in: Proc. 8th Int. Conf. on Entity Relationship Approach, Toronto, Canada, 1989, pp. 159-173.

Modeling with KRISYS: the Design Process of DB Applications Reviewed

N. M. Mattos, M. Michels

University of Kaiserslautern, Department of Computer Science,
P.O. Box 3049, 6750 Kaiserslautern, West Germany
e-mail: mattos@uklirb.uucp, phone: (0631) 2053280

During the last few years, various approaches to an extension of existing models with abstraction mechanisms have been proposed. Abstraction concepts enrich the semantics of the underlying data or knowledge model, but advocate modeling methodologies which are quite different from conventional ones: these mechanisms, in contrast to existing DB techniques, support a modeling process in a step-wise fashion. Our KBMS prototype KRISYS allows for such an incremental construction of data or knowledge bases by means of a complete and integrated support of the abstraction concepts provided by its knowledge model. In this paper, we show how the user can employ KRISYS to model some real world situations, thereby demonstrating that the abstraction concepts are at the focal point of the mentioned incremental modeling process.

1. Introduction

Since the early years of the Computer Science, most of the research activities in its various fields (Programming Languages, Software Engineering, Database Systems (DBS), Artificial Intelligence (AI), etc.) focuses on issues for facilitating the specification and management of the ever increasing complexity of software [1]. Based on the assertion that a computer system constitutes a model of the real world, about which it contains specific information pertinent to a particular application [2,3], the process of software development can be viewed as the process of building an accurate model of some enterprise. Thus, especially data modeling in the area of DBS and knowledge representation in the area of AI deal with the above mentioned software problem by working on the development of models that naturally and directly reflect the user's conceptualization of the real world [4]. This should facilitate the task of the modeler (as well as the interaction system/users), accelerating the process of software development as a whole. To achieve this goal, both fields try to improve the semantics of existing models by identifying and incorporating into them the constructs people apply during the process of describing the real world.

In [4], we have shown that most of these constructs have their roots in epistemological methods for organizing knowledge; i.e., in trying to describe the real world, people usually involuntarily apply some *abstractions* (classification, generalization, association, and aggregation) in order to organize their knowledge in some desired form. Abstraction permits one to suppress specific details of particular objects, emphasizing those pertinent to the problem or view of information at hand. It is therefore the fundamental tool used for organizing knowledge and is, for that reason, one of the most important constructs to be supported by any data or knowledge representation model.

Numerous papers [1,5,6] have tried to describe one or another concept and many models [7,8,9] have been developed following some of these descriptions. Nevertheless, approaches towards an integration of all these concepts into one model are rarely found. Most models developed concentrate on the support of one or two abstraction concepts, not taking the existence of the other ones into account. We argue, however, that object descriptions should embody all abstraction concepts so that objects of the model, like the real world entities, can play different roles (i.e., class, set, aggregate, instance, element, or component) at the same time, depending only on the relationships they have to other objects. Note, for example, that an object such as a concrete automobile is at the same time an instance of the class automobiles, an aggregate composed of the several automobile parts, and an element of the products of some particular factory. This integrated view of the abstraction concepts, which has been presented in [4], was incorporated into the knowledge model of a prototypical Knowledge Base Management System (KBMS), called KRISYS, implemented at the University of Kaiserslautern [10]. The knowledge model of KRISYS supports an object-centered representation of the application world. That is, an object incorporates not only organizational characteristics of the world (i.e., abstraction relationships) but also descriptive aspects of the real world entities (i.e., attributes with further specifications) and its operational aspects (i.e., object behavior).

Certainly, as models incorporating such a rich spectrum of concepts begin to be applied to model some real world situation, the corresponding modeling process turns out to be quite different from conventional ones. Abstraction mechanisms, in contrast to existing DB techniques, advocate a modeling process in a stepwise fashion. That is, in each step, only some of the details of the problem are considered so that those details less relevant to the subproblem at hand are neglected until some later step [1]. In such environments, the modeler may take advantage of the reasoning facilities provided by the abstraction concepts [4,11] in order to facilitate his task by exploiting the system to make deductions about the objects as well as to guarantee the structural and semantic integrity of the data or knowledge base. In other words, we are defining a new modeling methodology in which the data or knowledge model is also an active agent. Since such a model also reflects the user's view of the real world, it may at the same time be used as a model for information structuring of the application world (like the ER-model [12]) and as a model for the DB-schema design and DB manipulation. Therefore, we are leaving the stiff "world" of information structuring, subsequent DB-schema design, and posterior phase of DB manipulation to get into a very dynamic and flexible "world" in which modeling is done stepwise, thereby eliminating the difference between information structuring, DB-schema design, and DB manipulation almost completely.

Following this shift of viewpoint, this paper describes this incremental modeling process, thereby aiming at two different goals: first, it shows how a modeler can employ our KBMS prototype to model some real world situations, and second, it demonstrates that the mentioned abstraction concepts are at the focal point of this process.

2. The Example

Our modeling process will be illustrated by means of a knowledge-based application that will become more complex in the course of our discussion. Here, we just introduce a brief description of the situation to be modeled, as well as the task of our Knowledge-based System (KS).

The reader should imagine a first class restaurant in a city, let us say, like Paris or New York, where guests may select from several menus the finest and most exotic dishes and wines for their meal. The purpose of the KS is to

- advise guests in combining dishes and wines (e.g., suggesting red or white, French or Rhine wine, etc.) considering, however, known preferences since VIP guests usually frequent the restaurant regularly, and
- automatically control some conventional functions in order to supply the restaurant owner with important information such as necessity of wine ordering, stock statistics, and guests' preferences.

3. The Modeling Process

The first step in developing our application (figure 1) is to conceptualize the real world situation of its application domain (conceptualization phase). The result is expressed by means of a knowledge model (structuring phase) which is refined in a stepwise manner in order to improve it with more semantics (refinement). Finally, a validation phase is performed causing a kind of feedback to the previous ones in order to enrich as well as to correct the KB incrementally until the KS exhibits the same level of know-how as human experts.

In the following, we describe the development process of our hypothetical KS, showing how the activities underlying this process are performed by means of KRISYS. For sake of clarity (and due to space limitations), we concentrate on the modeling aspects of the knowledge about the application domain, neglecting the existence of expert knowledge (i.e., problem solving know-how like rules) almost completely.

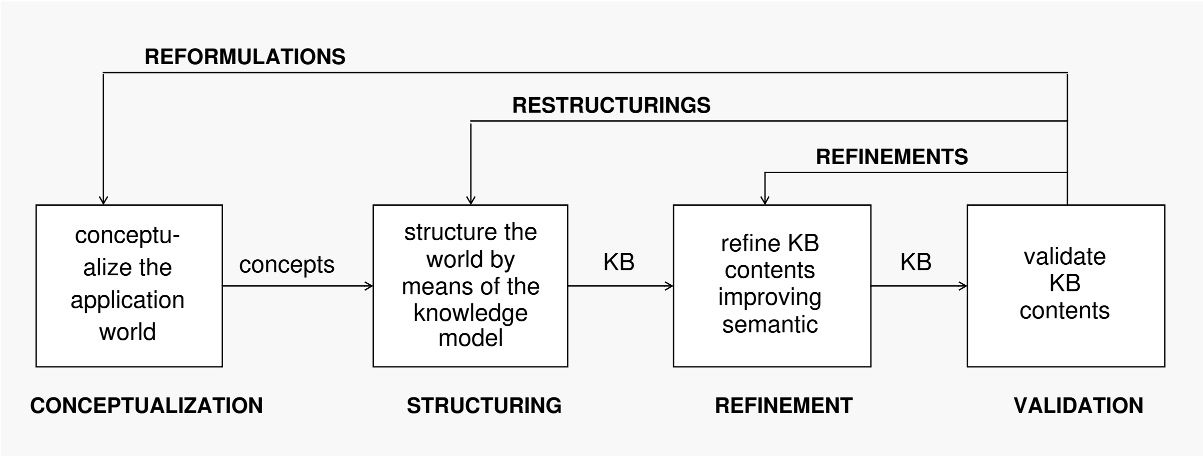

Figure 1: Evolutionary modeling process

3.1 Conceptualization

Usually, when a knowledge engineer analyses an application world, he identifies objects, also called entities, which are characterized by their properties. For instance, in our restaurant, different kinds of wine exist, and each of them has properties such as the year of vintage, site, kind of vine, number of available bottles, etc. Even events are viewed as entities: the visit of a guest has the properties guest's name, date of visit, dishes and wines enjoyed, etc. Each object is associated with some operations which may change the object's properties representing in some sense the behavior of the object as well as its interface to other objects. For example, ordering a bottle of wine is an operation associated with visits. This behavior decreases the number of available bottles of the ordered wine as well as updates the list of chosen wines of a visit.

In addition to properties and operations, objects have relationships to other objects which carry some important information. Most of these relationships are domain dependent. For example, a particular dish is related to some wines which should under all circumstances be recommended. However, there are important domain independent relationships that occur in nearly every domain and as such have well defined semantics. They are known as abstraction concepts (e.g., classification, generalization, association, and aggregation) and well understood by their underlying relationships (i.e., instance-of, subclass-of, element-of, subset-of, part-element-of, and subcomponent-of); they play a key role in knowledge modeling since they provide natural ways to structure and organize objects (for details about these concepts and a definition of their semantics see [4,1,5,6]). In our hypothetical example, all kinds of wine are grouped by classification into a new object, the class wines. Analogously, other beverages as well as dishes, guests, and visits may be grouped into the corresponding classes (figure 2).

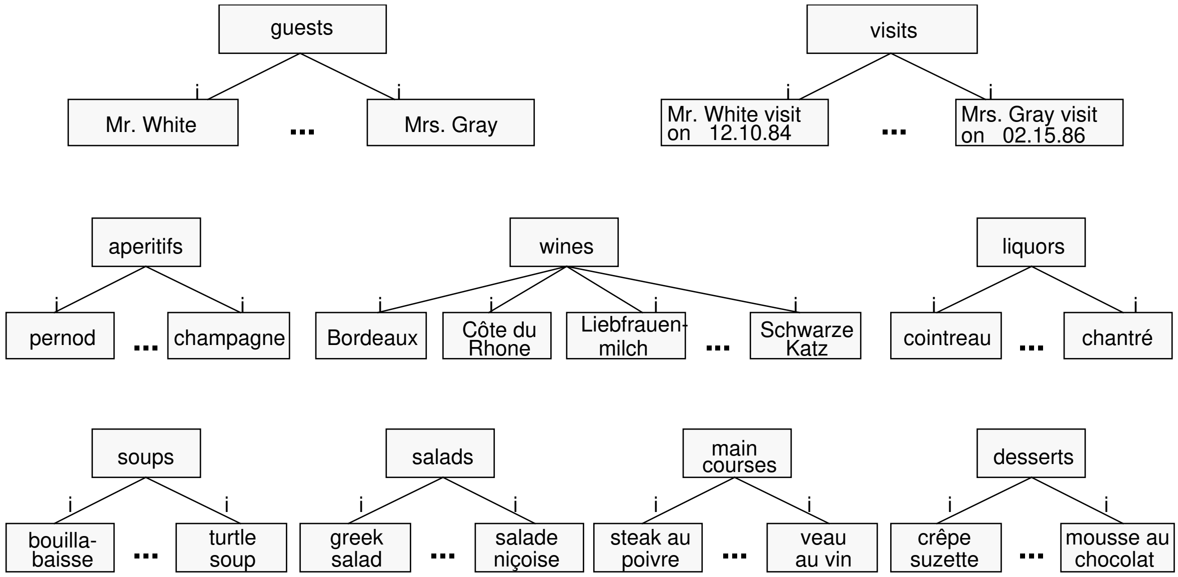

Figure 2: Classifying objects (i = instance)

Classes are generalized by extracting their common attributes: soups and salads are generalized to "hors d'œuvres", and recursively, "hors d'œuvres", main courses, and desserts are generalized to dishes (figure 3).

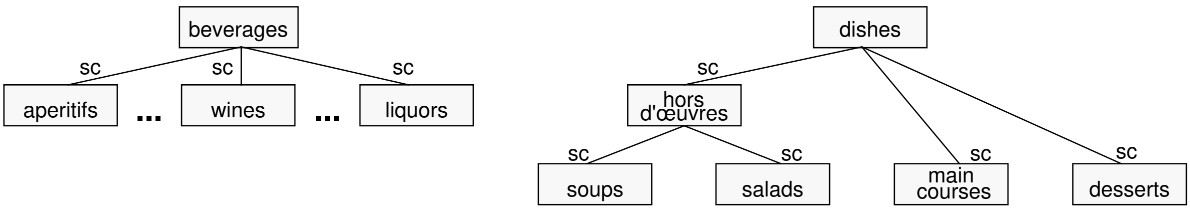

Figure 3: Generalization of objects (sc = subclass)

Objects fulfilling common conditions are grouped into sets by association: all visit objects having Mrs. Gray as their value of guest's name form the set of Mrs. Gray's visits represented by the object Mrs. Gray. Recursively applied, association results in supersets: the sets French wines and Rhine wines are subsets of European wines (figure 4).

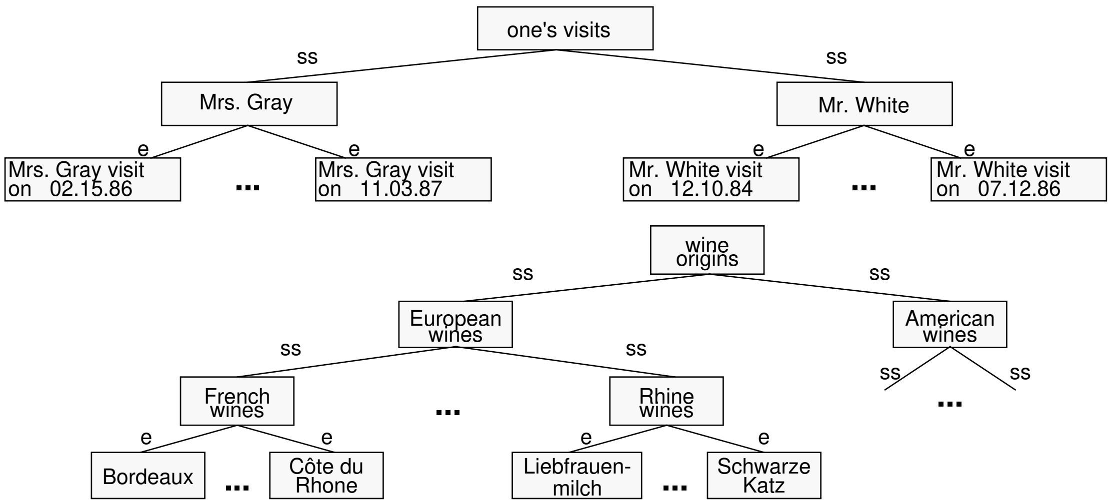

Figure 4: An association hierarchy (e = element, ss = subset)

Finally, a menu has an "hors d'œuvre", a main course, and a dessert as its parts defining a kind of aggregation relationship. Aggregation may be also applied recursively. For example, a main course, let us assume "steak au poivre", always has further parts: it is composed of a steak, some rice, and vegetables, which, in this case, is composed of peas and carrots (figure 5).

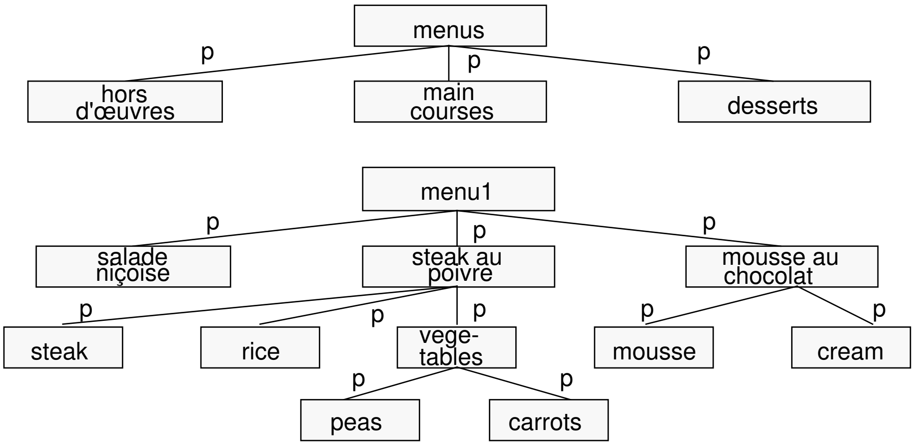

Figure 5: Aggregation of objects (p = part)

As already mentioned in the introduction, it is important to recognize that a real world object does not represent a class, set, instance, etc. by itself, but only in the context of being related to other objects. The object "hors d'œuvres" plays the role of a class because it has subclasses or because it has a superclass, and the object Mrs. Gray plays the role of a set because it has elements. As any object may be related to other objects by different abstraction concepts, it may at the same time play different roles representing a set, part, instance, etc. For example, desserts is a subclass of dishes and a subcomponent (part) of menu. Menu1 is an instance of menus and an aggregate of "salade niçoise", a steak, rice, peas, carrots, a "mousse", and cream. In the same way, Mrs. Gray represents the set of all her visits but is at the same time an instance of guests.

The most important aspect of the abstraction concepts is that they are the basis for drawing particular conclusions about objects (the so-called built-in reasoning facilities) [4,11]. There are several kinds of reasoning possible. The following are some of them:

- Instances of a class are also instances of the class' superclasses, e.g., because "veau au vin" is an instance of main courses and this is, in turn, a subclass of dishes, it may be reasoned that "veau au vin" is an instance of dishes. (The same holds for objects that are related by association or by aggregation).
- Classes define the structure with associated constraints of their instances and subclasses. From the fact that Bordeaux is an instance of beverages, the conclusion that Bordeaux has, at least, the properties and operations prescribed by beverages can be drawn.
- Since elements of a set always fulfil the set membership stipulation, it is possible to deduce that all elements of French wines are produced in France.
- Sets may possess some properties that describe the characteristics of the group of their elements as a whole. As such, the set Mrs. Gray has, for example, properties defining the average of expenses during her visits, the most frequented week day, etc. Since such set properties are in reality based on characteristics of each individual element, conclusions about the values of set properties can be drawn from the element characteristics.
- In the aggregation, reasoning is made based on certain properties of the involved objects and expressed by means of the so-called implied predicates. For example, menus are composed of "hors d'œuvres", main courses, and desserts; therefore, the calories of a menu is composed of the sum of the corresponding parts' calories. The same may occur for the menu's price, weight, etc.

By observing the above definitions, one may conclude that different kinds of properties with distinct semantics are involved in the abstraction concepts. Consider for example the object wines (figure 2), which has all existing kinds of wine as instances. This class object might have properties describing its instances such as year of vintage, site, and number of available bottles. Viewed as a set, the same object might have some properties describing the group of elements as a whole (e.g., most desired wine, most expensive wine, total number of bottles, etc.). Obviously, when an object represents both a set and a class, the set properties should not affect the instances, and the instance properties should not be used for describing the set. Additionally, instance properties do not have values because they abstractly define the structure of the instances; values are not assigned until instantiation. On the other hand, set properties possess a value because they describe characteristics of the group of elements. This even holds when instances and elements are the same as in the case of the above example. In an analogous way, when combining aggregation with the other concepts, instances and set properties should not be confused with those used to express the aggregation (i.e., aggregation properties). The latter ones are different from the other properties of the objects (i.e., those concerned with classification/generalization and association) since their values are to be interpreted as other objects of the knowledge base. In other words, while aggregation properties represent parts of the objects, set and instance properties express characteristics of them. (A mechanism that copes with different kinds of properties will soon be presented when showing how to map this conceptual view of real world objects onto the KRISYS knowledge model.)

3.2 Structuring

So far, we have described how real world objects are organized. But how can the KBMS KRISYS be employed to model such objects? In general, KRISYS provides an easy and convenient way to model real world situations according to the concepts presented above. Each entity is represented by a so-called schema (not to be confused with a DB-schema!), which has attributes that describe declarative aspects of the schema (slots) and attributes that describe procedural aspects (methods). To model the abstraction concepts, standard slots occurring in each schema are used: has-instances and instance-of for classification, has-subclasses and subclass-of for generalization, has-elements, element-of, has-subsets and subset-of for association (aggregation is represented by user-defined slots).

Therefore, KRISYS allows for an integrated representation of the real world. That is, rather than having one schema in the model for the representation of each role played by a real world entity (i.e., class, set, instance, component, etc.), a schema represents all roles of this entity at the same time. Consequently, the object-centered representation of KRISYS allows for a one-to-one correspondence between entities in the application domain and schemas in the model, preventing, for example, the knowledge about an entity (e.g., a wine) to be spread out among several objects of the model. Following this approach, figure 6 shows the integrated representation of our application world by means of the knowledge model provided by KRISYS.

Figure 7 presents some entities of our restaurant application represented as KRISYS schemas. (For sake of clarity, only the most interesting attributes are shown.) Their attributes may be of different types: Ownslots and ownmethods are used to describe properties of the object itself, and as such may have values. Instanceslots and instancemethods on the other hand, describe properties of the object's instances, and have therefore no values. Ownslots and instanceslots are further classified in non-terminal or terminal. Non-terminal slots indicate part-of properties (i.e., the components) of the objects since their values correspond to other objects of the model, whereas terminal slots describe either characteristics of the objects themselves (terminal ownslot) or of their instances (terminal instanceslot). Methods are always of type terminal.

After defining the structure of KB objects, KRISYS undertakes the maintenance of the structural and semantic integrity of the KB by means of the reasoning facilities provided by the abstraction concepts. For example, inheritance is employed to ensure that each instance or subclass has, at least, the attributes prescribed by its superclasses. KRISYS guarantees that such integrity holds even if changes to the KB structure occur: if the attribute phone number is added to the object guests, it will be inherited immediately by all subclasses and instances of guests. In the same manner, if the property site is removed from the class wines, KRISYS automatically withdraws it from each existing wine in the KB. In KRISYS, each instance attribute, independent of being of type terminal or non-terminal, is inherited by the subclasses as instance attribute and by the instances as own attribute; own attributes are not inherited at all.

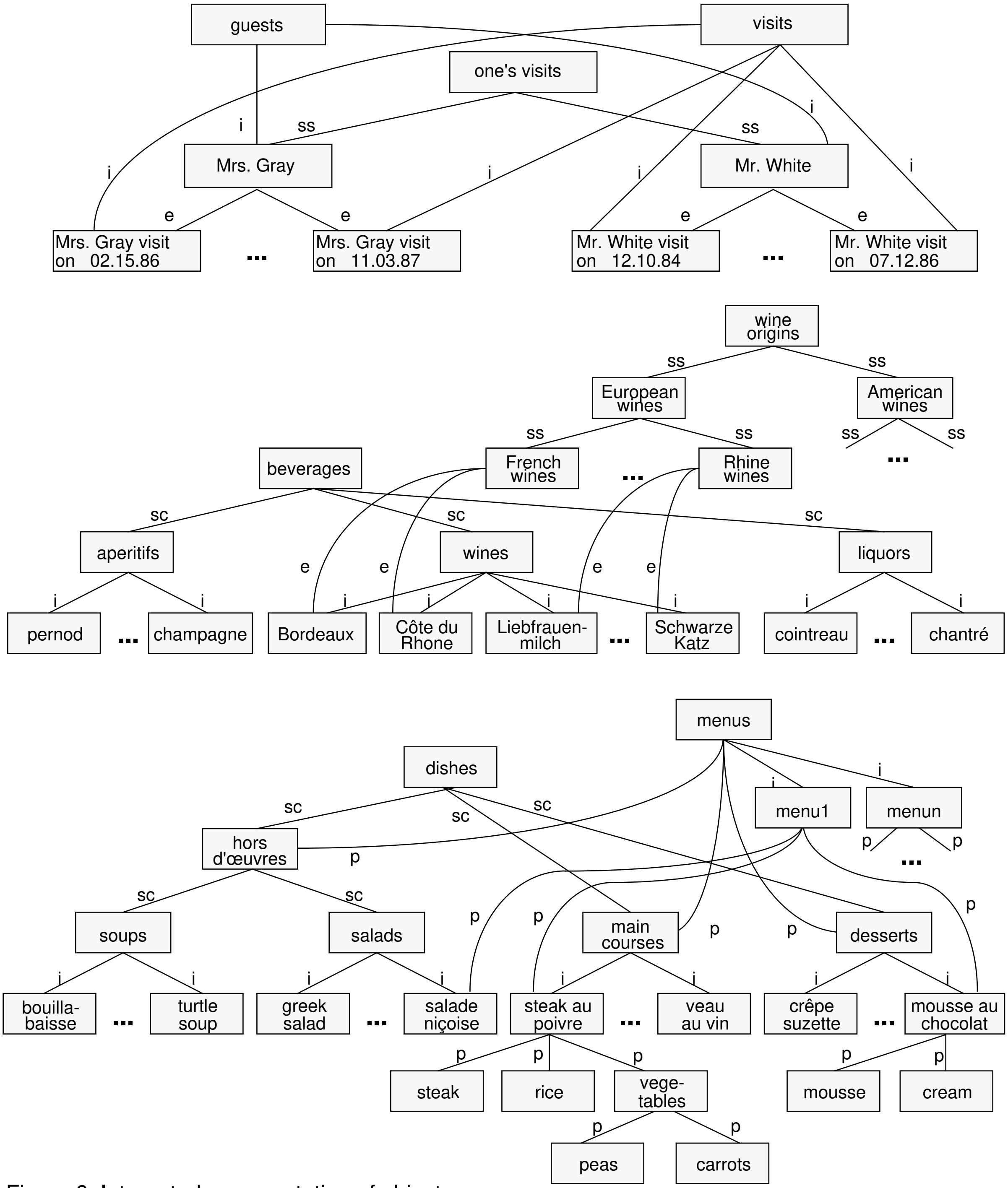

Figure 6: Integrated representation of objects

KRISYS also guarantees that every element of a set satisfies the corresponding membership stipulation. As such, when one erases the value France from the slot which defines the land where a particular wine is produced, this wine is automatically removed from the set French wines. Following, a recalculation of the values of the set properties of French wines (e.g., number of different wine sorts, average of price, etc.) is also automatically performed by KRISYS.

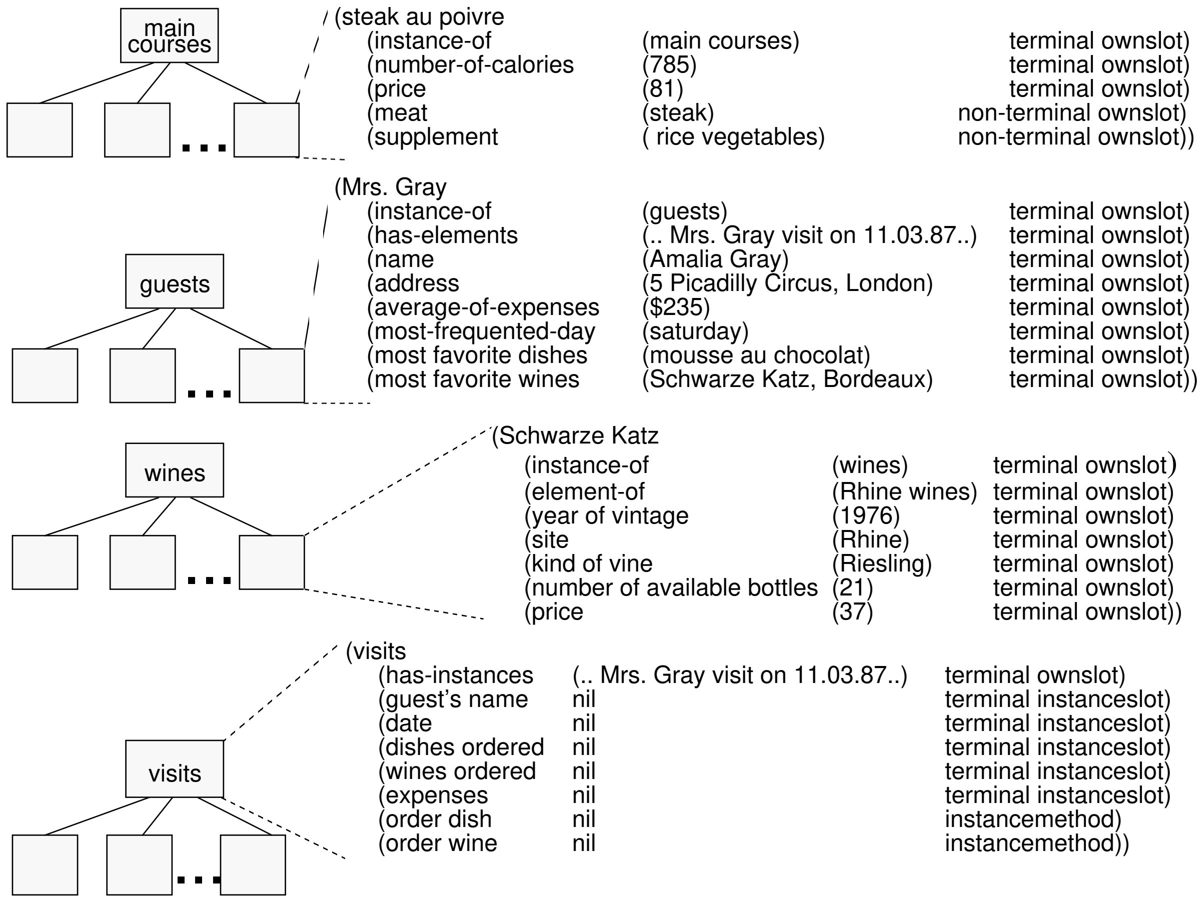

Figure 7: Some objects of the restaurant KB as KRISYS schemas

In aggregation hierarchies, the maintenance of the semantic integrity is kept by guaranteeing the truth of the specified implied predicates. As such, when the cook invents a new way to prepare steaks, thereby reducing their quantity of calories, KRISYS automatically reconsiders the number of calories of all objects composed of steaks (e.g., "steak au poivre", menu1, etc.).

Finally, operations are modeled by methods and invoked by message passing: ordering a bottle of wine means sending a message to the corresponding instance of visits. This message, which is passed along with some parameter values, as for example number of bottles, will then activate the method 'order wine'. Methods are allowed to invoke further methods, thus establishing an object-oriented communication facility for KRISYS schemas.

Another kind of operations are the reactions to real world events. For example, when the "number of available bottles" of a wine falls short to 20, some kind of supply should be ordered. This situation can be managed by so-called demons that watch for changes in the corresponding attributes. In the example, a demon is automatically invoked after modification of "number of available bottles" to examine whether it is necessary to order or not.

In KRISYS, demons are linked to those attributes describing in some sense the situation of activation. A demon can be attached to several attributes permitting that the same demon may be used for several purposes even for attributes with very distinct semantics. Additionally, demons may be specified by their own, so that there is no need for deleting a demon when it is unlinked from an associated attribute. Demons are always automatically invoked by the system when an operation is performed on an attribute. If the attribute is a slot, possible operations are get-value, change-value, add-value and retract-value. In this case, demons can be invoked either before or after the operation. (Such specification is done by the knowledge engineer). There is even the possibility of defining one before and one after demon for the same attribute. If the attribute is a method, the only possible operation is send-message, and demons may be invoked on entry and on exit.

A latter important modeling construct supported by KRISYS are rules of the form IF <condition> THEN <action>, which are used by the system to automatically infer some information when required by the user. In the KRISYS environment, rules that are applied to solve similar problems are grouped by association into one ruleset. For example, all rules that are used for determining a suitable wine for a chosen dish are grouped together. Rulesets need not be disjoint so that each rule may be an element of several rulesets which are the units of reasoning in KRISYS. If a guest preferring French wines chooses a "steak au poivre", KRISYS can suggest him to order a Bordeaux wine by employing its forward reasoning mechanism over a particular ruleset. On the other hand, if the guest chooses a Bordeaux wine, the system can use just the same ruleset (that means, the same group of rules) and conclude by backward reasoning that "steak au poivre" or "veau au vin" are the most suitable dishes for this wine.

3.3 Refinement

The next step after having defined the structure, the objects, and the operations of the KB is the so-called refinement phase. Such phase involves, in general, the addition of further semantic integrity constraints to the previously defined KB contents. Basically, two kinds of semantic integrity constraints exist which are involved in a KS application. The first kind represents, as already mentioned, the existing reasoning facilities underlying the abstraction concepts. They always have the same semantics and are, for this reason, permanently controlled by KRISYS when abstraction relationships have been defined (see conceptualization). As such, by every visit of Mrs. Gray the set properties of the object Mrs. Gray (average of expenses, most frequented week day, etc.) are automatically recalculated guaranteeing the correctness of the inherent semantics of the abstraction concepts. The second kind of semantic integrity constraints are those dependent on the application domain. They are also automatically controlled but do not depend on specified abstraction relationships: e.g., the value of 'number of available bottles' of the object wines must be a non negative integer, and 'most favorite wines' in guests must not have more than three values since it is desirable to record only the most favorite ones.

In KRISYS, both types of integrity constraints are expressed by aspects describing more exactly the attribute to whom they belong. The predefined aspects possible-values and cardinality are employed to model the above mentioned value and cardinality constraints (see figure 8). Membership stipulations are expressed as an aspect of the predefined slot has-elements (see object French wines in figure 8). To assume that the number of available bottles has a value of zero when no value is stored in this slot, the default-value aspect is used. Default-method may be applied to define a standard behavior to all instances of visits. To specify that the number of calories of a dish is composed of the sum of the corresponding parts' calories, the aspect implied-predicates may be used (see "steak au poivre"). Finally, user-defined aspects can be used to individually characterize an attribute, e.g., unit specifies that expenses are expressed in US$.

For a particular attribute, aspects are only defined once, i.e., if an attribute is inherited, it has initially the same aspects as the original attribute. KRISYS allows for modifications of aspect definitions as long as these modifications do not violate the semantics of the abstraction concepts. It is therefore possible to restrict possible-values or cardinality specifications when moving downwards in a superclass-subclass hierarchy. The extension, however, of such specifications is not permitted since the instances or subclasses having this extended specification will not be in agreement with the definition of their superclasses. The same occurs with the membership stipulations. That is, it is possible to restrict but not to extend such stipulations when moving downwards in a superset-subset hierarchy, otherwise elements of the subset are not necessarily also elements of the superset. KRISYS guarantees therefore that the semantics of an attribute remains the same throughout the KB. The maintenance of the attribute semantics is very proper when inheriting default values, which is especially useful for methods. (Remember that inheriting values does not make any sense). For example, the method 'order wine' in visits has the same code for each instance of visits and, consequently, can be defined just once as a default method in visits. Each instance of visits inherits 'order wine' as ownmethod; if no new value is assigned to the ownmethod, the default method is then automatically referenced when 'order wine' is activated.

Up to here, only integrity constraints concerning one attribute have been mentioned. Clearly, there are also integrity constraints spanning several attributes of different objects. For example, the object stock in the KB has as value of its attribute 'number of available bottles' the sum of the values of 'number of available bottles' of each instance of wines. This constraint can be controlled by demons. In this situation there are two possibilities: (a) a demon updates the sum of bottles kept by the stock object after each modification of the attribute 'number of available bottles' of any wine, or (b) every time the 'number of available bottles' of the whole stock is read, a demon first computes the sum and assigns this attribute.

Figure 8 presents some of the restaurant objects in more detail, i.e., including demons, value restrictions, etc. By now, the attribute structure can be fully understood: each attribute is represented by its name, its value, its type (i.e., non-terminal or terminal, ownslot, instanceslot, ownmethod or instancemethod) and its aspects. Value and aspects may be also empty lists.

```
(guests
    (has-instances              (.. Mrs. Gray ..)    terminal ownslot)
    ...
    (most favorite dishes       nil                  terminal instanceslot(cardinality [0 3])))

(Schwarze Katz
    (instance-of                (wines)              terminal ownslot)
    ...
    (number of available bottles (21)                terminal ownslot (possible-values  (and  (integer)
                                                                                               (interval <0 ∞>)))
                                                                      (default-value    (0))
                                                                      (demons           (stock demon))))

(visits
    (has-instances              (.. Mrs. Gray visit on 11.03.87..)    terminal ownslot)
    ...
    (expenses                   nil                  terminal instanceslot   (unit      (US$)))
    (order wine                 nil                  instancemethod (default-method     (<method code>))))

(french wines
    (has-elements               (.. Bordeaux, Côte du Rhone..)        terminal ownslot)
                                     (membership-stipulation (and (instance-of wines)
                                                                   (equal France (slotvalue produced-in))))
    (number-of-different-sorts  (97)                 terminal ownslot))

(steak au poivre
    (instance-of                (main courses)                        terminal ownslot)
    ...
    (meat                       (steak)                               non-terminal ownslot)
    (supplement                 (rice, vegetables)                    non-terminal ownslot)
    (number-of-calories         (785)                                 terminal ownslot
                                     (implied-predicates (sum(all(slotvalue number-of-calories))))))
```

Figure 8: Objects of the restaurant KB in more detail

3.4 Validation

At the end of the refinement phase, a new (or a first) version of a KS prototype is ready, and the knowledge validation phase begins. Here, an explicit representation of all aspects of knowledge is particularly important in order to allow for an easy localization of knowledge incorrectnesses. Generally, prototypes perform poorly at the start because of an erroneous transformation to objects of the knowledge model, an initial neglect of some refinements, or even because the knowledge engineer simply learns some more details about the application domain. By diagnosing the weakness and deficiencies of the KS, the knowledge engineer can go back to some of the developing phases either to conceptualize, structure, or refine some further aspects of the application world previously introduced into the KB, leading to an incremental, evolutionary development. Even after being constructed, KS will be continuously growing and expanding their capabilities by including new expertise into the KB so that, in reality, the described cyclic development never comes to an end.

Further refinements or changes of existing specifications are certainly the easiest modifications to be performed in the KB. These are directly achieved by using KRISYS operations to create new aspects, delete old ones, restrict them, introduce new demons, etc.

Restructuring usually generates new concepts or changes on existing ones to be modeled in the KB. The same occurs with reformulations after a new conceptualization of the real world. In our restaurant situation, the class wines exists, comprising all kinds of wine available in the restaurant. In order to supply guests with detailed information about wines, the head waiter wants to differentiate between the wines offered in the "carte" according to the several existing vines. This requires the introduction of further subclasses of wines into the KB (e.g., white wines, red wines, rosé wines, Riesling wines, Ortega wines, etc.), where the possible-values specification of the property 'kind of vine' previously defined in the superclass wines is correspondingly restricted. After this, the instances of wines have to be connected with one of the introduced subclasses according to their kind of vine (e.g., wines produced with Riesling vines are connected with the class Riesling wines which in turn is a subclass of white wines as shown in figure 9).

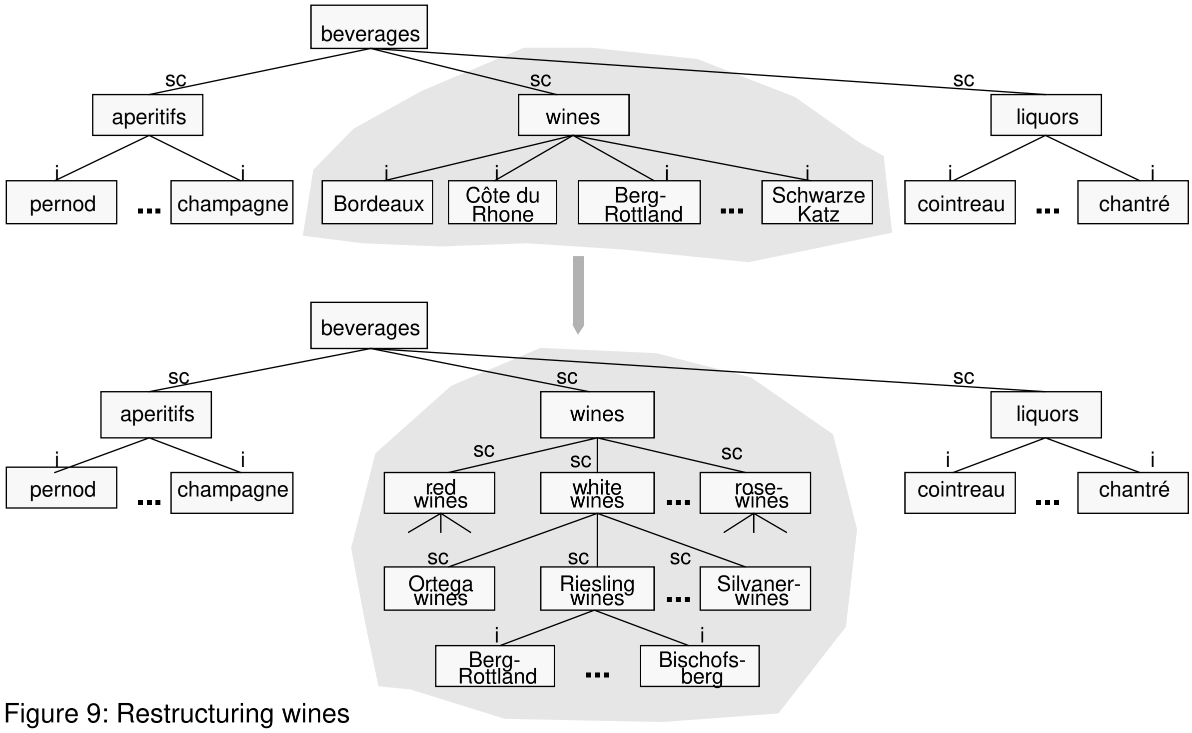

Figure 9: Restructuring wines

The chef is also not satisfied with the defined structure of dishes. He objects to the fact, for example, that cold dishes are not a kind of main course but a kind of "hors d'œuvre". Thus, the knowledge engineer has to introduce cold dishes as a subclass of "hors d'œuvres" and change the superclass of the corresponding dishes to that new class (figure 10).

Restructuring comprises adding new objects to the KB, inserting new classes or sets into existing hierarchies, changing the superclass of classes or instances, adding and deleting attributes, renaming objects and so on. In general, the most radical changes come from adding or deleting abstraction relationships between objects. But even in case of these radical changes, KRISYS provides direct and flexible ways to achieve such restructurings, keeping the correctness of the affected built-in reasoning facilities. By doing so, KRISYS exploits the semantics of the abstraction concepts in order to use their built-in reasoning facilities as a kind of "system rules" [13] that help in restructuring and keeping KB consistency (see [4] for a specification of such built-in reasoning facilities). Particularly, when disconnecting an instance from its class, the instance need not be deleted but may exist on its own thereby keeping its characteristics which are not dependent on its relationship to the class. This means that the knowledge engineer can restructure the KB as much as necessary without losing any of the correct information previously introduced into the KB.

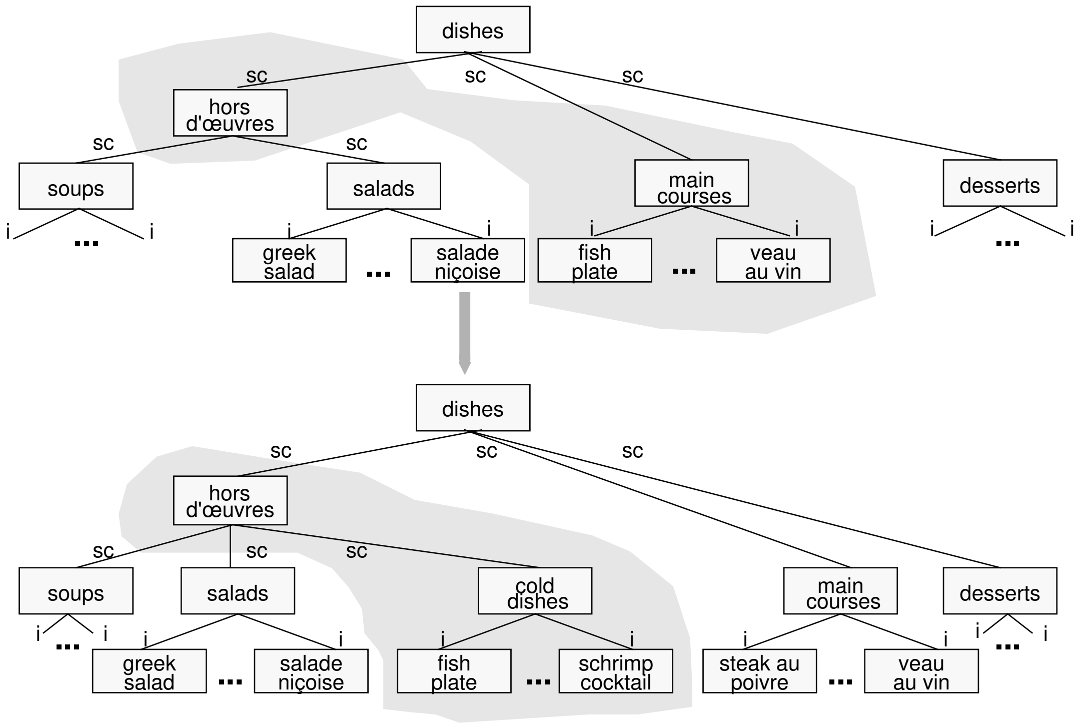

Figure 10: Restructuring dishes

A final kind of restructurings or reformulations with special significance are those involved with extensions of the kept knowledge or of the KS as a whole. For example, it may be important to extend the functionality of our hypothetical KS in order to automatize the printing of restaurant bills. Such extension may be easily achieved by defining a new method ('printing bill') for the class visits which, when activated, calculates the total expenses of the guest, updates the corresponding slot, and then prints the bill. Extensions may be therefore introduced effortlessly without affecting any aspect of the existing knowledge.

4. Short Notes on Our KBMS Approach

In the previous chapter, we have shown how a modeler can employ our KBMS approach to model a real world application. In order to support the described incremental modeling process appropriately, our KBMS should firstly be a kind of neutral modeling tool offering a rich spectrum of concepts for modeling with which the whole knowledge of an application world may be suitably represented. Secondly, it should pay attention to performance requirements in order to allow for an efficient management of the modeled knowledge base.

In reality, KBMS requirements are not however restricted to these two points. KBMS integrate an expressive and flexible representation of knowledge and a reliable and efficient management of the knowledge base. Therefore, traditional DBS aspects such as knowledge or data independence, KB distribution, multi-user facilities, availability, reliability, etc. should also be supported by KBMS. Particularly in KRISYS, these aspects are considered not by the component supporting the knowledge model (i.e., the engineering layer) but by distinct ones (figure 11). The application layer aims at the independence of knowledge. At its external interface knowledge is viewed in an abstract manner, independent of the representational framework supported by the engineering layer. Here it is seen functionally, in terms of what the KS can ask or tell the KBMS about its domain. This functional view of the KB is achieved by a powerful query language supported in the object-abstraction interface following in certain aspects the ASK and TELL approach of KRYPTON [14,15]. This query language is based on the model-theoretic approach [16] and presents some further features not supported by KRYPTON, such as set-orientation, user defined projections, recursion, etc. [17]. The goal of the implementation layer is to efficiently cope with storage of knowledge and its supply to the engineering and application layers. At this level, most of the issues are related to traditional database problems applied to large KB possibly shared by multiple users: storage structures, search techniques, efficiency, control of concurrent access, logging and recovery mechanisms, etc. For this reason, this layer is realized by a DBS kernel [18] which was specially developed for the support of the so-called non-standard applications (KS, CAD/CAM, geography etc.).

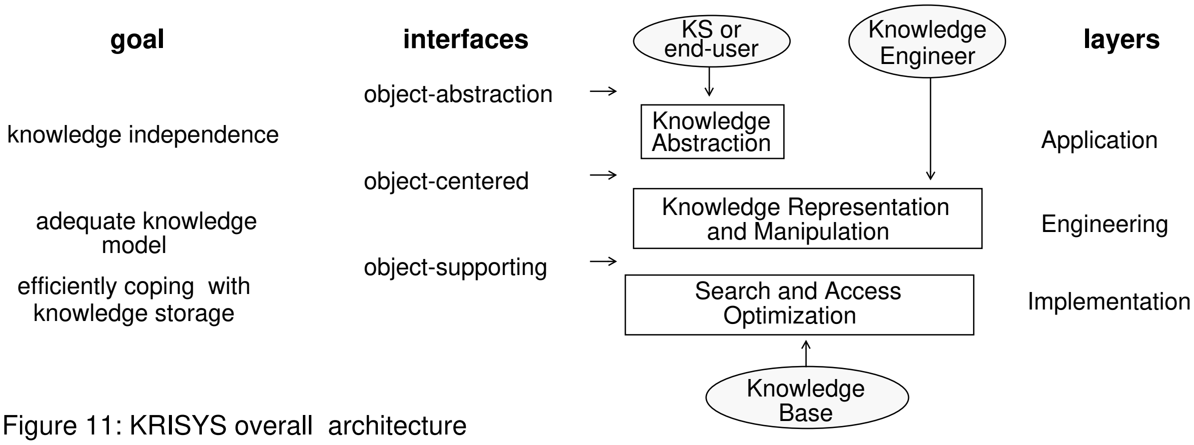

Figure 11: KRISYS overall architecture

In this paper, we have concentrated on the modeling aspects of the engineering layer, i.e., the layer which focuses on a KBMS from the point of view of the KB designer. For this reason, the contents of the paper might ignore some important considerations, reflecting implementation aspects of KBMS. (We refer to [19] for a detailed description of the implementation of our KBMS prototype). In KRISYS, the effective support of these modeling requirements is achieved by a mixed knowledge representation framework which focuses equally on declarative, operational as well as organizational aspects of knowledge. KRISYS allows for an accurate representation of the whole descriptive characteristics of the application domain, i.e., objects, properties, relationships, and constraints. It offers constructs to represent procedural characteristics of the application world such as behaviors of the domain objects (methods), reactions to real world events (demons), and situation-action rules to express the problem solving know-how. However, the most important constructs provided by the knowledge model of KRISYS are the mechanisms for knowledge organization. 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.

The support of all these abstraction concepts is one of the aspects that differentiate KRISYS from existing expert system tools like ART [20], KEE [21], KNOWLEDGE CRAFT [22], LOOPS [23], etc. as well as from existing DBS. Whereas KRISYS 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, KRISYS incorporates all roles of a real world entity in one single schema, which may describe any of the mentioned abstraction objects, depending on the existing relationships to other objects. Therefore, when modeling the application world, there is no need to introduce two distinct representations to support both association and generalization/classification in the model (as in the case of ART and KNOWLEDGE CRAFT). 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). Following the lines of its integrated view of KB objects, KRISYS avoids the introduction of redundancy into the KB keeping at the same time the semantics of each abstraction concept very clear. Even proposed extensions of DBS technology (semantic data models [24], object-oriented data models [25], etc.) focus only on some of these concepts (mostly aggregation and/or generalization/classification) and neglect most of the underlying reasoning facilities.

Finally, KRISYS treats the abstraction concepts as dynamic relations, which may be changed by the KB designer or the application at any time. The corresponding built-in reasoning is, in such cases, automatically applied, keeping the KB in a semantically consistent state. Inheritance, for example, is in some systems (ART and LOOPS) not more than a means to save some typing during the modeling process. Changes in the structure of a class (deletion, creation of attributes, etc.) are either not allowed or not reflected in the structure of the existing instances, leading to severe inconsistencies. In KRISYS, every time that relevant information is 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. It is, therefore, also possible to insert an object without any relationship to a class and then dynamically establish or change classification relationships. This capability which is very important to support the described incremental modeling process is neither possible in DBS nor in most of the above mentioned tools.

5. Conclusions

This paper describes the modeling process of a hypothetical application by means of our KBMS prototype KRISYS. At the focal point of this process are the abstraction concepts, advocating a construction of the knowledge base in a stepwise fashion. In other words, this modeling process is realized by a cyclic series of activities including a conceptualization of the application world, a structuring of the knowledge base, a kind of refinement to improve the semantics of the knowledge base, and a validation. The whole series of activities or only some of them are then repeated until the modeled knowledge base directly reflects the user's view of the corresponding application world.

Here, it is important to emphasize that the described modeling process with KRISYS is quite different from conventional ones. Firstly, KRISYS can be used as a very powerful modeling tool, which allows for a direct extension, modification, and restructuring of the user's conceptualization of the application domain. For this reason, its knowledge model may be used as a model for information structuring of some application world like, for example, the ER-model. And secondly, due to the integrated and complete support of the abstraction concepts, KRISYS explicitly integrates meta-information into the knowledge base, eliminating the existing difference between DB and DB-schema and, consequently, between DB-schema design and DB manipulation. A strict separation of DB-schema design and DB manipulation, which is required by existing DBS, does not reflect the real world situation, where changes that affect the application model may also occur. Thus, KRISYS allows for an interaction of information structuring, DB-schema design, and DB manipulation, providing a knowledge model that integrates these activities into one single process. As a consequence, this modeling process turns out to be

- more consistent - since the user deals with just one model (rather than with one for information structuring and one for DB design and manipulation), thereby diminishing the risk of losing relevant information or of introducing mistakes everytime the application world has to be represented by means of a different formalism-,
- more flexible - since it is possible to repeat previous modeling steps in order to make corrections or improvements on the knowledge base -, and
- easier - since the knowledge model is also an active agent in this process, continuously making deductions as to object structures and checking the semantic integrity of the knowledge base -.

Acknowledgment

B. Mitschang, S. Dessloch, and F.J. Leick have read an earlier version of this paper and have contributed to clarify and improve the description of some important issues. Also acknowledged are the useful hints of T. Härder as well as the work of the participants of the KRISYS Project.

References

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

[2] Abrial, J.R.: Data Semantics, in: Data Management Systems, (eds.: Klimbie, J.W., Koffeman, K.L.), North-Holland Publ. Comp., Amsterdam, Netherlands, 1974.

[3] Bobrow, D., Collins, A. (eds.): Representation and Understanding, Academic Press, New York, 1975.

[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] Brodie, M.L.: Association: A Database Abstraction for Semantic Modelling, in: Proc. 2nd Int. Entity-Relationship Conference, Washington, D.C., Oct. 1981.
[6] 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.
[7] Brachman, R.J., Schmolze, J.G.: An Overview of the KL-ONE Knowledge Representation System, in: Cognitive Science, Vol. 9, No. 2, 1985, pp. 171-216.
[8] Codd, E.F.: Extending the Database Relational Model to Capture More Meaning, in: ACM Transactions on Database Systems, Vol. 4, No. 4, Dec. 1979, pp. 397-434.
[9] Hammer, M., McLeod, D.: The Semantic Data Model: A Modelling Mechanism for Data Base Applications, in: Proc. of the ACM-SIGMOD Conf. on Management of Data, 1978, pp. 26-36.
[10] 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.
[11] Rosenthal, A., Heiler, S., Manola, F., Dayal, U.: Query Facilities for Part Hierarchies: Graph Traversal, Spatial Data, and Knowledge-Based Detail Supression, Research Report, CCA, Cambridge, MA, 1987.
[12] Chen, P.P.: The Entity-Relationship Model - Toward a Unified View of Data, in: ACM Transactions on Data Base Systems, Vol.1, 1976, pp. 9-36.
[13] Banerjee, J., Kim, W., Kim, H.-J., Korth, H.F.: Semantics and Implementation of Schema Evolution in Object-oriented Databases, in: Proc. of the ACM-SIGMOD Conf. on Management of Data, San Francisco, 1987, pp. 311-322.
[14] Brachman, R.J., Fikes, R.E., Levesque, H.J.: Krypton: A Functional Approach to Knowledge Representation, in: Computer, Vol. 16, No. 10, Oct. 1983, pp. 67-73.
[15] Levesque, H.J., Brachman, R.J.: Knowledge Level Interfaces to Information Systems, in: On Knowledge Base Management Systems (eds. Brodie, M.L., Mylopoulos, J.), Springer-Verlag, New York, 1986, pp.13-34.
[16] Gallaire, H., Minker, J., Nicolas, J.-M.: Logic and Databases: A Deductive Approach, in: ACM Comp. Surveys, Vol. 16, No. 2, June 1984, pp. 153-185.
[17] Mattos, N., Dessloch, S.: KOALA - A Query Language for a New Generation of DBS, internal report, University of Kaiserslautern, 1989, submitted for publication.
[18] Härder, T., Meyer-Wegener, K., Mitschang, B., Sikeler, A.: PRIMA - A DBMS Prototype Supporting Engineering Applications, in: Proc. 13th VLDB Conf., 1987, pp. 433-442.
[19] Mattos, N.: An Approach to Knowledge Base Management - requirements, knowledge representation, and design issues -, Doctoral Thesis, University of Kaiserslautern, 1989.
[20] Clayton, B.D.: Inference ART Programming Tutorial, Vol. 1 Elementary ART Programming, Vol. 2 A First Look At Viewpoints, Vol. 3 Advanced Topics in ART, Los Angeles, CA, 1985.
[21] 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.
[22] Carnegie Group Inc.: Knowledge Craft CRL Technical Manual, Version 3.1, 1987.
[23] Bobrow, D., Stefik, M.: The LOOPS Manual, Palo Alto, Calif., Xerox, 1983.
[24] Peckham, J., Maryanski, F.: Sematic Data Models, in: ACM Computing Surveys, Vol. 20, No. 3, September 1988, pp. 153-189.
[25] Dittrich, K.R.: Advances in Object-Oriented Database Systems (Proc. of the 2nd Int. Workshop on Object-Oriented Database Systems), Springer-Verlag, Berlin, 1988.