*In: Proc. 3rd Int. Workshop on Next Generation Information Technologies and Systems (NGITS'97), Israel, 1997, pp. 215-222.*

# Enriching Object-Relational Databases with Relationship Semantics

Nan Zhang, Theo Härder

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

Joachim Thomas

*IBM Toronto Laboratory*
*1150 Eglinton Av E, North York, Ontario, Canada*
*e-mail: jthomas@torolab.vnet.ibm.com*

## Abstract

*Data relationships play a very important role in application domains and thus should be a key issue in database modeling. However, they are still not well considered in the emerging object-relational database (ORDB) technology, where the main concern is put on the extension of the relational data model (RDM) with objects as basic modeling constructs. This paper addresses exploiting and supporting data relationships in ORDBs. The semantics of various data relationships are investigated with the help of an application example from manufacturing industry. The purpose of this work is to integrate relationship semantics into object-relational data models (ORDMs), that is, to provide a general mechanism which facilitates the explicit specification of relationship semantics. As a result, as much semantics of data relationships as possible can be captured, database management systems (DBMSs) will manipulate data in a consistent way, and users are to a large extent freed from having to take care of implementation details.*

## 1. Introduction

There is a trend to unify pure object-oriented (OO) and pure relational database (RDB) technology to allow users to effectively model and manipulate complex data and use OO technologies for application development without losing the benefits of SQL and all its robust features. This trend is reflected by the emergence of object-relational database management systems (ORDBMSs) [25, 15, 14, 29]. However, current research in this area is focused on extending relational database management systems (RDBMSs) with OO features. Considerably less effort is spent to let applications really take advantage of this new technology. Especially, data relationships with their complicated semantics and impact on both modeling as well as implementation deserve further study.

Data relationships have been extensively investigated in the design of RDBs [28, 7]. Those relationships are value-based, and their semantics is guaranteed through hard-wired referential integrity facilities of a DBMS. Most OO approaches [20], on the other hand, map different types of relationships to a single construct, reference pointers. As those pointers embody little semantics, consistency of relationships must be completely maintained inside application programs.

ORDMs embody the evolution of the RDM in the OO direction. This marriage is expected to retain the value-based information representation which enables set-oriented queries and, at the same time, support reference-based navigation. As a result, ORDMs provide powerful modeling facilities. However, adequate support of data relationships is still missing. We want to extend ORDMs beyond this limit. The central idea is to supply a mechanism as a front-end to an existing ORDBMS, which helps to explicitly specify semantically rich data relationships and to automatically generate enforcement methods for the maintenance of semantics. Relationship types are introduced to facilitate extensibility and the incremental implementation of a prototype system.

To illustrate our approach, we will refer to our project RITA [13] as the modeling example, whose goal is to develop an integrated information system for a leading manufacturer of automobile components. In the initial phase of the project, focus is put on the department of technical testing that performs tests on prototypical technical objects. Besides the standard data types, our information system also needs data types for representing long texts of requirement documents and test reports, sketches of test patterns, photos and videos of test progress, etc. Moreover, the data structures range from simple flat structures (e. g., part catalog) to heterogeneous compound ones (e. g., CAD objects). For these complex data types ORDBMSs promise feasible data processing and management support. Since data relationships are of particular interest, we will select only a set of representative data structures to facilitate the discussion.

In Sect. 2, we will give an overview of ORDMs. Based on the application modeling prototype, Sect. 3 illustrates various kinds of data relationships ranging from application-specific relationships to those having generic predefined semantics. The semantics of data relationships are addressed in Sect. 4. Sect. 5 describes our approach of embedding relationship semantics in ORDBMSs to allow designers to specify the desired semantics explicitly, as well as to enable the system to maintain the relationships' consistency. A comparison with related work is presented in Sect. 6. Finally, we conclude our paper in Sect. 7.

## 2. Object-Relational Data Models

At the time being, there exist several proposals for ORDMs, the most important of which is SQL3 [2] being followed by major database products such as [15, 14]. While SQL3 envisions entire objects to be stored in a single column of a table, another commercial system, UniSQL [29] proposes to identify object attributes with separate table columns. A comparable feature, in turn, is recently provided by SQL3 in the form of so-called *row types*. There is currently no generally accepted definition of "object-relational", nevertheless, some common features should build the foundation of an ORDM:

- One of the primary ideas behind object extensions is that, in addition to conventional built-in types, user-defined abstract data types (*ADTs*) may also be specified and used in the same way as the built-in types.
- *Objects* are associated with *types*. All objects of a type possess common behavior and common states. While *type* is an intensional concept, *class* extensionally defines an implementation of a corresponding type.
- *Inheritance* (single as well as multiple) is supported by organizing all types/classes into hierarchies.
- A conventional *table* is extended by allowing object references (of *reference types*), *ADTs*, and alphanumeric values as column domains. Users can attach procedures to a table and have them operate on the column values.
- To organize homogeneous as well as heterogeneous collections of objects, constructors such as *set*, *list*, *multiset* or *array* are also essential ingredients of an ORDM.

Despite all these modeling constructs, the support of data relationships in ORDMs is far too restricted. For example, association and aggregation abstraction relationships are usually not explicitly represented and must be modeled as a (non-transitive) set of references (*OIDs*) to the instances of other classes. Such an inexpressive modeling with a uniform semantics for all kinds of references causes many problems: There are no powerful, system-controlled operations that allow to easily specify and maintain different semantics. Hence, users must realize such operations in application programs and, as a result, are responsible for performance and consistency issues.

## 3. Data Relationships in Applications

To analyze relationship semantics more clearly, we describe a sample modeling at first.

### 3.1 Representative Application View

For better control of complexity, the graphical notations are introduced in Figure 1, where the rectangles contain entity names. Attributes and operations are ignored for the sake of brevity. Besides abstraction relationships [19] we identify also application-specific relationships, which are shown as lines between entities. Among the abstraction relationships, aggregation is represented with a black triangle while association and generalization are described with shaded and white triangles, respectively. Cardinality restrictions are indicated at the end of the connecting lines.

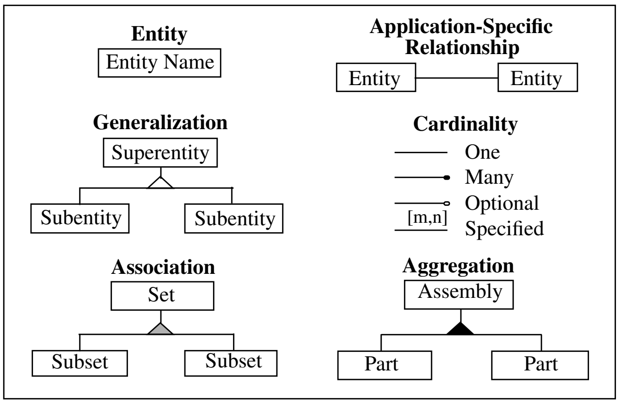


**Figure 1. Basic notations**

**Table 1. Data dictionary**

| Entity | Definition |
|---|---|
| *automobile* | consists of several parts; here we consider only three of them, i. e., *seat*, *door*, and *window*. |
| *unit* | of an automobile; can be specified by three categories: *seat*, *door*, and *window*. Each unit will undergo a set of tests for various purposes. |
| *requirement* | comes in three different forms: *legal-regulation*, *customer-requirement*, and *inhouse-standard*. Each kind of these requirements summarizes a set of test specifications. |
| *requirement-catalog* | is a set of requirements essential for a unit. |
| *test* | defines all attributes that are common to all tests. Each test must fulfil a set of requirements. |
| *dynamic-/ static-test* | describes the test suite of dynamic/static aspects of units. |
| *seat-/door-/ window-test* | describes the test suite of different kinds of units. |

Figure 2 illustrates a simplified view of the data structures in RITA with the explanation of some entities presented in Table 1. Several relationships are especially named for convenience of later discussion. They are *consist-of*, *unit-test*, and *requirement-test.*

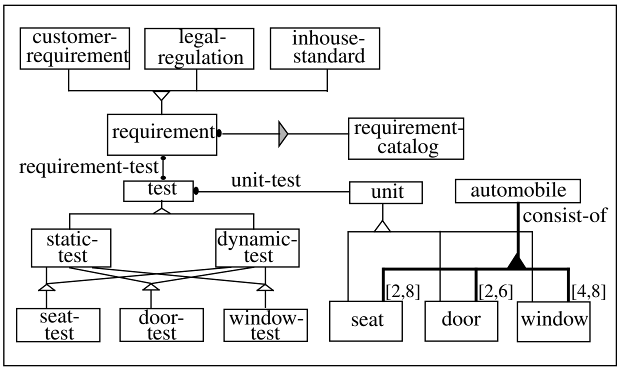


**Figure 2. A part of data model**

### 3.2 Various Kinds of Data Relationships

In our modeling example, there exist different kinds of data relationships, including not only generalization, association, and aggregation, which define abstraction relationships [19], but also application-specific ones.

**Generalization** (IS-A relationship) is a powerful abstraction for sharing structural and behavioral similarities among entities while preserving their differences. It describes the relationship between an entity and one or more derivatives of it. In Figure 2, for example, *test* exhibits characteristics common to all tests carried out for automobile units. *Dynamic-test* and *static-test* are two specified types of *test*. These two kinds of tests are further divided into three categories: *seat-test*, *door-test*, and *window-test*. Generalization is an important and useful construct not only for conceptual modeling but also for implementation, since inheritance facilitates the reuse of code. As a basic feature of all object-oriented or object-relational data models, it is well considered and supported.

**Association** (SUBSET-OF relationship) is an abstraction in which a relationship between member objects (or subsets) is considered a set object of a higher level. For example, an instance of *requirement-catalog* for an automobile unit is represented as a set of instances of *requirement*. Instead of characterizing the members themselves, set properties express properties of the set as a whole, which are determined based on the properties of the members. Besides, membership stipulations are valid for each member of the set. For example, supposing a set *requirement-catalog-of-seat* as a subset of *requirement-catalog*, it will have "*seat*" as value of the attribute *for-unit.* Moreover, association may span n levels, since it can be applied recursively. In ORDMs, "set" as type constructor is generally provided, however, the kind of association modeled in this way is restricted to a single level of reference without any transitive semantics of operations.

**Aggregation** (PART-OF relationship) is used to describe objects (aggregates, complex objects, or composite objects) that are assembled of certain parts. In this way, aggregations are special forms of object collections where an aggregate object is made up of components, and components are parts of the aggregate. In our example, seat(s), door(s), and window(s) together form an automobile, between them there exists an aggregation relationship *consist-of*, where *automobile* is the aggregate object with *seat*, *door*, and *window* as its parts. Since the aggregation concept specifies that a subobject should be an integral part of an aggregate, the aggregate and all its subobjects should be addressed and fetched collectively as a unit in a high level query (transitivity) and can be required to exist together (existence dependency). These properties have great meaning for referential integrity and some operations, as, e. g., *select*, *delete*, and *copy*. Therefore, this kind of relationship should be carefully identified and modeled. In most OODBMSs as well as ORDBMSs aggregation must be expressed by object references (e. g., attribute type *SET OF* <*ref*>). Warranting the semantics remains to be burdened on application developers.

**Application-Specific Relationships:** Besides the above described abstraction concepts, other kinds of data relationships with additional properties are needed in various application scenarios. For instance, in Figure 2, between *test* and *requirement* there is an m:n relationship *requirement-test*, and between *test* and *unit* there is an 1:m relationship *unit-test*. It is indispensable to maintain these cardinality restrictions in our application. Sometimes, a pair of matched references is employed to represent the interconnection. However, the fact that the forward and inverse references are dependent on each other as well as other kinds of semantics are not explicitly supported by DBMSs.

## 4. Relationship Semantics

We will now analyze properties of relationships, stressing that the data involved are not only structurally related to, but also semantically dependent on each other.

### 4.1 Structural Semantics

With respect to the connections of objects, there are the following kinds of semantics.

**Composition semantics:** Ordinary references carry no semantics of composition. It typically exist in abstraction relationships and has impact on the transitivity semantics (see below), which may not only refer to propagating operations over a hierarchy, but also to pass on structural or behavioral properties.

**Sharability:** This semantic concept is especially useful for abstraction relationships and determines whether an object of a lower level can participate in one or more relationship with another object of a higher level. For example, a seat, a door, and a window belong to only one automobile, therefore, the aggregation relationship *consist-of* is non-sharable. Moreover, sharability is also a factor influencing the existence semantics (cf., Sect. 4.2).

**Degree:** Relationships may be binary, ternary, or of higher degree. A binary relationship can be implemented as an attribute containing a reference to the related objects. For n-ary relationships, however, this simple structure is not suitable any more. In general, an n-ary relationship must be seen as an atomic unit and cannot be broken down into a set of binary ones without altering its proper meaning.

**Cardinality:** Cardinality characterizes the connectivity of a relationship and constrains the number of related objects. Cardinalities can be further refined and represented with pairs of values as [*min*, *max*], i. e., the minimum and the maximum number of objects that participate in a given relationship. As illustrated in Figure 2, an automobile can have at least two and at most eight seats.

### 4.2 Existence Dependencies

Existence dependencies specify how insertion or deletion of one object may influence the existence of connected objects. For instance, in the case of aggregation different kinds of existence dependencies w.r.t. *delete* operations might look as follows:

- On Deletion of Part
  - MDA (Mandatory Deletion of Aggregate): Upon deletion of one of the parts of an aggregate, the aggregate object is also deleted.
  - CDA (Conditional Deletion of Aggregate): The user may decide whether the deletion of a part incurs the deletion of the entire aggregate or not.
  - RDA (Restricted Deletion in the existence of Aggregate): The deletion of a part of an aggregate is not permitted. Only the aggregate object as a whole may be deleted (see below).
- On Deletion of Aggregate
  - MDP (Mandatory Deletion of Part): Upon deletion of the aggregate object, all part objects are deleted.
  - CDP (Conditional Deletion of Part): Upon deletion of the aggregate object, all part objects which are not participating in any other aggregation relationship are deleted.
  - RDP (Restricted Deletion in the existence of Part): The deletion of the aggregate object is rejected if it will leave a dangling part object.

In our example, the deletion of a seat, a window or a door will incur the deletion of an automobile and vice versa. In this sense, the aggregation relationship *consist-of* should be assigned with MDA and MDP semantics. Corresponding existence dependencies can also be defined, e. g., for *insert* operations and for association as well as other application-specific relationships.

### 4.3 Transitivity

Transitivity is a common property of many relationships. Since, for example, an instance of *seat-test* is also the instance of *dynamic-test* as well as of *test*, it will inherit features (such as attributes and methods) from *test* as well as from *dynamic-test*. Moreover, aggregation is inherently transitive. An aggregate has parts, which may in turn have parts, just like an automobile consists of several units, which can in turn consist of subparts. Whenever an aggregate object is requested, all part objects are also potentially accessed. For application-specific relationships, there can be also some operations implying transitive closure and operating on both directly and indirectly related objects.

### 4.4 Operation Propagation

According to propagation rules, operations applied to some starting objects will automatically incur operations on other related objects. As an example, operations at the aggregate level can be propagated to the part levels. This process provides a powerful way for specifying transitive actions and for securing consistency. Propagation is possible for both data manipulation operations as, e. g., *select*, *delete*, *copy*, and *display*, as well as data management operations such as *lock* and *save*.

### 4.5 Summary of Relationship Semantics

Table 2 summarizes the different aspects of relationship semantics discussed so far. Aggregation and association are taken as examples of generic relationships, for which some properties are standard (denoted as "+"), such as composition, transitivity and propagation. Other aspects vary in different concrete application domains, for instance, if a part can be associated with more than one aggregate or not, if a part can be reused or not after the aggregate is disassembled, etc. Therefore, these aspects (denoted as "-") should be further specified to catch the exact meaning in various scenarios. This consideration also leads us to model relationships in an inheritance hierarchy (cf., Sect. 5.1).

For application-specific relationships, we choose *consist-of* and *unit-test*. To reflect that an automobile consists of seats, doors, as well as windows, the relationship *consist-of* is defined. It is a special kind of aggregation with add-on semantics for cardinality, sharability and existence dependencies: Units can not be shared among different automobiles and should be deleted if the automobile does not exist any more. Another relationship *unit-test* embodies no composition and transitivity semantics. Moreover, more than one test can be carried out on a unit, but if the unit is deleted, all these tests should also be deleted. In the table, symbol "o" means that the corresponding semantic aspect does not exist or is not necessary for the relationship, while symbol ">" represents that this aspect is inherited from a more generic type of relationship.

**Table 2. Relationship examples with their semantics**

| Semantics | | Generic relationships | | Application-specific relationships | |
|---|---|---|---|---|---|
| | | Aggregation | Association | consist-of | unit-test |
| Cardinality | 1:1 | - | - | o | o |
| | 1:m | | | + | + |
| | m:n | | | o | o |
| Composition Semantics | | + | + | > | o |
| Sharability | | - | - | o | o |
| Existence Dependency | | - | - | MDA, MDP, ... | MD of *test* on deletion of *unit* |
| Transitivity | | + | + | > | o |
| Propagated Operations | | select, delete, ... | select, delete, ... | > | delete *unit* |

Notes: "+" defined; "-" to be further specified; "o" not existing; ">" inherited from a more generic relationship.

## 5. Integrating Relationship Semantics into ORDBs

With the exception of generalization, relationships with built-in semantics are not supported by current ORDMs. Schema designers or application programmers have to maintain the complicated semantics of various relationships discussed in Sect. 4. Therefore, the database modeling challenges posed by current advanced applications can not be well met. In order to remedy this deficit, we are developing a prototype system ORIENT (Object-based Relationship Integration ENvironmenT) to integrate relationship semantics into an ORDBMS. Our approach enables to specify relationship semantics explicitly, which are then mapped to basic mechanisms provided by the underlying DBMS. As a consequence, relationships are available as first-class modeling constructs, domain-specific logic can be caught and reflected more accurately in the conceptual modeling process, and the relationship semantics is no longer embedded, distributed, and replicated within the user-supplied code.

### 5.1 Data Structures

The core concepts of data structures, which we use to model and implement relationships, are types and type hierarchies. A type system introduces a formal framework defining integrity conditions for the extension of each type. Typing also allows to perform consistency checks which will give great benefits to the processing of operations on relationships. Moreover, a type hierarchy facilitates genericity and the incremental implementation of systems.

In ORIENT, *relationship* is defined as the most generic type which possesses several well-known relationships as default subtypes. Schema designers can select such a type with some built-in characteristics, then specify the exact semantics of the domain by introducing subtypes or instances thereof. Alternatively, they can define completely new types under *relationship,* if desired. An example of the resulting relationship hierarchy is illustrated in Figure 3.

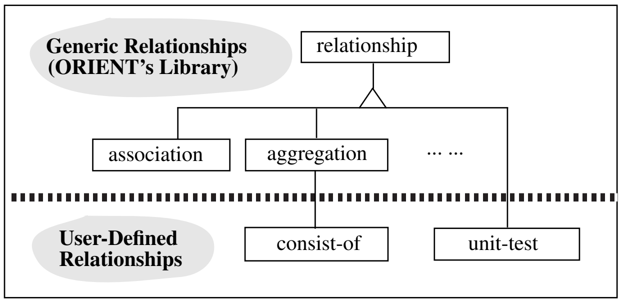


**Figure 3. Inheritance hierarchy of relationship types**

The upper layer contains a generic and extensible model of relationships that is implemented in an application-independent fashion, for example, the generic abstraction relationships *association* and *aggregation*. Generalization is not illustrated explicitly in Figure 3, because this concept is typically offered by the underlying DBMS without any options for tailoring its semantics. The layer below contains specialized relationships created by users, which include the specialization of abstraction relationships and the definition of relationships with application-specific meanings.

The information captured by the specification of relationship types includes: relationship name, degree, references to participants, semantic constraints such as cardinality and existence dependencies, etc. For example, relationship type *consist-of* should be described with the attributes shown in Figure 4.

```
relationship consist-of (
inherit-from:         aggregation;
degree:               4;
participants:         automobile, seat, door, window;
composition-owner:    automobile;
non-sharable:         seat, door, window;
cardinality:          1 : [2,8], [2,6], [4,8];
existence:            MDA OF automobile ON
                          seat, door, window;
                      MDP OF seat, door, window
                          ON automobile)
```

**Figure 4. Attributes specifying semantics**

Notice that most of the relationship semantics is by virtue some kind of integrity constraints and can be implemented relying on active mechanisms featured by the DBMS. Moreover, the rich modeling power of ORDMs provides the possibility to embed explicit constraints into data structures as procedural attachments. Thus, relationships can be implemented as database objects with enforcement methods that encode the active behavior of relationships. It is the task of ORIENT to transform declarative specification of relationship semantics into internal implementation of relationship objects.

***Type*** *consist-of*
*{**Participants***
*{Type automobile Owner;*
*Type seat* $P_1$*;*
*Type door* $P_2$*;*
*Type window* $P_3$*}*
***Monitor*** *( )*
*{if deletion of Owner then*
*Enforcement-Method*$_1$ *(Owner, Delete);*
*if deletion of* $P_1$ *then*
*Enforcement-Method*$_2$ *(*$P_1$*, Delete);*
*......}*
***Enforcement-Method***$_1$ *(Owner, Delete)*
*{delete Participants recursively}*
***Enforcement-Method***$_2$ *(*$P_1$*, Delete)*
*{delete Owner and other Participants recursively}*
*......}*

**Figure 5. Structure of a relationship object**

## 5.2 Relationship Objects

There is no distinction between a relationship object and a normal object except that the former has attributes and methods with relationship-specific meaning: a list of objects that may participate in the relationship, a method acting as a monitor, and associated semantic enforcement methods. The structure of relationship objects can be roughly illustrated as in Figure 5.

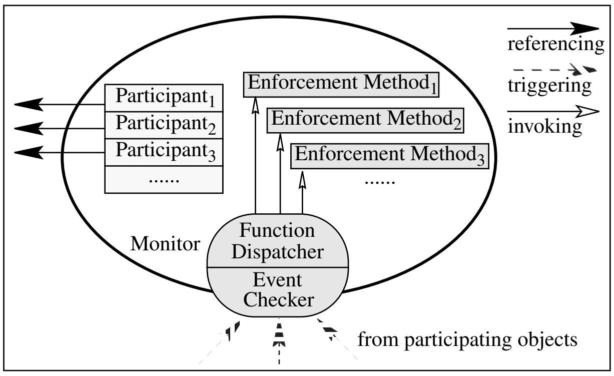


**Figure 6. Relationship object as event checker and function dispatcher**

The relationship semantics is implemented in such a way that it gives a relationship object the ability to react to the behavior of related objects and to take appropriate actions to maintain the integrity. These actions are encoded in enforcement methods, which make participating objects reject an operation that would violate integrity, enforce the propagation of an operation across relationships, etc. Enforcement methods are invoked by a monitor method based on two parameters: the name of participants and the operation which triggers the monitor. Thus, the monitor is a special method that plays a role as event checker and also function dispatcher as depicted in Figure 6.

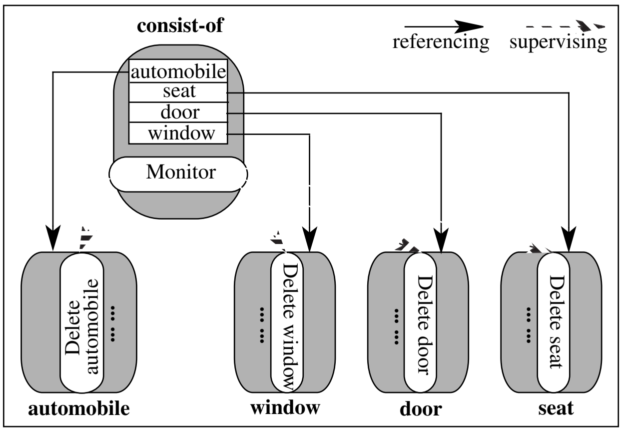


**Figure 7. Interconnection of relationship object and related objects**

The rationale of such an implementation is also determined by the inherent non-local nature of relationship semantics: the behavior of an object depends not only on the class to which it belongs but also on the objects to which it is related. As an example, take *consists-of*, which expresses a transitive and dependent relationship, so that deleting one participating object may cause the deletion of other associated ones. Therefore, relationship semantics should not be encapsulated within the boundaries of one single object, rather, relationship objects are needed to supervise the behavior of participants and to make appropriate responses. Figure 7 illustrates the interconnection between a relationship object and its related objects.

## 5.3 System Architecture

By facilitating a declarative specification as well as an automatic maintenance of relationship semantics, ORIENT can serve on one hand as a semantic front-end of an ORDBMS or OODBMS and on the other hand as a tool for database designers, which will facilitate the user interaction with databases and improve DBMS independence. The system architecture of ORIENT is shown in Figure 8.

Relationship specifications provide modeling constructs to aid the designers in expressing the relationship semantics in the conceptual schema. Internally these specifications are processed by a precompiler, whose main task is to identify the relationship types and to generate relationship objects which are compatible with the underlying DBMS. Various semantic aspects are specified by attributes of predefined types (cf., Figure 4). The most generic *relationship* type provides default methods to guarantee and, if necessary, to enforce relationship semantics. In this way, a framework of integrity maintenance is established. These default methods can be inherited and also adjusted to match the application-specific needs. In fact, method inheritance operates over the entire relationship hierarchy. The derivation of actual methods for enforcement and monitoring consists of the following steps:

- The related objects as well as the semantics of their relationships are identified with the help of attributes (e. g., *participants, composition-owner*, etc.).
- For every semantic aspect, all relevant operations should be determined, e. g., for *cardinality* the *insert* and *delete* operations must be supervised. For the transitive relationships, other operations such as *select* and *copy* must also be taken into account. Based on such an interplay of relationship properties, their related objects, and the affected operations, enforcement methods are generated.
- Finally, the monitor method responsible for the scheduling of these enforcement methods is constructed correspondingly. Thus, the relationship objects are automatically enriched by the required mechanisms.

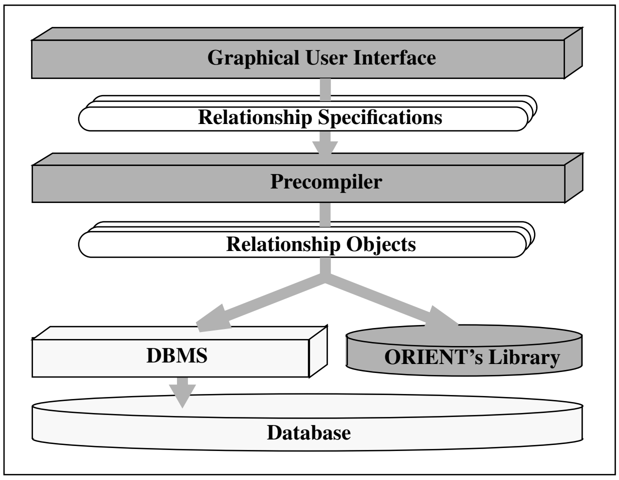


**Figure 8. System architecture of ORIENT**

It should be noticed that, although in the prototype implementation, we take the object-relational system Illustra [15] as the sample DBMS, our approach can be applied to enrich other DBMSs, including object-relational as well as object-oriented ones, with relationship semantics.

## 6. Related Work

The Entity-Relationship Model [6] is famous for including relationships as main primitives. Further developments in this area led to a broad spectrum of semantic data models [12, 22] providing a set of abstraction relationships. However, these models are rarely supported by a DBMS.

In the field of object orientation, [18] presents an algorithmic method and a tool for transforming a binary-relationship schema to an object-oriented one. Several authors [23, 9, 4, 21, 1] propose to extend OO models with explicit relationships. These attempts are compatible with the goal presented in this article. [23], as the first to address user-defined relationship constructs, has a strong impact on some proposals (e. g., [4]) which, like in [23], allow n-ary relationships to be defined over objects, but only with simple notions of integrity control such as cardinality constraints. The idea in [9], expressed in a knowledge-representation language based on frames, allows to define binary relationships between objects as well as several kinds of constraints on relationships. In [21], user-defined relationships and constraints are modeled via a static set of attribute objects. Moreover, [1] proposes a more comprehensive approach to model objects, relationships, and declarative constraints through the use of an object-oriented programming language. Several other approaches such as [17, 11] have been suggested to support aggregation relationships. Nevertheless, more general semantics for other types of relationships are not addressed.

Generally, complex semantics is expressed via constraints [26, 8]. The enforcement of integrity constraints is therefore of special concern for the implementation. Active rules are increasingly being incorporated in many commercial products and research prototypes. They are also fostered by the forthcoming and still evolving SQL3 standard [2]. Much of the research promotes triggers or database rules as a uniform mechanism to support various kinds of semantics that otherwise would have to be encoded in applications. The rule system in Illustra [15], for example, can be used to handle a variety of situations, including traditional integrity constraints, referential integrity, view management, access control and audits. As to relationship-based constraints, [10] focuses on describing the update rules for user-defined relationships. However, relationships are not supported as first-class objects, and only binary relationships are considered.

To avoid direct specification of update rules, some authors [27, 3, 5] advocate the automatic generation from declarative constraints. In particular, for relationship semantics, [24] addresses the control of operation propagation by means of propagation attributes. And [16] presents an approach for integrating inter-object constraint maintenance into an OODBMS. Relationships are not used as an explicit construct; instead, the semantics is encoded locally within the involved objects, which is exactly the opposite of our approach.

It should be noticed that most of the previous work put only attention to propagation in the context of *insert*, *delete*, and *update* operations. Since date relationships may also have a side effect on other database operations such as *select*, we believe that the general problem of specifying and maintaining relationship semantics deserves more detailed attention.

## 7. Conclusion

This paper has primarily been focused on the importance of data relationships in database application modeling and on investigating the semantics of data relationships. To remedy the deficits of the emerging ORDB technologies, which lack adequate support in this respect, we presented an approach that augments ORDMs with refined relationship facilities. A prototype system was described, whose generic relationship model can be adapted to the specific modeling needs of different application domains, particularly those which require support for complex data relationships.

This endeavor incorporates several important aspects such as the automatic generation of enforcement methods and a query mechanism that fully supports relationship semantics. In addition, a user-friendly interface is indispensable to provide system-supplied menus to tackle the complexity of the specification and to convey to the system as much application-specific semantics as possible.

The project RITA being carried out in corporation with an industrial partner presents a suitable setting to analyze user requirements as well as the diversity of relationships in real-world applications. Moreover, this environment provides a realistic test-bed both for the semantics specification and for the ORIENT system implementation. We will report the benefit as well as our experience of using the facilities offered by ORIENT in a future paper.

## References


[1] A. Albano, G. Ghelli, R. Orsini, "A Relationship Mechanism for a Strongly Typed Object-Oriented Database Programming Language", Proc. 17th VLDB Conf., Barcelona, Sept. 1991, pp. 565-575.

[2] ANSI X3H2, "Database Language SQL — Part 2: SQL/ Foundation, Committee Draft", July 1996.

[3] M. Bouzeghoub, E. Métais, "Semantic Modeling of Object Oriented Databases", Proc. 17th VLDB Conf., Barcelona, Sept. 1991, pp. 3-14.

[4] S. E. Bratsberg, "FOOD: Supporting Explicit Relations in a Fully Object-Oriented Database", in: Object-Oriented Databases: Analysis, Design and Construction, R. A. Meersman, W. Kent, S. Khosla (eds.), North-Holland, 1991, pp. 123-140.

[5] S. Ceri, P. Fraternali, S. Paraboschi, L. Tanca, "Automatic Generation of Production rules for Integrity Maintenance", ACM TODS 19:3, 1994, pp. 367-422.

[6] P. P. Chen, "The entity-relationship model — Towards a unified view of data", ACM TODS 1:1, 1976, pp. 9-36.

[7] C. J. Date, An Introduction to Database Systems, 6th Edition, Addison-Wesley Publ. Comp., 1995.

[8] S. Deßloch, Semantic Integrity in Advanced Database Management Systems, Ph. D. Thesis, Dept. of Computer Science, University of Kaiserslautern, Aug. 1993.

[9] O. Díaz, P. M. D. Gray, "Semantic-rich User-defined Relationships as a Main Constructor in Object Oriented Database", in: Object-Oriented Databases: Analysis, Design and Construction, R. A. Meersman, W. Kent, S. Khosla (eds.), North-Holland, 1991, pp. 207-224.

[10] O. Díaz, "The operational semantics of user-defined relationships in object oriented database systems", Data & Knowledge Engineering 16 (1995), pp. 223-240.

[11] M. Halper, J. Geller, Y. Perl, W. Klas, "Integrating a Part Relationship into an Open OODB System using Metaclasses", in: N. Adam, B. Bhargava, Y. Yesha (eds.), Proc. 3rd Int. Conf. on Information and Knowledge Management (CIKM-94), Gaithersburg, Maryland, Nov. 1994, pp. 10-17.

[12] R. Hull, R. King, "Semantic Database Modeling: Survey, Applications, and Research Issues", ACM Computing Surveys 19:3, Sept. 1987, pp. 201-260.

[13] T. Härder, J. Thomas, "RITA - ein rechnergestüztes Informationssystem für technische Anwendungen" (in German), ITG-Fachbericht 137 (STAK'96), Munich, March 1996, pp. 111-126.

[14] IBM DB2 SQL Reference - for common servers (Version 2), IBM Corp., 1995.

[15] Illustra User's Guide (Release 3.2), Illustra Information Technologies, Inc., 1995.

[16] H. V. Jagadish, X. Qian, "Integrity Maintenance in an Object-Oriented Database", Proc. 18th VLDB, Aug. 1992.

[17] W. Kim, E. Bertino, J. F. Garza, "Composite Object Revisited", Proc. 1989 ACM SIGMOD Conf., Portland, 1989, pp. 337-347.

[18] Y. Kornatzky, P. Shoval, "Conceptual design of object-oriented database schemas using the binary-relationship model", Data & Knowledge Engineering, 14 (1994), pp. 265-288.

[19] N. M. Mattos, An Approach to Knowledge Base Management, LNCS 513, Springer-Verlag, 1991.

[20] F. Manola (eds.), Object Model Features Matrix, ANSI X3H7-93-007v10, Feb. 1995.

[21] R. Nassif, Y. Qiu, J. Zhu, "Extending the Object-Oriented Paradigm to Support Relationships and Constraints", in: Object-Oriented Databases: Analysis, Design and Construction, R. A. Meersman, W. Kent, S. Khosla (eds.), North-Holland, 1991, pp. 305-329.

[22] J. Peckham, F. Maryanski, "Semantic Data Model", ACM Computing Surveys 20:3, Sept. 1988, pp. 153-189.

[23] J. Rumbaugh, "Relations as Semantic Constructs in an Object-Oriented Language", OOPSLA'87, pp. 466-481.

[24] J. Rumbaugh, "Controlling Propagation of Operations using Attributes on Relations", OOPSLA'88, pp. 285-296.

[25] M. Stonebraker, Object-Relational DBMSs - The Next Great Wave, Morgen Kaufmann Publ., Inc., 1996.

[26] B. Thalheim, "Semantical Constraints for Database Models", in: Advances in Database Systems - Implementations and Applications, J. Paredaens, L. Tenenbaum (eds.), CISM 347, Springer-Verlag, 1994.

[27] S. D. Urban, L. M. L. Delcambre, "Constraint Analysis: A Design Process for Specifying Operations on Objects", IEEE TKDE 2: 4, Dec. 1990, pp. 391-400.

[28] J. D. Ullman, Principles of Database and Knowledge-Base Systems (Vol. I), Computer Science Press, Rockville, Maryland, 1988.

[29] UniSQL Server User's Guide, UniSQL, Inc., 1996.