appeared in: Proc. BTW'91, Kaiserslautern, March 1991, IFB 270, Springer, pp. 89-108.

Handling Integrity in a KBMS Architecture for Workstation/Server Environments

Stefan Deßloch
University of Kaiserslautern
P.O. Box 3049, 6750 Kaiserslautern
Federal Republic of Germany
e-mail: dessloch@informatik.uni-kl.de

Abstract

This paper presents the concepts and mechanisms for maintaining semantic integrity of complex applications, which are provided by the knowledge base management system KRISYS. We discuss the integration of integrity maintenance into the global architecture of the system which is based on a workstation/server environment. As a result of our elaborations, we argue that semantic integrity should be handled in the knowledge model (at the workstation), leaving only basic consistency tasks for the data model (at the server).

1. Introduction

The efficient and effective support of applications with increasing complexity, such as engineering design, expert, and knowledge-based systems or office automation, has been a major concern in database research over the last few years. In this context, the provision of semantically enriched data or knowledge models for coping with the enhanced complexity of the application domains is an important issue. Besides such data or knowledge modeling requirements, the design of an appropriate system architecture supporting these applications has to take into consideration typical processing characteristics and hardware environments. In general, these applications run on workstations supplied with sufficient processing 'power', main memory, and private disk space, which are dedicated to single users and connected to a central server component, whose task is to provide and control access of centralized information. Among others, failure isolation, extensibility, and scalability of the whole complex can be considered as key advantages of such system architectures.

With these goals in mind, a prototype of a knowledge base management system (KBMS), called KRISYS, was developed at the University of Kaiserslautern [Ma89]. KRISYS provides the application with a powerful knowledge model, called KOBRA, offering flexible constructs for describing the application domain, such as object-oriented concepts for integrating behavior into the application model, abstraction concepts (generalization, classification, aggregation, and association) for representing organizational structures, general reasoning facilities for performing deductions, and several constructs for maintaining the semantic integrity of the knowledge base (KB). In order to fit into the above described processing environment, KRISYS is architecturally divided into two main parts (see Figure 1.1(a)). A database management system (DBMS) located at the server concentrates on efficient and reliable data management, whereas the second part residing on the workstation side supports the "nearby application locality" concept in order to efficiently implement the KOBRA model and its query language.

The architecture of KRISYS can be compared to the so-called DBMS-kernel architecture for non-standard database systems [HR85] depicted in Figure 1.1(b). This approach enables the support of different application classes by distinct application-specific layers residing on the workstations together with the application programs. The kernel, which is designed to run on the central server, is defined to be application independent, offering neutral, yet powerful data manipulation services (e.g. storage techniques, flexible representation and access techniques, etc.) that may be utilized by the application-specific layers. In fact, the current implementation of KRISYS utilizes as DBMS the PRIMA system [HMMS87], which is a DBMS-kernel prototype developed at our university. For this reason, from an abstract point of view, KRISYS may be seen as a specific implementation of this architecture, where the application-specific layer corresponds to the implementation of KOBRA (in reality, KOBRA is a general purpose knowledge model and not restricted to an application class).

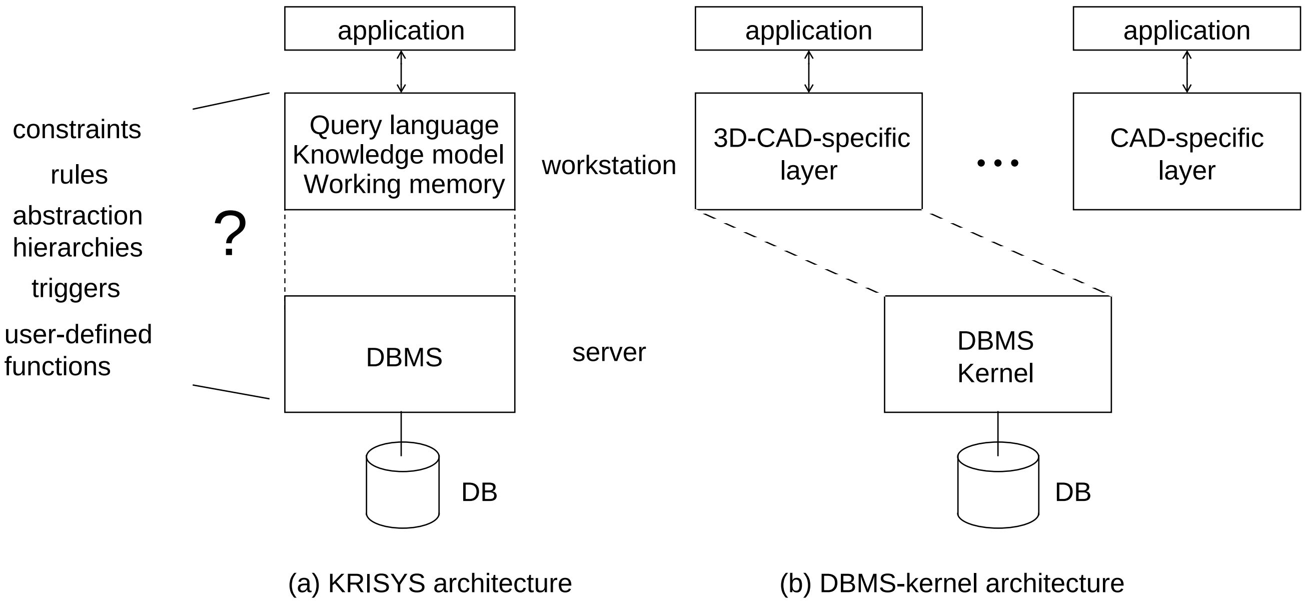


Figure 1.1: Architectural approaches considering a workstation/server environment

When considering the underlying workstation/server environment, the coupling between the workstation component and the DBMS becomes a performance-critical issue. Among other considerations, minimizing communication traffic between server and workstations emerges as a primary goal within the overall design of the system. One step towards this goal is taken by integrating an application buffer (called working-memory in KRISYS) into the workstation component in order to exploit locality of references [HHMM88]. In an engineering design system, for example, operations are usually addressed to one or several specific design objects under consideration. This fact can be exploited to retrieve all information relevant to the desired object from the DBMS and fix it in the application buffer. Manipulations of the object can then be accumulated in the buffer, being propagated to the DBMS only at the end of a design phase, when the object is discarded from the working memory and stored back into the DB.

The concept of an application buffer may be considered as important, but not as sufficient for the minimization of the workstation/server communication. An appropriate DBMS interface plays a crucial role in this context as well, since the functionality provided at the server side (e.g. trigger mechanisms, execution of user-defined operations, etc.) influences the amount of processing that is performed at both server and workstation. If the functionality of the server is enhanced, a lot of processing is shifted from the workstation to the server, because the operations performed by the server become more complex. On the other hand, less server functionality tends to leave more processing at the workstation and suggests the usage of the data model interface for data transfer purposes only. Therefore, it is important to decide very carefully by which system component each of the provided modeling constructs (e.g. abstraction concepts, integrity constraints, etc.) should be supported.

In this paper, we will concentrate on the constructs of KRISYS for semantic integrity maintenance in order to discuss the questions and problems arising in this context. We will show that integrity control should be almost completely dedicated to the workstation. As a consequence, the corresponding services to be provided by the server are only of a basical nature and restricted to mechanisms without any kind of application semantics. Since the arguments used in our discussion do not rely on specific modeling constructs of KRISYS, they can be easily generalized to conclude that most of the current research efforts to increase the modeling power of data models at the server side to improve application support are useless in system architectures for workstation/server environments.

In order to provide an understanding of the problems sketched above, we will first give an overview of the mechanisms provided by KRISYS for integrity enforcement. Subsequently, the consequences of a possible delegation of integrity maintenance to the server will be elaborated. Finally, we will conclude with a presentation of results achieved throughout our discussion.

2. Semantic Integrity Enforcement in KRISYS

After a short description of the basic modeling concepts provided by KRISYS, we give an overview of the mechanisms for maintaining the semantic integrity of an application. We will, however, concentrate on the principal issues (a more detailed description can be found in [De90]). The examples for illustrating the modeling constructs presented are taken from the area of architectural design.

2.1 Basic Concepts of the Knowledge Model

KOBRA, the knowledge model of KRISYS, embodies an **object-centered representation** of the application world. That is, every entity of the application is represented by a **schema** which may have attributes to describe its characteristics and is clearly identified by its name. A schema corresponds to a frame or unit in other knowledge representation systems and must not be confused with a DB-schema. Attributes are either **slots**, which can have multiple values representing properties of an object or relationships to other objects, or **methods** describing its behavior. In order to characterize an object in more detail, attributes can be further described by **aspects**. The schema 'room1' in figure 2.1 , for example, represents a certain room of a house, described by the slots orientation, position, sides, size, and neighbors, together with the corresponding values. The 'size'-attribute

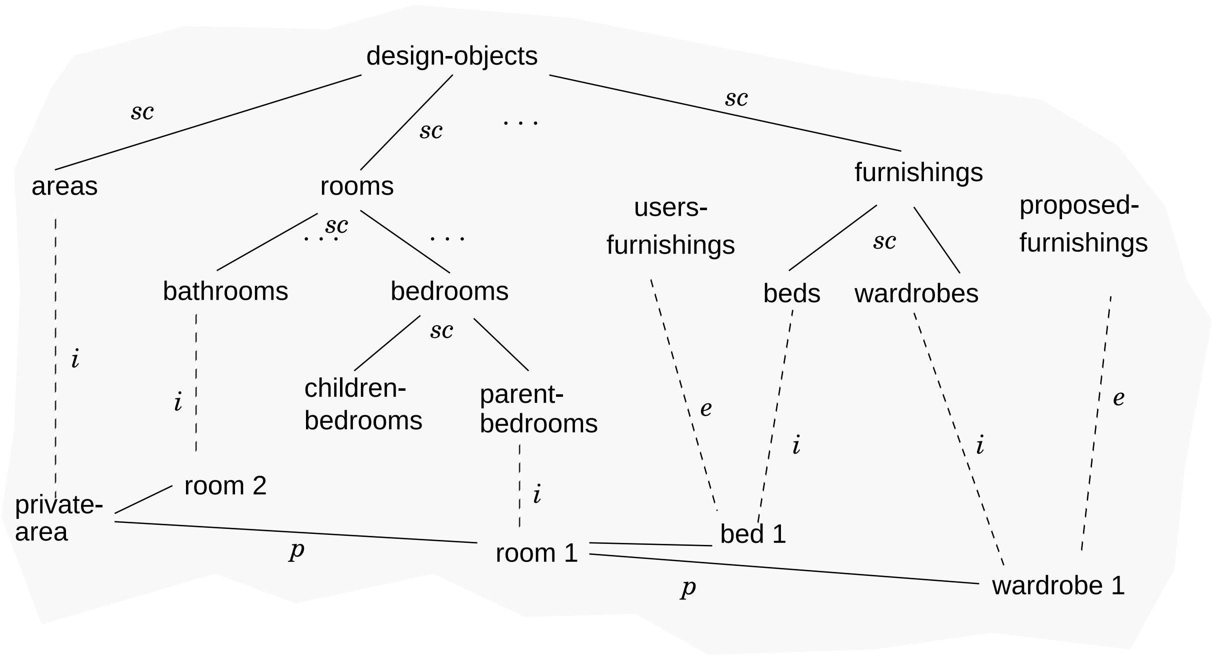
room 1		
orientation		-
position		-
sides		4, 4
size		16
	unit	square-meters
neighbors		room 2, room 4
add-neighbors		"procedure-code"

Figure 2.1: Sample schema description

is further described by the 'unit'-aspect, which fixes 'square meters' as unit for the size of the room. The schema 'room 1' is also characterized by the method 'add-neighbors' used for establishing a neighborhood relationship between 'room 1' and another room, which may, for example, involve certain recalculations of the position or orientation of a room.

For structuring the KB, KOBRA allows to employ the **abstraction concepts** of classification, generalization, association, and aggregation [Ma88]. These concepts are seen as special, predefined relationships between objects, specifying the overall organization of a KB as a kind of complex network of objects. KRISYS supports an **integrated view** of KB objects: there are no separate representations for classes, sets, instances, complex objects, etc. The same schema can, for example, represent a class with respect to one object, and a set or even an instance with respect to another. As a consequence, the difference between data and meta-data, which is usually apparent in existing data models, is eliminated in KRISYS, so that **meta-information is integrated into the KB**.

A partial view of our example KB is given in figure 2.2. The schema 'room 1', for example, is defined as an instance of parent-bedrooms (i.e. classification), which is a subclass of bedrooms (i.e. generalization), which in turn is again a subclass of rooms, etc. The concept of aggregation is required for expressing that 'room 1' is part of the 'private area', and contains furnishings like 'bed 1' and 'wardrobe 1'. During the design of the house, the system exhibits an active behavior, offering possible solutions for design problems or proposing additional furnishings for the rooms of the house. In order to distinguish furnishings already owned by the user from those which are proposed by the architectural system, we use the concept of association, which allows possibly heterogeneous objects to be grouped together, and introduce two object sets ('users-furnishings' and 'proposed-furnishings'). Because an integrated view of the abstraction concepts is supported by KRISYS, the schema 'bed 1', for example, plays three different roles at the same time: it is an instance of 'beds', an element of 'users-furnishings', and a component of 'room 1'.



sc: subclass-of (generalization)
i : instance-of (classification)
e: element-of (association)
p: part-of (aggregation)

Figure 2.2: A partial view of the example KB

2.2 Integrity Constraints Supporting the Abstraction Concepts

The abstraction concepts introduced above can be regarded as the basic constructs provided by KRISYS for modeling an application domain. For this reason, it is necessary that the semantics of these concepts are fully guaranteed by the system in order to keep the KB in a consistent state. For example, based on the notion of transitivity inherent in the abstraction concepts, the introduction of cycles (e.g. by establishing 'room 1' or 'parent-bedrooms' as a superclass of 'design-objects') or other ambiguities (e.g. by making 'room 1' an instance and a subclass of 'parent-bedrooms' at the same time) clearly contradicts the meaning associated with the abstraction concepts and, therefore, has to be prevented by the system. Of course, the association and aggregation concepts have to be controlled in a similar way.

In KOBRA, each abstraction concept is represented by a pair of system defined, symmetric slots (e.g., ’subclass-of’ and ’has-subclasses’ for the generalization concept). Upon modifications of abstraction hierarchies, the symmetry of the relationships is automatically maintained by the system, i.e., the corresponding inverse slots are automatically updated.

Classification / Generalization

According to the generalization/classification relationships, inheritance needs to be supported. However, in order to keep the KB consistent, inheritance has to be viewed not only as a means for passing on information in order to save typing efforts, but as a constraint on the structure of objects: subclasses and instances have to contain all attributes of their superclasses. For this reason, an attempt to delete an inherited attribute (e.g., 'neighbors' in 'bedrooms') has to be rejected by the system. Modifications of classes (e.g., deleting the slot 'neighbors' in 'rooms' where it was defined, or introducing new attributes) have to be dynamically propagated to existing subclasses and instances.

Association

In order to support the semantics of the association concepts, KRISYS allows the specification of so-called membership stipulations and set properties [Ma89]. Membership stipulations represent necessary conditions that have to be fulfilled by objects in order to become elements of a set (e.g., all elements of the set ’user-furnishings’ have to have the user’s name as value of the attribute ’owner’). Objects that do not fulfil the membership stipulations are automatically rejected as elements of the corresponding set. Set properties characterize the set itself and can be derived or computed from properties of its elements. For example, the value of the attribute ’total-price’ of the set ’proposed-furnishings’ corresponds to the sum of the prices of all of its elements. When elements of the set are deleted, added, or changed, the value of the set property will then be automatically updated by the system.

Aggregation

Analyzing the aggregation relationships in our example, we notice that some properties of the objects involved can be characterized as monotonic. For example, the size of a room is always smaller than the size of the area, in which it is contained. The size of a piece of furniture is again smaller than the size of the room where it is placed. This characteristic can be described by means of implied predicates in KOBRA. We may denote the property 'size' as monotonically decreasing, with regard to the aggregation hierarchy which enforces the system to prevent the violation of the monotony constraint. For example, changing the attribute 'size' of a room will only be permitted if the new size is smaller than the area size and greater than the size of the rooms furniture.

It is the support of these constraints as part of the knowledge model of KRISYS that guarantees the consistent and complete support of all four abstraction concepts. With respect to this issue, KRISYS is clearly superior to existing knowledge representation languages or expert system tools (e.g., [FK85, FWA85, In87, SB86]), since even the basic constraints associated with these concepts are usually ignored by such systems (see [De90] for details).

2.3 Integrity of Attribute Values

Basic requirements of applications concerning semantic integrity are usually associated with the values of attributes. In KRISYS, the domain and the cardinality of attributes may be specified by means of two predefined aspects ('possible-values' and 'cardinality'). If the operations of the user violate these constraints, they are rejected. Using the possible-values aspect, a constraint similar to the referential integrity in relational DBMS can also be enforced by restricting the values of an attribute to identifiers of objects belonging to certain classes. For example, in order to allow only rooms as neighbors of a room, we simply need to specify "instance-of rooms" as the domain description of the 'neighbor'-slot in 'rooms'. In this case, the system ensures, that only identifiers of instances of rooms can be inserted as slot values. Similarly, the deletion of a room from the KB is only allowed, if the object is not referenced by any attributes of other objects. Additionally, it is possible to construct more complex domain specifications from simple ones by means of logical operators (and, or, not). For example, to ensure that the neighbors of parents-bedrooms have to be either bathrooms or children-bedrooms, we specify "instance-of bathrooms or instance-of children-bedroom" as a value of the possible-values aspect for the neighbors-slot in parents-bedrooms.

Of course, it is required that the attribute domain and cardinality constraints strictly follow the semantics of the generalization concept. Therefore, the attribute domain and cardinality restrictions of a class can only be restricted by its subclass, i.e., they are never allowed to be more general than those in the superclass.

2.4 Demons

The constructs introduced above are sufficient for the description of the basic integrity constraints of an application domain. However, they cannot handle complex constraints that involve computations or represent dependencies among several attributes or even objects. In our architectural application, it is, for example, necessary to ensure that the size of a room is always represented correctly with respect to the lengths of its sides. In KRISYS, this can be accomplished by means of a so-called demon, i.e. a procedure attached to the relevant attributes (see figure 2.3). The complexity of dependencies involved in such kinds of constraints usually requires a high degree of flexibility and power for an appropriate integrity mechanism:

- Several attributes possibly associated with more than one object are involved (e.g., changing one side affects the size of a room and consequently influences the positions of neighbor-rooms).
- The power of a high-level programming language is required in order to carry out computations.
- Flexible reactions to the violation of constraints should be possible (e.g., changing the length of one side of the room could either result in rejecting the operation or in recalculating the area).
- An extensional as well as an intensional representation of the constraint should be possible (i.e., the dependency of the area and the sides of the room is assured by either recalculating the area when changes on the ’sides’-attribute are made, or by determining the area only when the value is required).

- Transitional constraints characterizing not only consistent states but also correct state transitions need to be supported.

The demon mechanism in KRISYS offers all of the above required power and flexibility and permits a detailed and flexible specification of activation conditions. Demons are attached to the attributes of objects, in order to react on all kinds of events in which the attributes are involved.

The activation events that can be defined for a demon (put, add, retract, get, send) correspond to the basic types of attribute access that are supported by generic KRISYS operations. For example, the demon in figure 2.3, which is attached to the slot 'sides', would contain 'put' as a proper activation condition, meaning that the demon is to be activated immediately upon an update operation on the slot. Each event specified for a demon can be related to a different action in order to provide individual reactions of the demon according to the events. Such actions may check complex constraints that might be invalidated by the event and take appropriate steps, if they are no longer satisfied. The scope of the actions is not limited to the attributes of objects involved in the attachment of the demon, but may involve arbitrary parts of the KB. Additionally, the time of activation (before or after the event is actually performed) may also be specified.

In KRISYS, demons are simply represented as objects of the KB containing certain methods that embody their reactions on events. A demon is therefore simply defined by creating a new object as an instance of a predefined class 'demon' and filling in the code of the methods inherited from the demon class (e.g. put-before, put-after, etc.) in order to specify actions to be invoked before or after the occurrence of certain events. The demon may then be attached to attributes of other objects using the (predefined) 'demon'-aspect of the attributes. This is accomplished by specifying the name of the demon as the aspect value (see figure 2.3). Therefore, a demon may easily be attached to several attributes (of probably distinct objects or object types) using the mechanism described above.

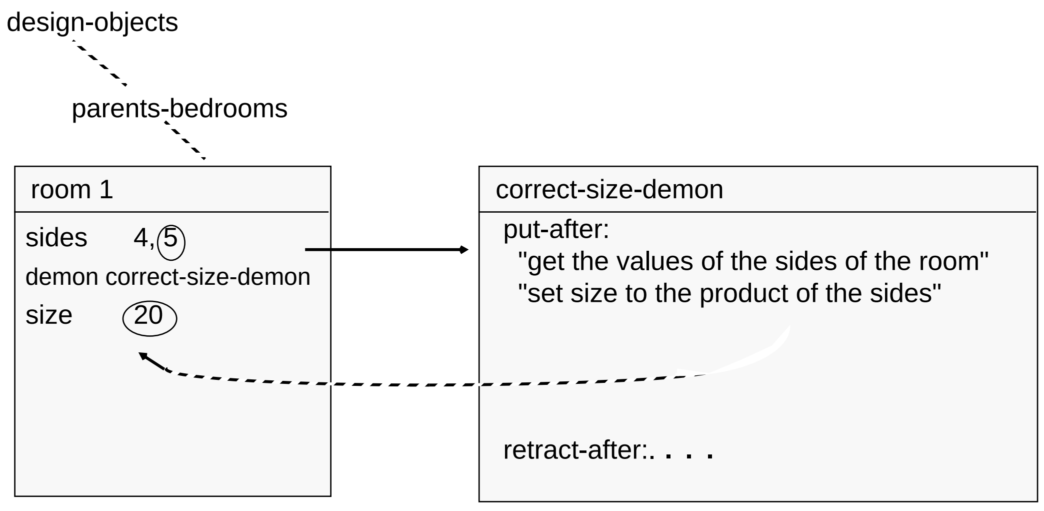


Figure 2.3: Using demons for preserving integrity

The demon mechanism of KRISYS described above certainly shares its basic intention and some of its properties with trigger facilities that can be found in existing database systems [Ko89,SYB88]. Nevertheless, there are some important differences:

- The actions supported by triggers are usually limited to a sequence of DML statements, whereas demons in KRISYS may also utilize the computational power of a high-level programming language (Common Lisp) for the specification of conditions as well as actions.

- Upon the activation of a demon, the system automatically supplies additional environmental information of the event (e.g. names of the involved attribute and object, attribute values, method parameter values, etc.), that may be utilized by the operations for accessing all objects and attributes that are actually involved in the integrity constraint to be checked or guaranteed by the demon. This information supplied to the demon also contains the old as well as the new values of modified attributes in order to permit the maintenance of transitional integrity constraints by the actions of the demon.
- Upon the detection of inconsistencies, several kinds of reactions are possible: the appropriate action of the demon may for example perform certain operations to reestablish the constraint (as, for example, in figure 2.3), notify or interact with the user, or it may cancel the operation that caused the event in order to keep the KB in a consistent state.
- Besides the above described extensional maintenance of integrity, which either rejects operations leading to inconsistencies or takes appropriate actions to reassure consistency, the demon mechanism alternatively allows an intensional maintenance of integrity, since actions may also be triggered by a read access to attributes. In this case, the demon will automatically provide the correct attribute value using the integrity constraint for its computation.
- The alternative ways of maintaining integrity and the various types of reactions that may be utilized by the demon are supported by further options of the operations specified as a reaction for an event (see figure 2.3): operations that are carried out before the actual event takes place (e.g., to compute a value upon a read event or to prevent an inconsistent update) are distinguished from those that are executed after the event (e.g., reestablishing a consistent state, as described in ons example).
- From a conceptual point of view, there are also differences between triggers and demons: Triggers are only capable of reacting to exactly one specified event, and, therefore, several triggers are usually defined in order to maintain one integrity constraint (e.g., there are distinct triggers for insert and another for update of a data element). In contrast to this approach, a demon unites all events involving an attribute and all reactions on the events into one conceptual object that guards all operations performed on the attribute.

2.5 Regarding methods as units of integrity

Similar to a transaction in DBMS, a method is viewed in KRISYS as a unit of integrity. This notion is supported by a mechanism that allows the effects of a method to be undone - e.g. as a reaction to a constraint violation. Since methods may activate other methods and, as a consequence, activations of methods may be arbitrarily nested, different levels of consistency can be defined (see Figure 2.4). Similar to nested transactions, undoing the effects of a method also removes the effects of all the methods that were called by it. For example, resetting method m2 in Figure 2.4 as a reaction to an integrity violation also removes the effects of m4. In many cases, it is certainly not desirable, as the effects of a method are undone upon a consistency violation, to recursively reset all the calling methods as well. For this reason, a send operation within a method can be issued with the request to handle integrity violations that might occur in the called methods. Then, after resetting the effects of the called method upon detecting inconsistencies, the system returns the control to the calling method which may handle the situation appropriately and proceed with its work. For example, method m1 in Figure 2.4 has issued the request to handle integrity violations of both m2 and m3. Therefore, control is returned to m1 upon a violation occuring in the scope of these methods after their effects are undone. This request has not been issued by the method m3. Since an integrity violation detected at the end of m6 cannot be handled by m3, control is automatically returned to m1 after undoing the effects so far provoked by m3, m5 and m6.

The notion of methods as units of integrity and the consideration of different levels of integrity is additionally supported by the possibility to attach demons to methods. A demon attached to a method may contain different operations to be carried out either before or after the execution of the method, supporting the specification of some kind of 'preconditions' or 'postconditions' for the method. Upon the detection of inconsistencies, the demon may, among other possible reactions, initiate an abort operation with respect to the method to which it is attached. For example, the integrity violations sketched in Figure 2.4 could all be detected by demons attached to the methods, which are activated at the end of the method execution.

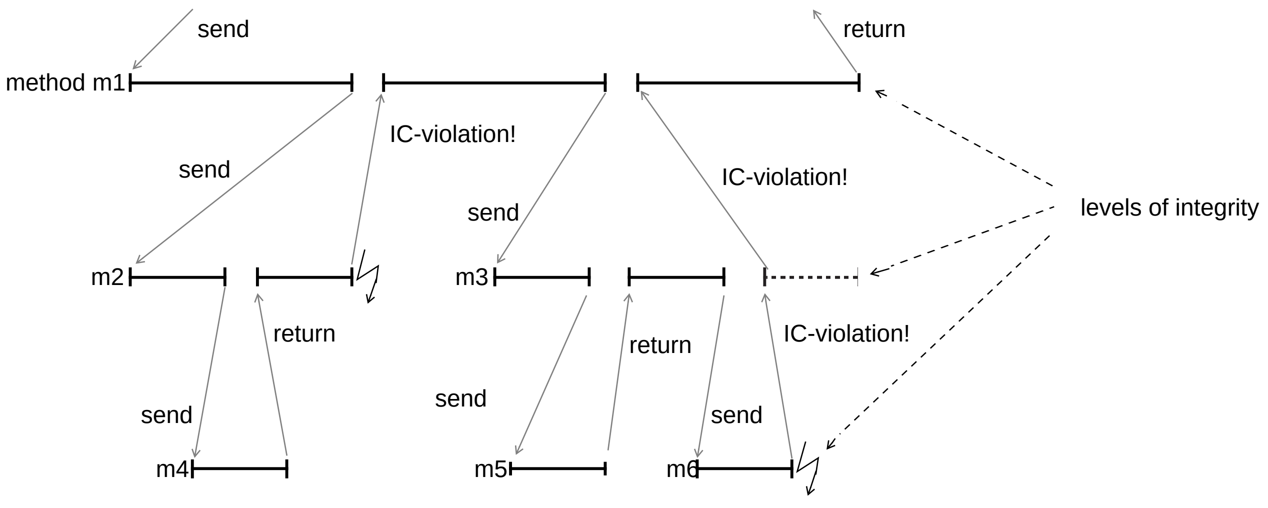


Figure 2.4: Methods as units of integrity

Although the attachment of demons to methods may seem rather questionable at the first sight, since all actions performed by the demon could also be integrated into the method as additional operations, there are several advantages that motivate this approach:

- A demon can be easily attached to several distinct methods, that require the same integrity checks.
- Integrity constraints are not hidden within the code of the method but explicitly separated from the methods by the definition of an appropriate demon. This is a prerequisite for the support of schema extensibility or schema evolution.
- Especially when constraints involving several attributes of different objects are considered, the notion of object modularity is supported. Methods may concentrate on the realization of object behavior without the necessity to consider additional operations involving other objects in order to support consistency.

Note that the attachment of demons to methods and, above all, the notion of a method as a unit of integrity is completely unknown in knowledge representation languages or expert system tools. As a consequence, integrity violations that cannot be immediately detected automatically lead to inconsistent KBs. In these systems, inconsistencies have to be either tolerated or compensated by the user.

2.6 Rules

Although demons can be regarded as a flexible and powerful mechanism for checking or improving consistency, there are a lot of cases where a more declarative representation of integrity constraints is required. Consider for example, that we want to allow changes which will affect the area of a room even if the position of its neighbors are already fixed. We then also need to automatically change the positions of these rooms thereby reorganizing the whole house - a process which may involve architectural expertise. This kind of information can be represented more appropriately by means of rules.

Rules, as means for supporting general reasoning facilities, can be employed in two different ways: Their activation may lead either to the derivation of intensional information (e.g., via a backward chaining process) or modify the state of the KB using a forward chaining mechanism. Both alternatives may be exploited for preserving the consistency of the KB. For example, the constraints for placing the rooms in our house, which we have represented as rules, can be assured by initiating a forward chaining process each time a side of one of the rooms is modified. This process will then change the positions of the rooms (i.e., modify the KB) during the rearrangement process. Alternatively, we could decide to maintain the constraints by representing the positions of the rooms as intensional properties, i.e., we will derive the positions by using our set of rules in a backward chaining process each time the positions of the rooms are needed.

When comparing demons and rules in KRISYS, certain commonalities appear: both mechanisms may be used for intensional and extensional integrity enforcement, and the scope of the constraints describable by them may transitively reach arbitrary attributes, objects, and abstraction relationships within the KB. A comparison, on the other hand, reveals several important differences concerning the method of activation and the language for the description of constraints: conditions and reactions for demons are formulated using a programming language allowing the integration of extensive computations into the consistency checks, whereas the definition of rules is based on a declarative, predicate-based language. Also, in contrast to demons, which are activated automatically on certain events, rules are activated explicitly. This is especially useful for certain technical applications, like CAD, where complete consistency of a design object is usually not permanently required, but tested explicitly by the engineer as the design reaches a certain stage. An additional difference is based on the fundamental difference between the procedural and rule-based programming paradigms [Ba88]. When specifying actions for demon in a procedural programming language, the control structures determining the exact sequence of operations to be carried out are explicitly fixed in the specification of the actions. For rules, on the contrary, control structures need not to be specified, since the overall control of the inference process is inherent in the inference and problem solving strategies, allowing the definition of rules to be performed in a more abstract manner.

2.7 The representation and organization of integrity constraints

The enormous complexity of non-standard applications is usually also revealed by the increasing number of dependencies or integrity constraints. A support not only of flexible and powerful integrity control mechanisms, but also of means for organizing and maintaining the constraints during the design or prototyping phase of the application is therefore strongly required. The organization of constraints remains important even after the application has been completely designed, since changes in the application world have to be reflected in the KB and may also affect integrity constraints.

Therefore, rules and demons are represented in KRISYS as regular objects of the KB, which leads to the following advantages. Firstly, new language constructs for the definition of rules and demons do not have to be introduced, since the regular KRISYS operations can be used, i.e., creating objects, connecting them as instances to classes, changing attribute values, etc. Secondly, the KB-designer may employ the modeling constructs of KRISYS to extend the description of rules and demons. For example, additional slots can be defined for demons as well as for rules or rule sets in order to give a better characterization of the constraints realized by them. In our architectural design system, for example, certain constraints leading to design restrictions are related to statutes that have to be obeyed when a house is built. The rules and demons can therefore be described by an additional slot that contains the number of the relevant statutes they are related to.

In order to organize demons and rules into class or set hierarchies according to their different tasks, the abstraction concepts may be used. For example, in our architectural KB, we can define a set 'determine-correct-positions' which contains all the rules involved in the placing of rooms in order to organize all constraints relevant for the correct position of rooms into one conceptual object. We may further include this set in an association hierarchy containing all constraints relevant for rooms (see figure 2.5). In order to distinguish constraints that are only defined between different attributes within one instance of 'rooms' from those that describe dependencies between different rooms, we introduce appropriate sets as subsets of 'room-constraints'. Note that the association hierarchy does not only contain rule sets. The demon 'correct-size-demon', for example, which has been introduced in figure 2.3, is regarded as an element of 'constraints-within-room'. The constraints related to rooms are, of course, only a small subset of all integrity constraints describing our architectural application. Consequently, the association hierarchy introduced above only forms a subhierarchy of the set 'architectural-constraints', that has all integrity constraints of our application as its elements.

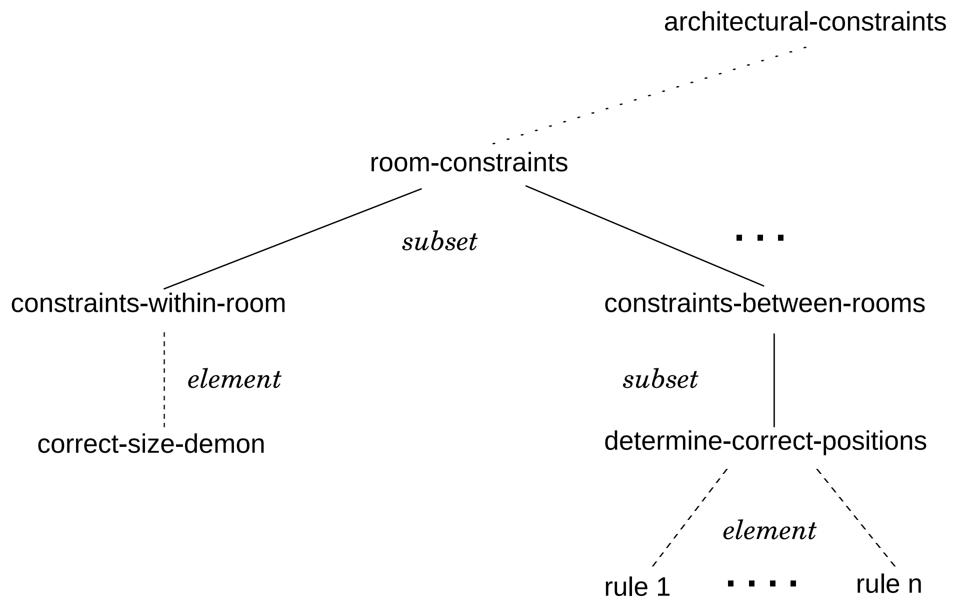


Figure 2.5: The organization of constraints

The representation of demons and rules as objects of the KB makes it possible to ask queries with respect to the set of constraints. Therefore, it is easy to locate all constraints that correspond to a certain statute with a single query.

The representation of constraints as objects and the applicability of the abstraction concepts offer a high degree of modularity, flexibility, and extensibility, which is a prerequisite for dealing with the evolution of an application and easily supports modifications of integrity constraints caused by changes in the application domain. For example, when a new statute passes legislation, it is straightforward to define new constraints and include them into the organization shown in figure 2.5. When statutes change, the corresponding constraints can simply be determined by a query and are easily deleted or modified.

3. The Impact of a Workstation/Server Environment on Semantic Integrity Maintenance

When considering the integrity maintenance of an application, the overall architecture of KRISYS (see Figure 1.1) gives rise to the question as to whether the constructs for integrity enforcement described above can be supported by or delegated to the DBMS component. Apparently, the underlying workstation/server environment plays an important role in the overall discussion of this question. In order to provide the necessary basis for the discussion, we will first give a more detailed description of the MAD model [Mi88], the data model provided by the DBMS, and show how information is mapped from the representation supported by KOBRA to the MAD representation. Subsequently, we will discuss a possible delegation of integrity maintenance to the DBMS from several points of view. At the end of the chapter, we will summarize the general results of our discussion and show implications for the services required from the DBMS.

3.1 The MAD Model and its Support of KOBRA

The DBMS used by KRISYS, called PRIMA, embodies the MAD (Molecule-Atom-Data) model, a structurally object-oriented data model offering dynamic definition and handling of complex objects. A detailed description of the MAD model and a general motivation of its design (which is certainly beyond the scope of this paper) can be found in [Mi88].

The basic modeling construct of the MAD model is the atom (or atom type), corresponding to a tuple (or relation) in the relational model. A direct and symmetric representation of different kinds of relationships (1:1, 1:n, n:m) between atoms is provided by means of links, which allow the DB to be viewed as a complex network of atoms. Structural object-orientation is achieved dynamically by the definition of so-called molecules, specifying a graph with certain atoms and relationships as a sub-graph of the database.

In order to realize the application-oriented view of a domain, which is represented in terms of the modeling constructs of KOBRA (chapter 2.1), it becomes necessary to map KOBRA objects into an appropriate MAD representation, which is sketched in Figure 3.1. A schema in KOBRA described by its attributes and aspects corresponds to a molecule of the three atom types 'schema', 'attribute' and 'aspect' in MAD. The abstraction concepts are represented as (recursive) relationships between schema-atoms. This mapping process is performed automatically at the workstation, whenever objects are stored into or discarded from the application buffer of KRISYS, so that the buffer representation already embodies the semantics of the KOBRA model.

In order to realize KOBRA operations, the query language of MAD, called MQL, can be directly employed. For example, the definition of a new attribute for a class, which requires the dynamic inheritance of the attribute in order to guarantee the model-inherent integrity of the generalization concepts, can be supported by an MQL statement that selects all relevant object classes and instances by recursively following the apropriate references (i.e., has-instances and has-subclasses) [HM90]. The selected part of the generalization hierarchy may then be modified within the application buffer in order to carry out the operation.

The generic mapping schema sketched above is very flexible and therefore especially appropriate for supporting the design process or prototyping phases of an application, where object definitions and abstraction hierarchies are frequently changed or extended. For application processing, where this flexibility is usually no longer required, more specific mappings, which are optimized for the application requirements, may lead to significant gains in performance [Ma90]. For example, the attributes of an object, which are represented as separate atoms of the respective molecule in the generic mapping, could all be represented as attributes of a single atom in the

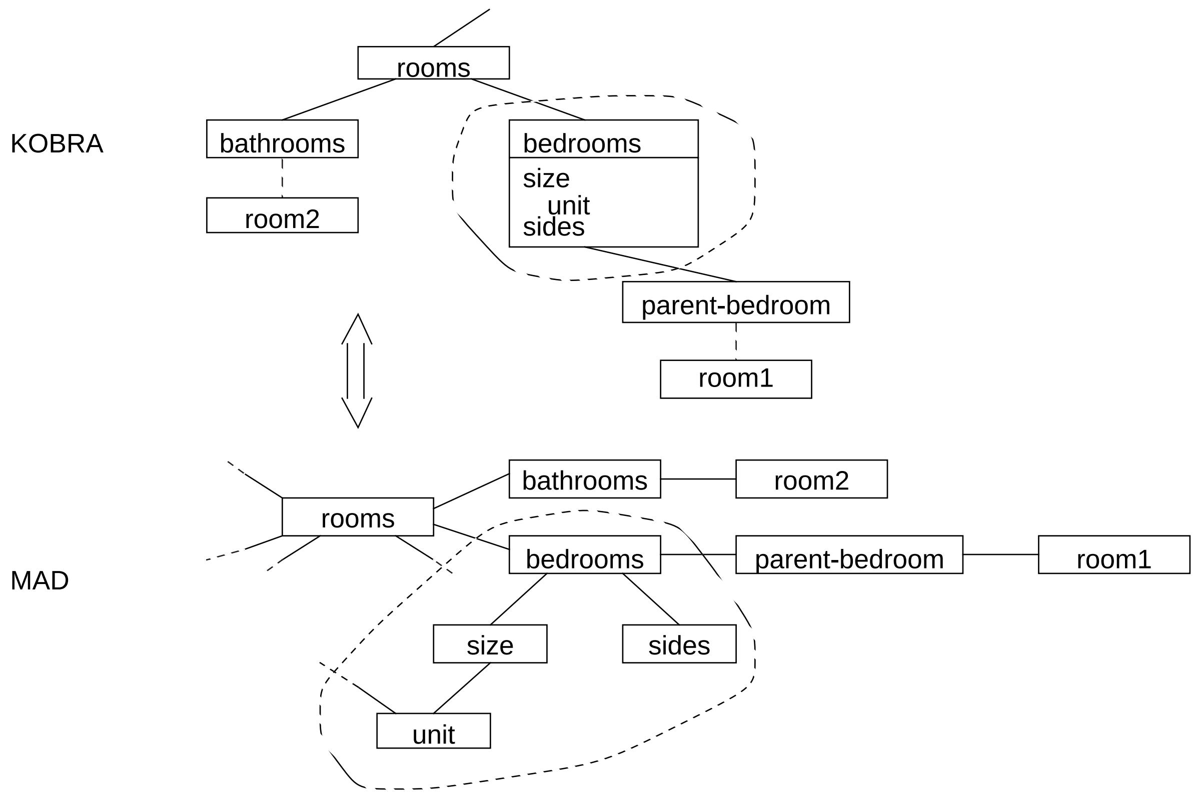


Figure 3.1: Mapping KOBRA ↔ MAD

specific mapping. In order to exploit such specific mapping schemas, the transformation process of KOBRA objects into MAD atoms must not be fixed to a certain mapping scheme (e.g., the one described above), but has to be performed in such a way, that it can be guided by some kind of heuristics or meta-information, which describe an optimal mapping scheme supporting the structures and processing characteristics of a specific application [Ma90].

3.2 Discussing the Delegation of Integrity Maintenance

In the following, we will discuss from different points of view the consequences of a partial delegation of the integrity maintenance from the application-oriented component of the system to the DBMS.

Implications of processing characteristics

Two basic approaches can be identified, by means of which the delegation across the hardware boundary between workstation and server component could be accomplished:

- All modifications of objects are not accumulated in the application buffer, but immediately reflected at the server side. In this case, the DBMS could generally maintain the integrity of the application. Of course, this case can be immediately outruled, since it clearly contradicts our efforts to support locality of references by an application buffer and multiplies communication between the components.
- All integrity checks are deferred to the end of the complex design transaction. Then, the DBMS must check the integrity upon check-in of the application buffer contents. This assumption clearly restricts the notion of integrity provided at the workstation component. Especially in the area of engineering applications, where the design process is considered a long-term activity and therefore long-duration design transactions are re-

quired, integrity has to be checked repeatedly within the transaction at iteratively performed design steps in order to ensure a consistent design. The nesting of operations may additionally lead to different levels of integrity checking (see chapter 2.5), which cannot be accumulated into one single level. Furthermore, the application may require certain integrity constraints to be checked immediately (e.g. consistency of abstraction hierarchies, value class checks, ...), which contradicts the deferred case. Moreover, the violation of constraints may require flexible reactions, that are, for example, provided by the demon concept involving interactions with the user who may again wish to change or refine the design object under consideration, leading to additional check-ins or communication overhead. Therefore, an approach to defer integrity checks to the end of the design transaction will cover only a small amount of the semantic integrity constraints of the application and does not deliver the flexibility necessary in the considered application domains.

Note that these implications are not limited to KRISYS, since they are generally based on processing characteristics of the applications and not on modeling constructs. Therefore the same arguments apply to the DBMS-kernel architecture in general, where application-orientation is achieved by different mechanisms (e.g. ADTs) provided at the workstation.

Considering the different semantic levels of the components

With respect to the modeling constructs provided, MAD and KOBRA can be clearly associated with different semantic levels. An interconnection between the two levels is established by a transformation or mapping process (chapter 3.1). The integrity constraints describing the application domain are, as we have seen in chapter 2, strongly tied to the semantics of other modeling constructs, as for example, the abstraction concepts, and are therefore more easily maintained at a level where the semantics of these constructs is controlled. A delegation of integrity constraints to the MAD model would require the constraints to be also included into the transformation process, meaning that integrity constraints specified in KOBRA would have to be mapped to semantically equivalent constraints specified in an appropriate language based on the MAD model. Consequently, the resulting MAD constraints would be very complex even for simple KOBRA constraints, since they would have to mimic in some sense the transformation process necessary to achieve the KOBRA semantics. Additionally and most importantly, errors and results of integrity checks requiring the interaction with the user would have to be appropriately reinterpreted at the higher level.

Efforts to introduce generally modeling constructs of KOBRA (e.g., abstraction concepts, rules, user-defined functions, etc.) into MAD in order to make the model more powerful and close the semantic gap between the two models would not offer solutions, but only shift the problems into the MAD model. The implementation of a new data model, which is now located at a higher semantic level, would certainly rely on an internal interface supporting similar semantics and providing the same services as in the existing MAD model. For such a new data model, the question would arise, where the borderline between server and workstation should be drawn. If the borderline is attached to the internal interface, then we are basically considering the same architecture as provided by KRISYS, and therefore all arguments elaborated in this paper may be equally well applied. If the borderline is attached to the enhanced data model interface, the additional functionality is completely integrated into the server. As a consequence, the additional semantics of the new data model either has to be repeated at the workstation (implying, that the enhancement of the DBMS is not necessary), or the server is hopelessly overloaded with additional processing, since the execution of methods, control of integrity, as well as rule based processing would now be shifted together with the abstraction concepts to the server. Workstation oriented processing, on the other hand, would be limited to a minimum of capacity, leading to a more or less centralized processing environment, that neglects the advantages provided by a workstation/server environment.

Implications of a flexible mapping process

The problems associated with the mapping of integrity constraints from one semantic level to another become even harder, if a flexible transformation process, which has been described in chapter 3.1, is considered in order to support mappings tailored to a specific application. Depending on the characteristics of the specific mapping process, the constraints may have different forms or can be specified only in some cases, but not in all. This is due to the fact that the same KOBRA construct may be mapped to different MAD constructs depending on the mapping process (e.g., a slot may be mapped to an atom, or to a specific attribute of an atom). The mapping process as a whole therefore becomes extremely difficult. For example, the cardinality constraint of a slot in KOBRA can be directly mapped to MAD, if the slot is represented in MAD as an attribute of an atom. This is, however, not possible if the slot is mapped to an atom.

As a consequence, either some integrity constraints have to be checked at different levels, depending on the mapping, or the variety of mappings is restricted to those that allow the mapping of integrity constraints as well.

Summary

Let us summarize the results of the above discussion. Considering the underlying processing characteristics, the delegation of semantic integrity maintenance to the server would either restrict the types of integrity to be specified for the application domain, or lead to an increased communication between workstation and server. As far as the semantic levels of the two components of KRISYS are concerned, the control of integrity is more easily performed at the KOBRA level, since it supports the semantics necessary for the specification or control of the constraints. Maintaining application-dependent integrity at the MAD level requires an appropriate transformation process for integrity constraints, resulting in complex constraints that mimic part of the semantics of KOBRA. Additionally, the flexibility required for mapping information between the two levels additionally increases the complexity of the transformation required for the constraints. In all cases, a significant amount of processing is shifted from the workstation to the server, thereby neglecting the advantages of a workstation/server environment.

We therefore conclude that the maintenance of the semantic integrity of an aplication should be performed at the workstation and not delegated to the server. It is of course necessary that a certain amount of consistency is maintained by the server, taking into account the constraints inherent in the DBMS data model. Otherwise the basic structural consistency of the database can not be guaranteed. Additionally, consistency considerations related to multi-user operation at the server and the synchronization of rollback operations, at the server and the workstation require additional concepts. Nevertheless, additional facilities for maintaining user-defined constraints at the DBMS are superfluous.

3.3 Consistency Support of the DBMS

First of all, the information contained in the DB has to be consistent with respect to the data model of the server, i.e., it has to obey the model-inherent constraints of the MAD model. In this context, the main task of the DBMS is to guarantee the basic structural consistency of the objects represented in the MAD model, i.e., the referential integrity and the cardinality restrictions used to model relationships between objects has to be preserved. This task is performed by the DBMS in a self-correcting manner [Sch90], meaning that the deletion of objects or modification of links automatically lead to correcting modifications in symmetric link attributes. Since abstraction relationships between objects are directly transformed into referential links at the server, the symmetry of the abstraction relationship attributes addressed as a model-inherent constraint in chapter 3.2 is directly reflected by the model-inherent consistency of the MAD model, thereby making the transformation process safer and easier to implement.

The most significant support for integrity provided by the server is, however, related to the structure inherent in the workstation processing. As already described in chapter 2.5, methods are regarded as nested units of processing which carry application semantics (e.g., expressed as certain levels of integrity). Demons and rules can be viewed in the same sense, since the activities performed by a demon may lead to the activation of other demons or require other kinds of integrity checks. If a constraint violation occurs in one of the actions provoked by a demon, which cannot be appropriately handled, the effects of the actions already performed have to be undone. Similarly, a rule-based inference process can be viewed as consisting of nested units of processing, since it may activate, as part of a reasoning process, other reasoning processes on different sets of rules or lead to the activation of demons when accessing attributes. For example, a forward chaining process may additionally incorporate backward chaining processes for determining certain subgoals.

Because the operations performed at the workstation may in some cases request additional services from the server (e.g., in order to fetch additional objects into the buffer), the units of processing at the workstation implicitly correspond to units of processing or data exchange at the server. The specific semantics of the processing unit is, however, not relevant to the server. It is necessary to appropriately coordinate the processing units of the workstation and the server, otherwise the effects of undoing the workstation operations cannot be reflected appropriately at the server. This coordination has to reflect especially the nesting of the units of processing (e.g., in the sense of nested transactions).

4. Conclusions

In this paper, we have described the maintenance of semantic integrity in KRISYS, a KBMS designed for supporting complex applications in a workstation/server environment. KRISYS is architecturally divided into two main parts: the DBMS, located at the server, performs data management tasks in an efficient manner, providing as its interface the MAD model, which supports dynamic definition and handling of complex objects as well as basic structural consistency (e.g., key uniqueness, referential integrity) of a database. The second part, which is placed at the workstation, realizes KOBRA, the knowledge model of KRISYS, which offers semantically enhanced modeling constructs (abstraction concepts, behavioral object-orientation, rules) as well as mechanisms for maintaining complex integrity constraints (e.g., model-inherent constraints, attribute-value restrictions, demons, etc.).

Considering semantic integrity issues, it has to be determined, whether the maintenance of integrity can to some extent be delegated to or supported by the DBMS. We have discussed this question from different points of view. An examination of the relevant processing characteristics reveals that a possible delegation would either multiplicate the communication overhead between workstation and server or restrict the flexibility of integrity maintenance in an unacceptable way. From a semantic point of view, the different levels at which KOBRA and MAD can be located suggest that integrity should be maintained at the KOBRA level, because the integrity constraints rely on the semantics of the knowledge model (e.g., certain conditions of the constraints may be based on abstraction relationships). A delegation to the MAD level requires a complex transformation process for integrity constraints and additionally shifts considerable amounts of work to the server side. In the same sense, the enhancement of the MAD model by KOBRA modeling constructs (e.g. abstraction concepts, object behavior, etc.) may overload the data model with application-specific aspects.

We therefore conclude that the semantic integrity of an application should be maintained at the workstation. The support provided for this task by the server can only be of a basic nature. As a consequence of this approach, not only local consistency (i.e., consistency of objects located in the application buffer of the workstation), but also global consistency of the knowledge model, involving relationships to other objects stored in the server, has to be checked at the workstation. Of course, our architectural philosophy in such cases requires the transfer of additional data affected by the constraints from the server to the workstation, but on the other hand avoids cumbersome mappings of integrity constraints as well as interpretation of low-level error reports in the case of a failure, and allows integrity to be checked step by step, timed with the requirements of the application. When processing at the workstation is finished and the relevant data is checked in, the DBMS only has to guarantee constraints inherent in its data model.

Summarizing, our conclusions consolidate the architecture of KRISYS and the DBMS-kernel approach in general, since they follow the above described architectural philosophy. An appropriate alternative to this architectural philosophy can only be developed for an environment not based on workstation/server cooperation. Since we consider a workstation/server environment as necessary in the support of complex application, current efforts to increase the modeling power of data models without considering the above philosophy (e.g. [DKM85, KDM88, Sch89, Sch90]) tend to lead into the wrong direction.

Following the design decision which have been motivated in this paper, we will use KRISYS for more thorough investigations concerning integrity maintenance in a workstation/server environment. From this future work we will hopefully gain a deep understanding of the requirements addressed to both server and workstation and of the tasks they have to perform, leading to a redesign or refinement of the KRISYS prototype, which has been implemented in a more or less "ad-hoc" fashion.

Acknowledgement

N. M. Mattos has contributed to clarify and improve the description of important issues of the paper. Also acknowledged are the useful hints of T. Härder.

References

[Ba88] Barth, G. (ed.): Informationstechnik it, Vol. 30, No. 6, 1988, Schwerpunktthema: Nicht-prozedurale Programmierung.

[BTW85] Blaser, A., Pistor, P. (eds.): Proc. Datenbank Systeme für Büro, Technik und Wissenschaft, GI-Fachtagung, Klarsruhe, März 1985.

[De90] Deßloch, S.: Enforcing Integrity in the KBM KRISYS, to appear in: Proc. of the Second Int. Worksthop on Foundations of Models and Languages for Data and Objects, Aigen, Austria, September 1990.

[DKM85] Dittrich, K.R., Kotz, A.M., Mülle, J.A.: DAMASCUS - ein Datenhaltungssystem für den VLSI-Entwurf, in: [BTW85], pp. 70-72.

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

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

[HHMM88] Härder, T., Hübel, Ch., Meyer-Wegener, K., Mitschang, B.: Processing and transaction concepts for cooperatino of engineering workstations and a database server, in: Data and Knowledge Engineering 3 (1988), pp. 87-107.

[HMMS87] Härder, T., Meyer-Wegener, K., Mitschang, B., Sikeler, A.: PRIMA - A DBMS prototype supporting engineering applications, Proc. 13th Int. Conf. on VLDB, Brighton, 1987, pp. 433-442.

[HR85] Härder, T., Reuter, A.: Architektur von Datenbanksystemen für Non-Standard-Anwendungen, in: [BTW85], pp. 253-286.

[In87] Inference Corporation: ART Reference Manual, Version 3.0, Inference Corporation, Los Angeles, CA, 1987.

[KDM88] Kotz, A., Dittrich, K.R., Mülle, J.A.: Supporting Semantic Rules by a Generalized Event/Trigger Mechanism, in: Schmidt, J.W., Ceri, S., Missikoff, M. (eds.): Advances in Data Base Technology - Proc. of the Int. Conf. on Extending Data Base Technology, Venice, Italy, 1988, pp. 76-91.

[Ko89] Kotz, A.: Triggermechanisms in Database Systems (in german), IFB 201, Springer-Verlag, Berlin, 1989.

[Ma89] Mattos, N.M.: An Approach to Knowledge Base Management - requirements, knowledge representation and design issues, Dissertation, Fachbereich Informatik, Universität Kaiserslautern, April 1989.

[Ma90] Mattos, N.M.: An Approach to DBS-based Knowledge Management, in: Proc. 1. Workshop Informationssysteme und Künstliche Intelligenz, Ulm, März 1990.

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

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

[Sch89] Scholl, M.: A synthesis of complex objects and object orientation, in: Proc. Workshop on Foundations of Object Models and Languages, Lessach, September 1989.

[Sch90] Schöning, H.: Preserving Consistency in Nested Transactions, in: Proceeding of the 23rd Annual Hawaii Int. Conf. on System Sciences (HICSS-23).

[SYB88] SYBASE-DateServer, product information, SYBASE Inc., Berkeley, 1988.