrating all aspects of the product into one KB-object. The abstraction concepts can be used as the basic mechanisms for describing the organizational semantics of the application domain. Distinct aspects of the design object may be modeled using different abstraction concepts or distinct hierarchies of the same concept (e.g. one class hierarchy for representing geometrical, another for functional, and a last one for manufacturing information about the object), which has the advantage that they are easily distinguishable in the model. An integration of all aspects into one object is easily achieved by overlaying the corresponding hierarchies. An object may, for example, be an instance of several classes, each belonging to a different hierarchy. It may also be a component of another object or a member in a set object. The object-centered representation and the semantics of the abstraction concepts appear to be very beneficial for the support of the application. If, for example, a design object is deleted, because the design engineer regards it as useless, KRISYS automatically deletes all aspects of the object (i.e. the different views of the object) as well as all object components, updates set properties (if necessary), and performs other built-in reasoning operations related to the semantics of the abstraction concepts.

In our architectural design system, we have described aspects concerning the usage, adjacency, and geometry of objects within three distinct classes (see figure 7). The class 'usage-objects' describes all aspects related to the usage of a design-object (e.g. a room) or the activities associated with it. A kitchen may, in this case, be described by the activities 'preparation of food' and - probably - 'eating'. Information about the adjacency of objects (e.g. when two rooms are connect via a common door) is captured by the class 'adjacency-objects', while all geometrical aspects, such as the size or the position of a room within the house, are covered by 'geometrical objects'. The class 'design-objects' inherits all attributes of its superclasses and therefore integrates the different aspects related to the design object. Since the design objects, like areas, rooms and furnishings are also embedded in an aggregation and an association hierarchy, their role as complex objects, object components, or elements are also considered. If the user has specified requirements about an area, (e.g. the children-area) and wants the whole area to be redesigned because he has changed his opinion, the architecture application only needs to delete the object 'children-area'. KRISYS automatically deletes all representations of the object and its subparts (e.g. different rooms and furniture proposed by the system) and additionally updates the 'total-price' in the set 'proposed-furnishings'. The user is then free to enter new requirements for the house.

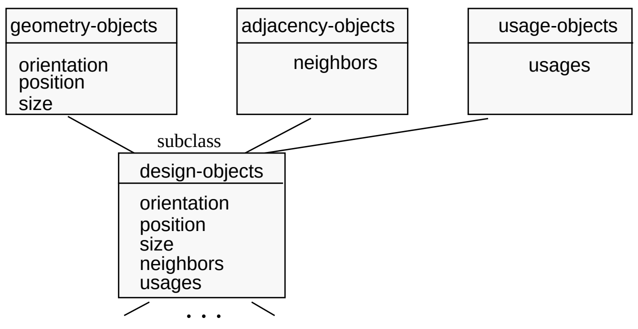


Figure 7: Different aspects of the design objects

Integrating behavior into the application model

So far, we have only considered how the KOBRA model can be used to describe properties and relationships of objects as well as the organizational structure of a KB. The behavior of objects or operations and actions in which they are involved may also be included into a schema description as procedural attributes. For this reason, the KOBRA model distinguishes between *slots*, which are declarative attributes used to describe properties and relationships of an object, and *methods* denoting procedural attributes for modeling object behavior.

Our application utilizes methods for several tasks. For example, the algorithms performing the actual geometric design, i.e. partitioning the house into geometric areas corresponding to the rooms (once their usage, functionality, and other requirements are fixed) can be modeled as operations of 'geometry-objects'. The end-user of the system is of course not directly concerned with operations upon geometrical representations. His interaction with the application is usually settled at a higher semantic level. E.g. the method 'add-neighbors', defined in 'adjacency-objects', is performed when the user adds new requirements about the connections between rooms currently under design. Since these new requirements can invalidate previously stated geometric design decisions, the method 'add-neighbors' may, in turn, call methods defined in 'geometry-object' to cause a redesign of certain areas or of the whole house.

Similarly, the advantages of behavioral object-orientation become apparent in systems using form features for solid modeling. The designer is not interested in the geometrical representation (e.g. BREP, CSG) of an object or the operations associated with it (intersection, union of objects, ...), but thinks in terms of holes, slots, sockets, etc., and of operations like 'drilling a hole'. Object properties and operations of both geometrical and feature-logical nature can be integrated into the product model, where changes of the form-feature representation via feature-operations are then directly transformed into operations of geometrical nature.

To include additional information into the KB and make the integrated product model more powerful, one may also integrate knowledge about the environment, such as production and manufacturing information, into the KB, using the same modeling constructs for this task, as we have described in the above text. A machine for manufacturing a design object can be represented as a schema containing machine properties such as the tolerance guaranteed by it, and methods corresponding to machine operations. This information could be directly utilized for production planning or generating NC-programs.

Maintaining the semantic integrity

Until now, we have described how KOBRA supports the representation of structural, behavioral, and organizational knowledge of an application domain. The introduced concepts may be directly utilized for an integrated product model describing various aspects of the design product and its environment. Nevertheless, a significant part of the application world semantics is embodied in restrictions of and dependencies between certain aspects of the world. These are usually described as integrity constraints applied to ensure that changes in the KB always result in a semantically correct state. These constraints may involve certain granules of knowledge, such as attributes (e.g., restricting the set of permitted values), objects (e.g., maintaining the integrity between different views of the same object), or groups of objects (e.g., relationships between the design object and environmental information like "Can the design object be manufactured with the available machines?").

```
design-objects
orientation        -
position           -
size               -
        cardinality [1  1]
        possible-values > 2
        unit        square-meters
neighbors          -
        cardinality [1  ∞]
usage
        possible-values (INSTANCE-OF activities)
```

Figure 8: Specifying constraints by using predefined aspects

KRISYS provides several mechanisms for explicitly describing integrity constraints and integrating them into the application model. Firstly, the value class and the cardinality of slot values may be restricted with the use of two predefined aspects, 'cardinality' and 'possible-values', which may be defined individually for each attribute. Changes of the attribute value violating the given restrictions are automatically prohibited by KRISYS. Figure 8 gives an example using the class 'design-objects'. Every design-object is described by exactly one size, which is expressed through the value '[1 1]' of the cardinality aspect. A room also has to have at least one neighbor, otherwise it is not accessible. The maximum number of neighbors is, on the other hand, not limited since a corridor, for example, can have quite a lot of neighbor rooms. Using the 'possible-values' aspect, we can state, that the minimum size of a room is 2 square meters. This aspect can also be used to ensure a constraint similar to the referential integrity in database systems: in our architectural design application, we have represented the design objects, such as rooms or furniture, and the usage of the objects (e.g. sleeping, cooking) in separate hierarchies. This is necessary because a lot of information can be directly associated with usage or activities, while activities and rooms are not always related in a one to one correspondence. As an example, the activity 'eating' may be performed in the dining room and in the kitchen, which, in turn, is also related to the usage 'preparation of food'. In order to assure that the values of the usage-slot, which represents the relationship between a design-object and its usages, always reference an actual instance of the class 'activities', we assert '(INSTANCE-OF activities)' as the value of the appropriate possible-values aspect.

More complex integrity constraints can be expressed using so-called *demons*, i.e. procedures attached to attributes, which are activated when the attributes are accessed. Distinct actions can be specified for different types of access (get, put, add, ...) making it possible to define flexible reactions on different events. For example, our application has to check whether an activity specified for a specific room is compatible with previously asserted usage. There are statutes that strictly determine incompatible usage. The architect has, for example, to consider that facilities for both cooking and bathing must not be provided in the same room. For this reason, we define a slot called 'contradictory-activities', which contains all incompatible activities for every activity in the KB. A demon is then attached to the usage slot of every room (see figure 9), which is activated when a usage is added in order to check if one of the old usages is a 'contradictory activity' of the new one. Upon detecting an inconsistency, an error is reported, automatically eliminating the previous change of the KB.

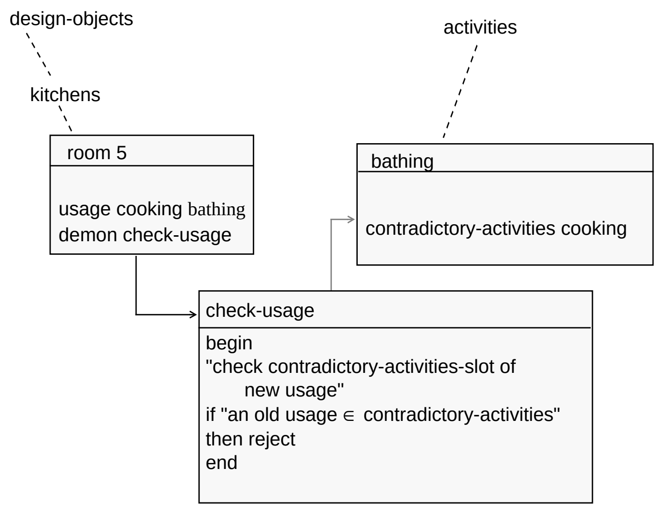


Figure 9: Using demons for checking integrity constraints

One way to guarantee the consistency of a KB is to reject any changes that would violate integrity constraints, as we have already demonstrated in the above text. Another approach, that may very often be taken, is to 'trigger' additional operations upon changes violating the integrity that transform the inconsistent KB into a consistent one. This is also desirable within an integrated product model, where changes concerning one representation of an object (e.g. the functional aspects) should automatically cause changes in the other representations (e.g. the geometry). In KOBRA, this approach may be realized either with demons to trigger the appropriate changes or with the concept of rules, which can be used to dynamically derive information and insert it into the KB. In KRISYS, *rules* are special kinds of schemas containing as attributes the conditions (i.e. 'if'-part) and the actions (i.e. 'then'-part) of the rule. Basic inference strategies (forward-chaining, backward-chaining) are also provided by the system.

The architectural system utilizes rules extensively for keeping the KB consistent. As we have already mentioned, the user is able to interact with the application at a very high semantic level. He may, for example, specify activities or usage for the house, but need not relate them to any rooms. This task is then automatically performed by the system. E.g. when the user specifies the activity 'working', the system activates rules associated with the 'working' object in the KB that either establish a relationship between the activity and an already existing room (e.g. the living-room), considering, for example, extra places for a work desk, or generate a new room (e.g. the office room) for the activity (see figure 10).

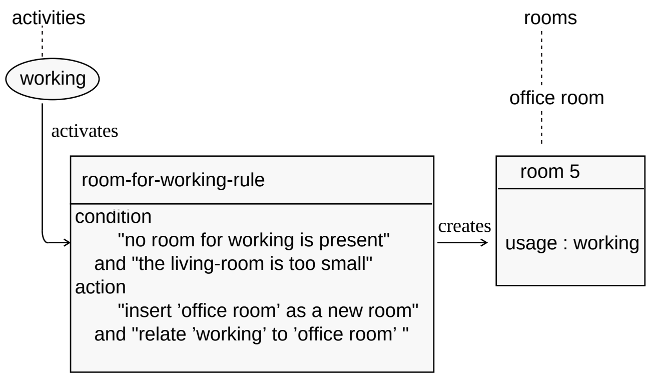

activities
working
activates
rooms
office room
room-for-working-rule
condition
"no room for working is present"
and "the living-room is too small"
action
"insert 'office room' as a new room"
and "relate 'working' to 'office room' "
creates
room 5
usage : working

Figure 10: Using rules to keep the KB consistent

Supporting an active system behavior

In general, demons and rules are constructs to achieve an active system behavior. An intelligent CAD system should support and guide the user throughout all design phases, suggesting alternative product designs or possible solutions to his design requirements. A significant part of the knowledge of our architectural design system is represented as rules, which are responsible for the activities of the system. Since the requirements of the user may be incomplete, the design system uses, for example, rules to derive rooms and activities that are standard in every house (e.g. kitchen, bathroom, ...) and are not specified by the user, as described above (see also figure 10). The location of a room within a certain area (e.g. bathroom, bedroom are parts of the private-area) and the connections between rooms (e.g. a kitchen is connected to the dining-room, but not to the bathroom) can also be inferred as well as the size of a room, which strongly depends on the activities and the number of people using the room. Standard sizes of doors and windows as well as prices of material and equipment are considered in order to keep the financial restrictions given by the user at the beginning of his session.

All the information specified during the session or derived by the application is then used to construct an architectural sketch of the house. The user may accept this sketch or refuse it, which causes the design system to present alternative sketches. Upon additional requirements posed by the user, it has to repeat previous design phases and probably revise decisions made earlier. An explanation facility is also provided for making design decisions plausible to the user.

The actual sketch or geometric design of the house is represented in the KB as a tree-like structure, where every node of the tree corresponds to a design decision. The derivation of design alternatives is performed in a non-redundant way by backtracking in this tree and generating alternative decision nodes, thus utilizing previous design steps. Different design alternatives are kept in the KB until the user makes his decision. The explanation of the sketch is also easily performed via the tree-like representation. After the design session is finished, the resulting design is kept in the KB. A user may therefore start a new session with the result of a previous one as the basis for his new design.

Extending the integrated product model

A last important issue is related to the notions of extensibility. In KRISYS, there is no difference between information and the meta-information describing the application model - contrary to the DB-like view, which strictly separates the DB from the DB-schema. All information is contained in the KB and may be modified. It is therefore easy to keep track of changes in the application environment because the application model may easily be extended or changed. Since a significant amount of the application semantics is represented in the model, the application programs are to a large extent immune to these changes. Also, the modeling facilities provided by KOBRA, i.e. constructs and operations used for specifying the application model, are available to the application as well. An intelligent CAD system may therefore easily provide user operations for defining new object types and extending or browsing through catalogues since these operations are already supported by KRISYS.

As a consequence of the notion of extensibility presented above, our architectural design application may be extended to include the design of many-story houses or other kinds of buildings (e.g. bungalows, villas) in a natural way. Also operations could be provided, that allow the end-user to define a new class of rooms with special properties (sound insulation), that he might require for his house.

3.2 KOBRA Features Emphasized

After presenting the modeling concepts provided by KRISYS, we would like to summarize the features of its knowledge model, thereby making a comparison with the concepts supported by other existing systems (DBS and XPS tools like ART [8], KEE [9,10], KNOWLEDGE CRAFT [11], and LOOPS [12]) as well as with the way these concepts may be applied.

Integration of declarative, procedural, and structural knowledge

In KRISYS, the effective support of modeling requirements is achieved by a mixed knowledge representation framework which is not usually found in existing DBS or XPS tools. Whereas these systems emphasize either a declarative, procedural, or structural view of the world, KOBRA remains neutral, equally focusing on all of them. It integrates all different aspects of knowledge, consequently obtaining the powerful framework necessary to enable a natural and accurate modeling of all aspects of the application world.

KRISYS allows for a representation of the whole descriptive characteristics of the application domain, i.e., objects, properties, relationships, and constraints. It offers constructs to represent operational characteristics of the application world which are either used to model the behavior of the domain objects (methods) or to specify complex integrity constraints (rules and demons). The most important aspect of the operational concepts provided by KRISYS is their use to achieve an intelligent and active system behavior. Finally, KRISYS supports the most significant abstraction concepts (classification, generalization, association, and aggregation), enriching considerably the semantics of its knowledge model with their different built-in reasoning facilities.

Complete support of abstraction concepts

The support of all four abstraction concepts is a further aspect that differentiates KRISYS from the above mentioned systems. Whereas KOBRA puts special emphasis on a complete support of the abstraction concepts in order to use their semantics as the basis for drawing conclusions about the objects and for maintaining the integrity of the knowledge base, these systems neglect the existence of some of these concepts (generalization by DBS, association by DBS, KEE, and LOOPS, and aggregation by all of them), forcing a substantial amount of real world semantics to be maintained in the application programs, thereby severely weakening the expressiveness and the semantic power of their knowledge or data model. Furthermore, since KOBRA incorporates all roles of a real world entity in one single schema, there is no need to introduce two distinct representations to support both association and generalization/classification (as done by ART and KNOWLEDGE CRAFT) in the model. It is also not necessary to make a kind of "hodgepodge" with the semantics of the generalization/classification in order to be able to support the representation of set properties (as done by KEE). Even proposed extensions of DBS technology (semantic data models [13,14], object-oriented data models [15], etc.) focus only on some of these concepts (mostly aggregation and/or generalization/classification) and neglect most of the underlying reasoning facilities.

Dynamic built-in reasoning

Finally, KRISYS allows for changes on the abstraction relationships at any time. The corresponding built-in reasoning is, in such cases, automatically applied maintaining the KB in a semantically consistent state. In some systems (ART and LOOPS), inheritance, for example, is not more than a means to save typing work. Changes in the structure of a class are either not allowed or not reflected in the existing instances until a latter system recompilation, leading to severe inconsistencies in the meantime. In KRISYS, when relevant information has been changed, inheritance as well as the other built-in reasoning facilities are recalculated so that the system can guarantee the structural and semantic integrity of the knowledge base.

Integration of meta-information into the KB and extensibility

As a consequence of the previously discussed integrated and dynamic view of the abstraction concepts, KOBRA does not differentiate between data and meta-data. Such a differentiation (required by DBS) does not reflect the situation in the real world since changes affecting the application model may also occur. Thus, it is possible to insert an object into the KB and then dynamically establish or change abstraction relationships or directly introduce new information into its structure. This capability is particularly important in achieving the above mentioned extensibility of the application model.

In summary, KRISYS supports constructs and mechanisms that allow for an integration of a much higher degree of application semantics into the model, thereby showing several advantages in comparison to other systems:

- Structural object-orientation (i.e., the representation of complex objects) is supported by means of the abstraction concept of aggregation.
- Behavioral object-orientation is achieved by the integration of procedural knowledge into the object description (methods).
- Data-driven and demand-driven computation are provided by the concept of demons allowing for the specification of reactions to certain events.
- Intensional knowledge may be represented by rules (also by demons) in order to dynamically derive new knowledge.
- Organizational principles are provided in an integrated fashion by abstraction concepts. The semantics of these concepts are incorporated into the system via built-in reasoning facilities, which guarantee the structural and application-independent semantic integrity of the KB.
- Application-dependent semantic integrity is explicitly specified and maintained via aspects, demons, and rules.

4. Our KBMS Approach

After having introduced the most important modeling concepts of KRISYS, with which the whole knowledge of an application world may be suitably represented, we now want to focus on performance-increasing measures in order to reach an efficient management of the KB. For this reason, it is necessary to develop an appropriate system architecture, a corresponding mapping concept, and an effective processing model. In the following, we want to concentrate on each of these issues and their implementation concepts as applied in the KRISYS KBMS.

4.1 Overview of the KRISYS Architecture

As shown in figure 11, the system architecture of KRISYS is divided into three hierarchically ordered layers. They are responsible for a stepwise abstraction process and for the realization of the corresponding tasks within each layer. In the previous section, we have described in detail the knowledge model KOBRA that is implemented by the engineering layer. The KOBRA implementation offers at its interface an object-centered view of knowledge representation and manipulation to the knowledge engineer. To keep the end-user or application programs independent from this representation, the application layer realizes an external interface where the knowledge is viewed in a more abstract manner. This object-abstraction interface is achieved by the powerful query language KOALA [16]. The goal of the lowest layer is to efficiently cope with storage of knowledge and its supply to the other layers. At this level, most of the issues are related to traditional DB technology applied to large KB, possibly shared by multiple users: storage structures, access techniques, efficiency, integrity features, transaction support, etc. Therefore, this layer is realized by a non-standard database system (NDBS) which seems to be quite advantageous in a KBMS architecture for a number of reasons [17]. NDBS kernels are much more powerful than traditional DBS and are, for this reason, able to satisfy knowledge maintenance requirements.

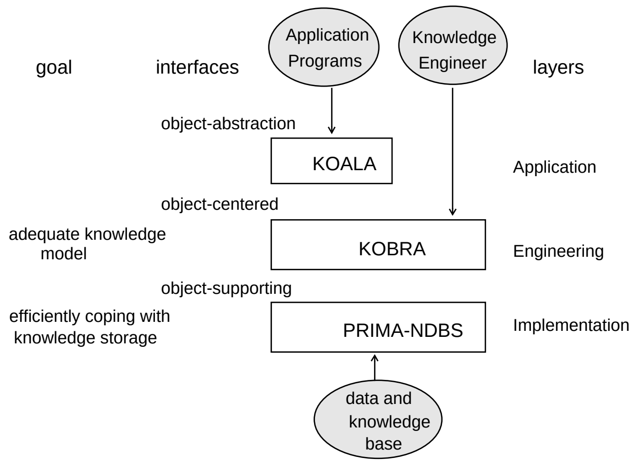


Figure 11: Overall system architecture of KRISYS

4.2 The Implementation Layer

KRISYS makes use of the PRIMA-NDBS [18], also developed at our university, to realize the implementation layer. It offers neutral, yet powerful mechanisms for managing the KB. PRIMA is the PRototype Implementation of the MAD model, whereas the MAD model is an acronym that stands for Molecule Atom Data model [17]. This non-standard data model offers dynamic definition and handling of complex objects based on direct and symmetric management of network structures and recursiveness. These concepts enable the mapping of knowledge structures in an effective and straightforward way as we are going to illustrate in the following sections.

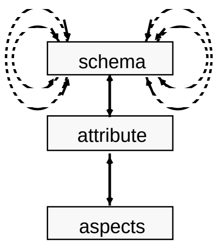
Short notes on the MAD model

The most primitive constructs of MAD are denoted atoms. They are, in analogy to tuples in the relational model, composed of attributes of various types, have their structure determined by an atom type, and possess an identifier. Atoms represent components of more complex objects, called molecules. Composition as well as the decomposition of molecules are performed dynamically, following the specification of a molecule type. A molecule type is defined as a graph having atom types as nodes and relationship types as edges. The specification of relationship types is made in attributes of the atom types by using a special attribute type called reference. In MAD, each relationship type is symmetrically modeled. That is, for each reference attribute, there exists in the pointed atom type a back-reference attribute, whose mutual referential consistency is automatically maintained by the system. MAD allows for a direct representation of all existing relationship types (i.e., 1:1, 1:n, and n:m) achieved by a combination of attribute type reference with the attribute type set_of. In this sense, atoms together with their references build the molecules that may be dynamically defined. Molecules are views or snapshots, in which atoms are natural-joined on their references (foreign keys). Such views may, in turn, be used to construct other more complex molecules, which may even occur recursively. Therefore, the specification of molecule types can be defined in such a way that they directly allow for an appropriate mapping of knowledge structures to data model structures as described below. From a more general point of view, the MAD model is a direct implementation of the well-known entity-relationship model (cf. figure 12).

Mapping KOBRA objects into MAD

In the previous chapter, we have seen that KOBRA presents all its concepts for building a KB well integrated in its central construct: the schema that renders an object-centered representation possible. From a structural point of view, one may observe that the schema (cf. figure 1) is composed of attributes which, in turn, consist of aspects, suggesting a MAD schema that contains three atom types (corresponding respectively to 'schema', 'attribute', and 'aspects') connected via the references has_attributes and has_aspects. The

MAD schema diagram in figure 12 shows a graphical view of atom types (rectangles) and their interconnections (double arrows).



a) MAD schema diagram

```
CREATE ATOM_TYPE        schema
  ( schema_id         :  IDENTIFIER,
    name              :  CHAR_VAR,
    is_subclass_of    :  SET_OF(REF_TO(schema.has_subclasses)),
    has_subclasses    :  SET_OF(REF_TO(schema.is_subclass_of)),
    is_instance_of    :  SET_OF(REF_TO(schema.has_instances)),
    has_instances     :  SET_OF(REF_TO(schema.is_instance_of)),
    is_subset_ot      :  SET_OF(REF_TO)(schema.has_subsets)),
    has_subsets       :  SET_OF(REF_TO(schema.is_subset_of)),
    is_element_of     :  SET_OF(REF_TO(schema.has_elements)),
    has_elements      :  SET_OF(REF_TO(schema.is_element_of)),
    has_attributes    :  SET_OF(REF_TO(attribute.is_attribute_of)))
  KEYS_ARE(name);
CREATE ATOM_TYPE        attribute
  ( attribute_id      :  IDENTIFIER,
    name              :  CHAR_VAR,
    value             :  BYTE_VAR,
    type              :  (memberslot,ownslot)
    kind              :  (own,inherited)
    is_attribute_of   :  REF_TO(schema.has_attributes)
    has_aspects       :  REF_TO(aspects.is_aspect_of))
  KEYS_ARE(name);
CREATE ATOM_TYPE        aspects
  ( aspect_id         :  IDENTIFIER,
    name              :  CHAR_VAR,
    comment           :  CHAR_VAR,
    value_set         :  BYTE_VAR,
    cardinality_min   :  INTEGER,
    cardinality_max   :  INTEGER,
    default           :  BYTE_VAR,
    is_aspect_of      :  SET_OF(REF_TO(attribute.has_aspects)))
  KEYS_ARE(name);
```

b) atom_type definitions

Figure 12: Generic KOBRA schema expressed in terms of the MAD model

This MAD schema defines just one 'aspects' occurrence for each 'attribute' because aspect specifications have been grouped into the atom type 'aspects' and expressed as attributes thereof (note: at the data model level). Thus, aspect specifications are shared between several attributes preventing redundancy (especially in case of inherited attributes). Here, we explicitly exploit the capability of the MAD model to handle network structures in a direct and non-redundant manner. Such an adequate modeling is hard to achieve using data models that only allow for hierarchical structures (e.g. NF^2 models [19,20]). The organizational axes of KOBRA, i.e. attributes with abstraction semantics, are not represented as ordinary 'attributes' but as recursive MAD references between the atom type 'schema'. Since for each MAD reference, there always exists a back reference, each organizational axis is represented by a pair of MAD-attributes of type reference (shown as hatched double arrows in figure 12). In this illustration, we have also listed the atom-type definitions that lead to the MAD schema introduced above.

Transformation of KOBRA operations to MAD queries

On one hand, KOBRA operations can be viewed as general functions to work on the schema objects as a whole. On the other hand, there are also operations to directly manipulate the schema objects' attributes or their aspects. To exemplify our transformation approach, we just present two KOBRA operations together with the corresponding MAD statements.

The first operation, which expresses the function of reading a whole schema object, can be directly mapped to one MAD statement (figure 13). The data manipulation language defined for the MAD model is syntactically related to the well-known and easy to understand SQL language developed for relational database systems. Therefore, it is easy to capture the semantics of MAD statements: projection (SELECT) and qualification (WHERE) clause are quite similar to SQL, whereas the FROM-clause of MAD provides the important capability to dynamically express molecules just by writing the corresponding molecule structure or molecule type; thus, a dynamic complex object facility has been introduced.

Example: Selection of the schema 'bedrooms'

```
SELECT     ALL
FROM       schema-attribute-aspects
WHERE      schema.name = 'bedrooms'
```

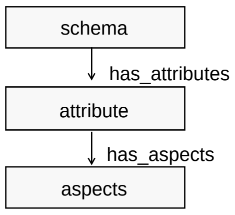


Figure 13: Selection of a schema object expressed in terms of one MAD statement

The second operation (figure 14), i.e. insertion of a new instance-attribute, is much more complex; for reasons of clarity, it has been carried out in several steps (and for sake of simplicity, we omit the insertion of the attribute's aspects). Since the insertion of an instance-slot (in the example, number_of_windows) provokes its inheritance, it is necessary to obtain the whole generalization hierarchy before inserting the attribute. This is accomplished by the first MAD statement of figure 14, which retrieves the whole generalization hierarchy beneath the schema object 'rooms'. Such a hierarchy is specified as a recursive molecule type starting with the root atom type 'schema' moving on firstly to all subclasses (using the reference type 'has_subclasses' in a recursive manner) and then to their instances (exploiting the reference type 'has_instances'). Additionally, for all schema objects, their attributes are also retrieved (taking the reference type has_attributes). The molecule diagram of this generalization hierarchy is also depicted in figure 14 visualizing the evaluation process of the corresponding molecules as introduced before.

Example: Insertion of the instance-slot 'number_of_windows' to the schema named 'rooms'

1. Retrieval of the whole subclass hierarchy with corresponding instance schemas

```
SELECT  ALL
FROM    generalization-hierarchy
        (schema-(.has_attributes-attributes,
                 .has_instances-schema.has_attributes-attributes)
        (RECURSIVE: schema.has_subclasses-schema))
WHERE   generalization-hierarchy.schema(root).name='rooms'
```

2. Modification within the KOBRA layer:
   - addition of the instance-slot in the root schema, i.e. the schema named 'rooms'
   - inheritance as instance-slot to all subordinate classes
   - inheritance as own-slot to all subordinate instance schemas

3. Update of the database:

```
MODIFY    generalization_hierarchy
FROM      generalization_hierarchy
```

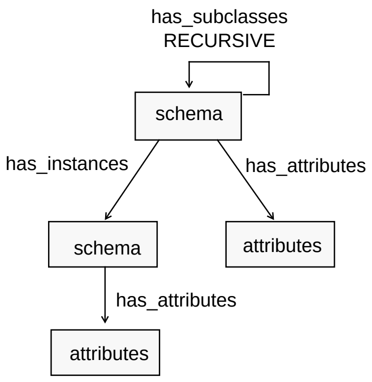


Figure 14: Insertion of an instance-slot expressed in terms of the MAD language

After this, the hierarchy is modified in the KOBRA layer to process the insertion and inheritance of the instance-slot, as exemplified in the second step. Following on, the complete schema hierarchy is then updated in the DB via a modify statement. This operation changes the specified molecule in accordance to the given hierarchy by inserting atoms not yet stored (here, all inherited attributes) and modifying the corresponding connections (here, the references between the 'schema' objects and their new attributes). In this example, it should have become clear that the recursiveness of the MAD model and its language could be extensively exploited to yield a direct and natural transformation of the KOBRA operations. Thus, recursiveness should be an integral part of the object-supporting interface.

Processing model and efficiency considerations

Before describing the dynamic behavior of the KRISYS system, it is necessary to get more detailed information concerning the implementation layer. Internally, this layer is divided into two subsystems (cf. figure 15): the PRIMA kernel and the working-memory system (WMS).

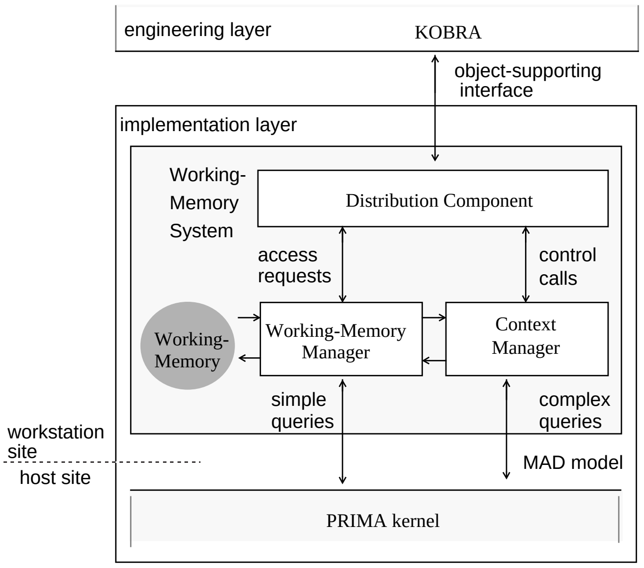


Figure 15: Components of the implementation layer

The PRIMA kernel offers application-independent data management functions at its interface which is determined by the MAD model. From the kernel's point of view, the task of the WMS is the embedding of the data model into an environment that could be easily and efficiently accessed by the engineering layer. In the PRIMA approach, a number of system design decisions were taken with special emphasis on efficient molecule processing which is enhanced by a variety of storage structures and other tuning mechanisms (e.g. molecule materialization or molecule caching). All these performance enhancements are transparent at the kernel interface. An in-depth description of the design and implementation concepts of PRIMA can be found in [1].

From the point of view of the higher levels, there is another very important issue when working with large KB: to efficiently cope with long execution paths of KB accesses, and time consuming requests to secondary storage. Thus, the task of the WMS is to firstly considerably reduce the path length and secondly to minimize the number of kernel calls when accessing KB objects. For this reason, WMS realizes a kind of application buffer, which temporarily stores needed objects in a main-memory structure, called working memory, that offers almost direct access at costs comparable to a pointer-like access. Consequently, WMS supports a processing model aimed at a high locality of object references, thereby drastically reducing the path length when accessing the KB. To reach this end, WMS offers the concept of contexts being a collection of objects which are needed during a specific processing phase. This concept is realized by the context manager, which fetches and discards such contexts as notified by specific control calls. These calls are then transformed into set-oriented kernel operations (complex queries) to extract the specified objects from the DB or to discard them from the working memory. Access requests to schema objects or parts thereof are dealt with by the working-memory manager, which generates and sends simple queries to the kernel if the requested objects are not found in the working memory. The distribution component is only responsible for proper routing of the respective KOBRA calls.

As indicated in figure 15, the KRISYS architecture fits nicely into the most realistic and prevailing architectural environment for engineering/design applications, i.e. workstation environments. Here, workstation-oriented processing could be effectively enhanced by delegating WMS, engineering, and application layer together with the application to the workstation and the PRIMA kernel to the host system. This partitioning is further favored by the set-orientation of the kernel interface and the locality preservation of the WMS, both minimizing workstation-host communication. Additionally, the loose coupling greatly facilitates failure isolation. This is a very critical design objective since a large number of users may be affected by any kind of failure, and because interactive design transactions are typically very long.

4.3 Summing up

After reading the previous sections, one should conclude that KBMS, and especially KRISYS, incorporate techniques of both AI and DB. The engineering layer embodies expressive, accurate, and flexible knowledge representation and manipulation mechanisms of AI, which are necessary to enable a precise and semantically rich modeling of the application world. The implementation layer supports DB mechanisms for a reliable and efficient management of persistent data, which is essential for a consistent and practicable object processing. Further, the application layer realizes the system interface (its description, due to space limitations, was left out of this paper; for details see [16]). Since KRISYS did not result from an extension or a coupling of existing systems, but from an integration of different technologies, it combines the advantages of both fields, which is fundamental for an effective support of advanced CAD. AI systems might allow for intelligent system behavior, but they lack efficiency, reliability, multi-user support, etc. DBS do fulfil management tasks but are, however, unable to provide the necessary modeling mechanisms.

In KRISYS, the integration of both technologies has been achieved in a natural way, reflected in the stepwise abstraction process realized at each layer. The implementation layer of KRISYS treats knowledge structures simply as a kind of network of complex objects which are consistently, reliably, and efficiently managed. At the engineering layer, these structures obtain semantics, being then applied for knowledge modeling tasks. Thus, the application-independent structural object representation provided by the object-supporting interface guarantees that the semantics of knowledge structures is known only by the KOBRA model, remaining, for this reason, outside of the NDBS component. So, the NDBS is not overloaded by application specific aspects, being, therefore, able to concentrate on performance and on other relevant DB aspects. On the other hand, the KOBRA model is not bounded by the semantics provided by the MAD model, being, as a consequence, able to come nearer to the application semantics by offering a rich and powerful spectrum of concepts for modeling.

5. Conclusions and Outlook

In advanced CAD systems, the user, i.e. the designer, is not the only active unit within the overall design process (as is the case in conventional CAD systems. In such environments, the CAD system also exhibits an active behavior, therefore providing a more intelligent interface to the user: guaranteeing a consistent design and offering appropriate design hints, relevant problem solutions, refined simulation results, and adequate diagnostic information at all stages of the design process. In some sense, there is a kind of 'partnership' between designer and system, where it is possible to switch between automated design guided by the system (e.g. in standard cases) and human design controlled by the user's decisions (e.g. in special cases). The main goal of advanced CAD is to improve the design process by incorporating not only geometrical but also technical and functional aspects as well as construction and manufacturing dependencies along each design step. This leads to a penetration of the design phases and to a manufacture-oriented design methodology. Recently, for example, the technical term feature modeling has arisen to express the shift from simple geometric modeling (where it is only dealt with geometry) to a modeling concept that additionally considers production and manufacturing aspects of the design object [21].

Due to KRISYS's rich spectrum of modeling concepts, the process of modeling and, therefore, also the design process turn out to be

- more consistent - since a substantial amount of the application semantics has been incorporated into the model,
- more flexible - since it is possible to repeat previous modeling steps in order to make corrections or extensions on the knowledge base, and
- easier - since KRISYS is also an active agent in the modeling process, continuously making deductions on object structures, checking the semantic integrity, and inferring new information relevant to the application.

On the other hand, the architecture and implementation design decisions made (e.g. context-driven working-memory system, NDBS usage) are a prerequisite for persistent, reliable, and efficient management of the KB on secondary storage, additionally enhancing the prevailing workstation-oriented processing. Thus, we can conclude that this integration of DB and AI techniques fortify each other, aiming at a base system such as KRISYS that effectively supports advanced CAD.

Though our approach looks like having removed some main obstacles on the way to better CAD, a lot of problems have not been taken into our consideration. There is a need for a tailored version concept (including version graphs, design alternatives, and different configurations) structuring the overall design process. These structuring capabilities are embedded into a design-transaction concept to guarantee durability, consistency, and isolation (or sometimes just the contrary, i.e. group cooperation) of each design phase with respect to the whole design process. Here it should be mentioned that there are already some contributions to this problem area: e.g. in AI the concept of viewpoints [10] is known, and in DB, for example, long transactions [22, 23], synchronization, recovery, and failure masking in workstation-oriented engineering environments [24]; note that these concepts are applicable to our KBMS approach because of the separation of DBS kernel and WMS in the implementation layer's architecture. Although other extensions are conceivable, it is important to gain practical experience with KRISYS in some in-the-field applications. Therefore, we have worked out applications in the areas of diagnoses, planning, and design. The results of these quite different and varied applications will offer more insight into modeling and efficiency requirements of advanced CAD systems.

Acknowledgment

We would like to thank Heike Neu for the preparation of the manuscript.

References

1 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.

2 Mattos, N.: KRISYS - A Multi-layered Prototype KBMS Supporting Knowledge Independence, in: Proc. Int. Computer Science Conf. - Artificial Intelligence: Theorie and Applications, Hong Kong, Dec. 1988, pp. 31-38.

3 Hartzband, D., Maryanski, F.: "Enhancing Knowledge Representation in Engineering Databases, in: Computer, Vol. 18, No. 9, Sept. 1985, pp.39-48.

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

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

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

7 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.

8 Clayton, B.D.: Inference ART Programming Tutorial, Volume 1 Elementary ART Programming, Volume 2 A First Look At Viewpoints, Volume 3 Advanced Topics in ART, Los Angeles, CA, 1985.

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

10 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.

11 Carnegie Group Inc.: Knowledge Craft CRL Technical Manual, Version 3.1, Carnegie Group, 1987.

12 Bobrow, D., Stefik, M.: The LOOPS Manual, Palo Alto, Calif., Xerox, 1983.

13 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.

14 Peckham, J., Maryanski, F.: Sematic Data Models, in: ACM Computing Surveys, Vol. 20, No. 3, September 1988, pp. 153-189.

15 Dittrich, K.R.: Advances in Object-Oriented Database Systems (Proc. of the 2nd Int. Workshop on Object-Oriented Database Systems), Lecture Notes in Computer Science 334, Springer-Verlag, Berlin, 1988.

16 Mattos, N., Deßloch, S.: KOALA - An Interface to Knowledge Base Management Systems, internal report, University of Kaiserslautern, 1989, submitted for publication.

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

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

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

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

21 Shah, J.J., Rogers, M.T.: Expert form feature modelling shell, in: Computer-Aided Design, Vol. 20, No. 9, November 1988, pp. 515-524.

22 Kim, W., Lorie, R., McNabb, D., Plouffe, W.: Nested Transactions for Engineering Design Databases, in: Proc. 10th VLDB Conf., Singapore, 1984, pp. 355-362.

23 Bancilhon, F., Kim, W., Korth, H.F.: A Model of CAD Transactions, in: Proc. 11th Int. Conf. on VLDB, Stockholm, Aug. 1985, pp. 25-33.

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