In: Proc. of the First East-European Symposium on Advances in Databases and Information Systems (ADBIS'97), St. Petersburg, Sept. 1997.

On Modeling Power of Object-Relational Data Models in Technical Applications

Nan Zhang, Theo Härder
Department of Computer Science, University of Kaiserslautern
67653 Kaiserslautern, Germany
E-mail: {zhang | haerder}@informatik.uni-kl.de

Abstract

Technical applications, that is, all kinds of engineering applications (CA*), are distinguished from usual business applications in that they require enhanced data modeling facilities and operations to cope with complex objects and their behavior. Emerging object-relational database management systems (ORDBMSs) which are conforming to the SQL3 standard promise to support technical applications more adequately as compared to conventional relational DBMS (RDBMSs). They offer concepts such as complex object support (that is, reference-based relationships), an extensible type system including set-valued attributes, DB-based abstract data types (ADTs), and more. However, modeling and processing support of complex objects and their versions, large objects, semantic-rich relationships, etc. is only rudimentary or even missing in current ORDBMSs.

Moreover, their architecture, functionality, and implementation are server-centric, that is, all DBMS functions are confined to the server. Technical applications, however, typically run in client/server environments where operation performance enabling interactive design work has to be provided. As a consequence, database management services have to be specialized for these performance requirements and, therefore, have to be dispersed across the server and the client side. Hence, used as database server technology, ORDBMSs have to be complemented by adequate client-side data management and long-running design transactions encapsulating the client processing model, in order to provide satisfactory support for technical applications.

This paper investigates the potential as well as the shortcomings of object-relational database technology for technical applications. Our observations are based on practical experiences with developing an integrated information system in a technical application domain. In this respect, we will evaluate the advantages and disadvantages of the enhanced ORDBMS facilities and identify areas which deserve refinements and improvements in future evolutions of both the standard and the systems.

1. Introduction

ORDBMSs [2, 13, 22] embody an upcoming and promising database technology and therefore attract more and more attention from both the research and the commercial realms. How can database applications benefit from the exploitation of this technology? What is still missing for the new development trend to really meet the expectations of the real world? In this paper, we will try to answer these questions with the help of our experiences gathered from an ongoing project as well as our observations of the current developments in this area.

The goal of our project RITA [8] is to realize an integrated information system in a technical domain. This project is carried out in cooperation with a leading manufacturer of automobile seats. Currently, we are implementing a subsystem supporting the design and test of prototypical technical objects. Besides workflow management and client/sever computing, a fundamental ingredient of the system is information management which requires effective and efficient database services. Typical characteristics w.r.t. the information management in RITA can be summarized as below:

- The amount of data to be managed is very large.
- Besides standard data types, there is also a variety of complex data types, including long texts of requirements and reports, sketches of test plans, photos and videos of test progresses, etc.
- The object structures are diverse, ranging from simple flat structures (e. g., part catalogs) to heterogeneous compound structures (e. g., CAD objects).
- Ad-hoc querying of the stored data is indispensable.
- Concurrent users will share the data in a distributed and heterogeneous environment.
- Data security and availability must be guaranteed in order to provide commercial-strength data management functions for transaction processing, workflow support, as well as decision making.

The potential of RDBMSs for implementing information systems has been extensively explored. However, advanced application requirements such as handling complex data types go beyond what RDBMSs can offer. Object-oriented database management systems (OODBMSs) support a range of additional features that appear to meet some of these requirements [18]. But they are proven to perform poorly w.r.t. some key DBMS properties and services such as integrity control, backup, and recovery. For these reasons, we have chosen the ORDBMS Illustra [10] for our prototype system, since it claims to possess the following capabilities:

- It enables the user to define ADTs, composite and constructed data types.
- It allows the user to define operations on ADTs, composite and constructed data types.
- It contains a rule system supporting database consistency and facilitating the implementation of diverse application semantics with a uniform syntax.
- It offers a contents-based query language (SQL) and traditional DBMS facilities.
- It provides pre-built or user-developed DataBlade Modules based on ADT extensions which enable to store and manage advanced data types (e. g., multimedia data, spatial data, HTML, and so on) as well as to tailor and extend the DBMS for specific application domains.

These features characterize a new trend in database research and development: Recently, almost all top database vendors [9, 11, 20] have redirected their strategies to object-relational and are extending their database server architectures accordingly. The overall goal is to allow users to effectively model and manipulate unconventional data and to use object-oriented technologies for application development without losing the benefits of SQL and all the necessary features of a commercial-strength DBMS. Therefore, these efforts embody a promising potential to build advanced information systems satisfying more complicated requirements on how data are used and managed. However, being in their infancy, current ORDBMSs cannot yet meet all the objectives and challenges.

In this paper, we will primarily evaluate the modeling power of object-relational data models (ORDMs) in the setting of advanced technical applications. To this end, we resort in Sect. 2 to the RITA project which delivers the modeling example. In Sect. 3 and Sect. 4, we analyze the strengths as well as the weaknesses of current ORDMs, respectively. Other processing-related issues, including Persistent Stored Modules (PSMs) and client-side data management, are considered in Sect. 5. Thereafter, Sect. 6 summarizes the state-of-the-art of the object-relational technology and gives an outlook on future developments. Finally, we conclude the article with Sect. 7.

2. Representative Application View

In the following, we sketch a part of data structures and relationships extracted from the RITA project to indicate the spectrum of modeling requirements of technical applications. This simplified view serves to derive adequate data modeling constructs and, later on, processing requirements which can be used to explore the suitability of ORDBMS facilities in our context.

For this purpose, we introduce at first the graphical notations in Fig. 1, where the entity box contains the entity name. Attributes as well as operations are ignored for the sake of brevity. Besides abstraction relationships [17], we identify also application-specific relationships, which are shown as lines between entity boxes. 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.

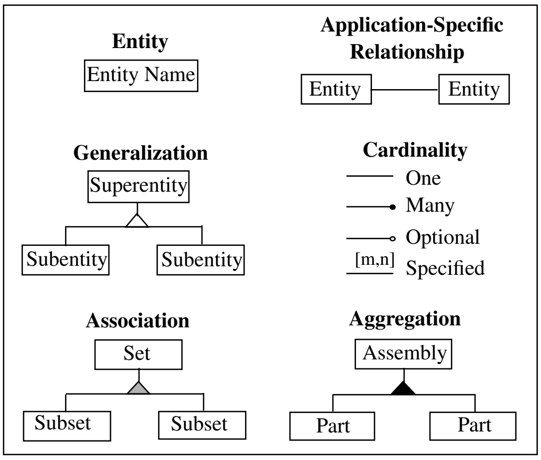

Fig. 1: Basic notations

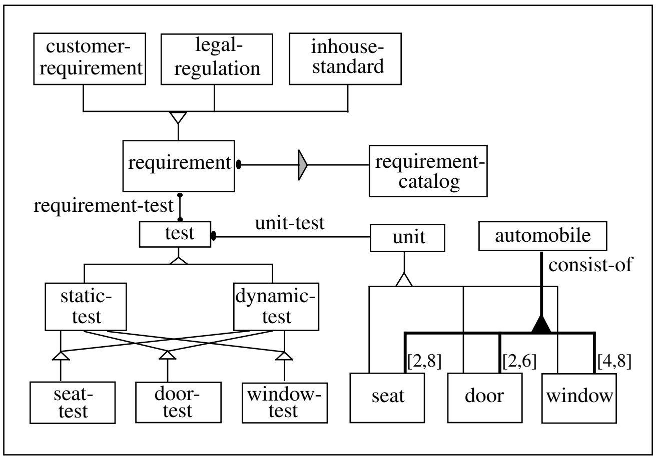

Fig. 2: A part of data model

Fig. 2 illustrates a representative view of the entities and their relationships in RITA with the explanation of some entities in Tab. 1. In this 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*. And an instance of *requirement-catalog* for an automobile unit is represented as a set of instances of *requirement*. Our view of the application assumes automobiles to consist of seat(s), door(s), and window(s). Between them there exists an aggregation relationship *consist-of*, where *automobile* is the aggregate object with *seat*, *door*, and *window* as its parts. Moreover, between *test* and *requirement* there is an (n:m)-relationship *requirement-test*, and between *test* and *unit* there is a (1:m)-relationship *unit-test*.

Tab. 1: Data dictionary

Entity	Definition
automobile	consists of several parts; for the time being, we consider only three of them: 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.

To facilitate our discussion on the modeling power of ORDMs, a small part of schema definitions are given in Fig. 3 where some of the abstract entities and relationships of our example are translated into DDL statements using Illustra SQL [10].

```
create function n_in (text) returns name_t
  as external name ' name.so' language C;
create function n_out (name_t) returns text
  as external name 'name.so' language C;
create type name_t
 (internallength = 24,
  input = n_in,
  output = n_out);
create type description_t
 (header varchar (50),
  length integer,
  bodies setof(text));

create table unit of new type unit_t
 (unit_name name_t,
  designer varchar(20),
  model_builders setof(varchar(20)),
  has_tests setof(test_t),
  ... ...);
create table requirement
   of new type requirement_t
  (requirement_name name_t,
   description description_t,
   has_tests setof(ref(test_t)),
   ... ...);

create table test of new type test_t
 (test_name name_t,
  test_unit ref(unit_t),
  for_requirements setof(requirement_t),
  test_date date,
  test_pattern array(array(name_t)),
  sketch image,
  ... ...);
create table dynamic_test under test
 (acceleration string,
  amplitude_progression string,
  ... ...);
```

Fig. 3: A part of schema definitions

3. Benefits of Using ORDMs

How can the rich modeling constructs and the corresponding query facilities of ORDMs improve schema definition and query specification? In the following, we refer to the schema definitions of a part of our RITA application in Fig. 3 to outline and illustrate the use of the most important ORDM concepts.

3.1 Abstract data types

ADTs enable the set of built-in types of the DBMS to be extended with new data types and type-specific behavior. Assume, the names identifying some entities (requirements, units, etc.) have a special composition. ADT *name_t* can reflect this fact more accurately than the built-in type *text*. Moreover, *image* provided by the Image DataBlade is used to describe the sketch of a test plan.

Correspondingly, a query can be enriched with method invocations in search conditions. Thus, it is possible to shift a considerable amount of processing into the method code and to avoid complete specification of computations in SQL, thereby making the query clean-looking. For example:

Query: *Find all automobiles whose seats have a total weight less than 300kg.*

SQL:
```
SELECT *
FROM automobile a
WHERE weight_of_all_seats (a) < 300
```

3.2 Table extensions and composite types

A conventional table is extended by allowing reference types (*ref,* see below), ADTs, and alphanumeric types as column domains. Users can attach procedures to a table and have the procedures operate on the column values. Moreover, composite types (row types in SQL3 [12]) make it possible for tables to enjoy object properties. Together with reference types, composite types can be employed to model composite objects (e. g., a whole automobile with a collection of integral units). These structures lead to the regular use of path expressions:

Query: *Find all requirements whose description has a length of more than 10000 words.*

SQL:
```
SELECT *
FROM requirement
WHERE description.length > 10000
```

3.3 Reference types

Reference types, e. g., *ref(unit_t),* represent inter-object connections in a direct way and make queries through reference paths more compact than through *JOINs* based on primary-key/ foreign-key pairs. Another advantage of implicit joins is that they usually offer more optimization potential than *JOINs* explicitly specified by the user. Moreover, queries with *ref/deref* (dereferencing) can be issued over a type and may automatically affect many different tables. Suppose there is a separate *unit* table defined for each testing project:

Query: *Find all tests that are carried out on the units designed by Smith.*

SQL:
```
SELECT *
FROM test t
WHERE deref(t.test_unit).designer = 'Smith'
```

While *JOIN* or *UNION* operations require an explicit list of all the tables involved, *ref/deref* is preferable in situations where several involved tables are of the same type.

3.4 Inheritance

Inheritance (single as well as multiple), which enables natural variations among types/tables, is supported by organizing types/tables into hierarchies. For example, each single test category can be modeled as a subtable of *test*, and a query can be issued against a set of tables in the hierarchy:

Query: *Find all tests that have been performed on March 12, 1997.*

SQL:
```
SELECT *
FROM test
WHERE test_date = '1997-03-12'
```

Such a query searching the *test* table and all its subtables will be much more complicated if it is issued against a relational schema with separate tables for every kind of tests, which must be combined together with explicit multi-way *UNION* operations. Furthermore, if a *JOIN* has to be performed among two table hierarchies, the complexity will strongly increase.

This impediment can be somewhat eliminated when a *VIEW* definition is done based on *UNION* operations. In this case, to control the base table where a new tuple is placed, the tuple can usually not be inserted through the *VIEW*, instead it should be directly inserted into the base table. Even though such inconvenience could be accepted, supporting updates through *VIEWs* defined with *UNION* queries remains a problem which has not been well solved yet (also in ORDBMSs).

3.5 Constructed types

Multi-valued attributes constructed as set, list, or array make data modeling and handling much easier. For example, we can define *test_pattern* as a matrix using the constructed type *array(array(name_t))*, and the relationship between *test* and *unit* can be specified through the attribute *has_tests* with constructed type *setof(test_t))* and the attribute *test_unit* with reference type *ref(unit_t)*. These typed specifications obviously provide a much clearer understanding than storing the set members in a separate table having an attribute as a foreign key referencing the parent table. A query involving set-valued attributes may look like this:

Query: *Find all the names of tests carried out on units with Smith as one of the model builders.*

SQL: *SELECT test_name*
FROM test t
WHERE 'Smith' IN deref(t.test_unit).model_builders;

Although this query requires the membership test of "Smith" in the set *model_builders*, it is more compact than using *JOINs* when embedded set-valued attributes must be simulated through the access of separate tables.

So far, we have only illustrated simple examples. It is well known that data structures and processing requirements in real applications are much more sophisticated. Nevertheless, ORDMs will reveal more strengths in regard of clean schema definitions and concise query specifications when combining the above features skillfully.

4. Recognized Modeling Problems

Although we have gained much progress concerning adequate data modeling for (some of the typical requirements in) technical applications, several problems call for more satisfactory solutions. The drawbacks of current ORDMs that we have recognized especially involve the following aspects:

4.1 Complex objects

Complex objects, a prime requirement in technical applications, embody highly structured objects aggregated from heterogeneous sets of elementary objects, e. g., an *automobile* in Fig. 2. They can be modeled and constructed either by ADTs or by *ref* types (together with composite types). Hence, an entire spectrum of design choices is available ranging from more static complex object definitions by ADTs to more dynamic complex object definitions or derivations using the *ref*/row type facilities [5].

Using ADTs will result in a lack of "(de-)composability": A minimum requirement for ADT composition, that is, for building complex objects from less complex ADT structures, would be ADT sharing by using pointers. However, since a reference concept for ADTs has not been invented so far, the need of object sharing may provoke high degrees of replication and severe integrity problems.

On the other hand, extensive use of *ref* types and individually stored objects provides the highest degree of freedom for complex object construction. The complex object structures must not be fixed at database creation time, but may be specified in the referencing queries (cf., MQL [19]) and evaluated at run-time. This approach also guarantees flexible view processing for complex objects. However, it leads to the abundant use of path expressions to traverse structures. Optimization of path queries has not yet achieved satisfactory results due to the difficulty of index definition and the presence of arbitrary methods. Moreover, a path expression implies an execution order; this order may, however, not be the most efficient way to process the query. For these reasons, it is sometimes a burden for schema designers or application developers to choose an optimal solution. As pointers must be followed to assemble a composite object, sophisticated clustering is ideal to avoid extra disk accesses in this context, but clustering concepts are still not a salient feature of ORDBMSs. This can be explained by the fact that, if subobjects are to be shared, it is impossible to determine "generally optimal" object clusters.

4.2 Versioning

In order to support the exact and complete preparation of future design projects, RITA will be employed to maintain and provide information about earlier projects and test suites. Moreover, relevant data on new projects should be steadily integrated into the already existing information repository. Therefore, modeling and managing historical data are important issues in our setting.

Even further, capturing the design history of objects is a prime requirement in technical information systems. Hence, versions of complex objects have to be stored and managed by the DBMS. Depending on the version model to be supported, the creation or derivation history of complex objects has to be represented by sequences, trees, or even nets of object versions [13, 16]. A set of operations on object versions and their derivation graphs has to be made available for the design applications. With the help of the extension mechanisms of ORDMs, application-specific version concepts could be provided in a concise and elegant way. In addition, this version support can be used to further supply configuration mechanisms in order to compose more complex structures (e. g., aggregate objects or products) from versioned objects in a tailored fashion [21].

4.3 Set-valued attributes

Genuine set-valued attributes are not supported. Instead, they are “internally simulated” using separate tables which must be joined implicitly if a set-valued attribute is retrieved, and a cursor mechanism must be used to fetch set members one-at-a-time until the query result is exhausted. Furthermore, typical set operations such as *INTERSECT* and *UNION* are not provided. All these concepts are just like a “reprint” from RDBMSs and incur a lot of troubles in our application modeling and implementation.

We have noted that Illustra’s extensibility to allow user-defined data types and user-defined functions to be supported by the database server is achieved through its table-driven mechanisms. All new-added features are mapped to internal table constructs. Therefore, Illustra is actually an extended RDBMS and this relation-centric view causes also some other problems.

4.4 Object caching and navigation

Another drawback is due to the RDBMS computational model which falls short in supporting navigational access to memory-resident objects typically cached in a client-side object buffer (workspace). Technical applications must deal with large volumes of complex and tightly meshed data (embodying many (n:m)-relationships) like CAD objects. In OODBMS, such data are pre-loaded and cached in client-side main memory in order to meet the performance requirements of interactive design work. In addition, pointers are swizzled to speed up processing. Illustra, however, does not offer a mechanism like a function call for an OID-based object fetch in application programming interfaces (APIs). Therefore, it cannot directly provide such support of object caching and navigation.

In general, client-based processing of persistent objects, including ADTs and references, leaves today many questions open, since client-side data management is not supported by ORDBMS processing concepts [6]. A desirable solution for this aspect would require an adequate client-side processing model, e. g. enhanced transactions using checkout/checkin [15], and would have to include accumulation of database updates until commit, descriptive object buffer management using contexts, main-memory query evaluation based on the contents of client and server buffers, consistency maintenance, schema evolution, etc. Due to space restrictions, we cannot further detail all these problems.

4.5 Large objects

Our observations on the support of large objects (LOBs, BLOBs, or CLOBs) in ORDBMSs show that the implementations usually concentrate on low-level issues such as space management and layout of objects on storage media. The entire object is stored in an unstructured "long field", in which exact representation and manipulation of the object are entirely left under user control. The DBMS does not model the contents of the long field, so queries cannot apply functions to its contents. For example, in Illustra, large objects are the kernel of its object-relational features since ADTs and other data types based on ADTs (e. g., *image*) are all internally implemented through large objects as an extension of the relational model. However, due to the described deficits, most of the operations must be defined explicitly by application developers. Manipulation of an image can only be performed on the entire image file but not on an arbitrary single pixel.

To make large objects much more useful, the current drawbacks must be removed to a great extent. Generally, in addition to the facility of manipulating LOB segments with deferred materialization, LOB support should break through the "black-box" limitation and be more "structured".

4.6 Data relationships

Our example also indicated that the restricted support of data relationships in ORDMs may lead to unnatural modeling. With the exception of generalization, relationships with built-in semantics (e. g., association and aggregation) are not supported by current ORDMs, although technical applications can frequently take advantage of them. They have to be modeled as a (non-transitive) set of references to the instances of other tables, and there do not exist system-controlled operations for these relationships. As a result, semantics must be completely maintained in the user code.

5. Processing-Related Issues

So far, we have outlined the essential data modeling concepts of ORDM and, in turn, we have analyzed the still existing weaknesses as far as modeling expressiveness is concerned when technical objects have to be represented in an accurate and adequate way. In this subsection, we want to give a few words on important processing issues related to the object-relational database technology. They are PSM suggested in SQL3 [12] and client-side data management, which have great impact on application development and, therefore, should also be taken into consideration when discussing the power of ORDMs.

Our discussion is focused on the client/server environments, since the specific nature of technical applications leads to a workstation-based hardware architecture with decentralized and autonomous processors. Each processor equipped with sufficient "horse power", main memory and private disk space serves a fixed number of users and provides special functions for the application (e. g., graphics), if necessary. Coupled via LAN, the workstations have access to a central server complex (consisting of one or multiple processors) which hosts the database.

From a hardware point of view, extensibility of the application system can be achieved by attaching additional workstations to the server complex. However, with an increasing number of new nodes added to the system, bottlenecks in the central database server and the LAN will occur and degrade the system performance. The closer the coupling between server and workstation (client) is, the more they will mutually influence one another (communication delays, various kinds of failures). On one hand, the use of workstations is the key to solve many aspects of scalability (available “horse power”, number of users, etc.); on the other hand, it requires a careful architectural design to overcome the illustrated problems. The only reasonable approach seems to be the use of a loosely coupled system of components which have only a limited need of database access and communication.

5.1 Persistent Stored Modules

Client-side data management relies on efficient data supply (checkout) and propagation of accumulated updates at commit (checkin). A building block for enabling this support in a navigational way is the PSM mechanism. PSM defines procedural extensions in SQL3 to achieve computational completeness. As a result, SQL is enriched with many new control statements such as begin/end block, assignment, call, if, case, loop, local variable, signal/resignal, and exception handling. Moreover, multiple SQL statements can be included in a single EXEC SQL statement. Obviously, they will largely increase the power and expressiveness of SQL.

Currently some commercial systems are trying to realize this part of the SQL3 proposal in their database server. It is not an easy task to implement the procedural extensions without sacrificing the power of existing query optimization mechanisms in the database sever. Nevertheless, it is worth the trouble to do so because of various advantages that the PSM technology brings about:

- Firstly, readability and portability of application programs are augmented, since database access operations can be written with a unified language and be packaged into a single SQL statement.
- Secondly, by moving the realization of some program logic to the database server, some DB-related application functions can be executed in close connection to the server, thereby significantly enhancing performance.
- Thirdly, stored procedures, user defined functions, and triggers can be more efficiently implemented due to the possibility of global optimization and the abandonment of binding operations with host variables.
- Fourthly, based on the PSM technology, large amounts of elementary objects can be selected from the database and assembled to complex objects. Using suitable transfer granules, they can be loaded to a client-side object buffer with minimum communication overhead. This method, called data shipping, is indispensable in client/server architectures to achieve efficient data supply for technical application processing.

The PSM mechanism primarily provides enhanced data management support at the server side. Together with the ORDM concepts, it greatly improves the realization of comparably simple functions which can be directly processed by the database server (query shipping). More sophisticated object manipulation in technical applications, however, requires data shipping, data caching, and data handling at the client side.

5.2 Client-side data management

A large share of DB-based technical applications manipulate complex objects and their versions. Predefined application functions (often called tools), which allow the application to abstract from the object details, are manipulating these complex objects, thereby repeatedly accessing them in varying reference patterns. Therefore, in contrast to typical business applications using query shipping to the server, such technical applications necessarily require the data shipping approach consisting of three phases: load, operate, and merge [1, 4]. Hence, they rely on adequate data management and a tailored processing model at the client side to achieve their performance goals. For example, geometric modeling in 3D-CAD refers to solids consisting of large numbers of elementary objects (10^4 and more). To evaluate application functions (cuts or perspective views of solids, etc.) more than 10^5 data references/sec are necessary to guarantee tolerable response times. Therefore, such an approach has to include

- long-duration caching of large data volumes in a client object buffer;
- local data manipulations and integrity checks;
- perfect exploitation of reference locality "near by the application" to achieve "interactive response times";
- fast computation of local and ad-hoc object transformations;
- direct visualization of the "object under design" at the workstation screen; and
- transaction control adjusted to the needs of long-running technical applications.

Furthermore, to achieve high-performance complex object manipulation and visualization, special main memory representation and addressing of the objects to be processed are absolutely necessary.

Although we have experienced broad and active database research in this area during the last 15 years, the data management problems are still tough, and we have made, in fact, only little progress. An ORDBMS (and SQL3 [12] as the standard) is in its processing philosophy purely server-centric, that is, this future DBMS standard does not consider client-side data management in its current perspectives. Hence, their enhanced and new built-in functionality in form of database procedures, PSMs, ADTs with user-defined functions, pre-built or user-developed modules such as DataBlades, Snap-ins, or Data Extenders [3, 5], etc. is confined to the DBMS kernel running at the server side. As a consequence, such approaches lead to "fat servers and thin clients" with processing concepts which are inapplicable for engineering design environments.

On the other hand, an ORDBMS offers extensibility paths to enhance and tailor its functionality and, hopefully, to adjust the processing location of functions to the needs of the application. The challenging question is now, how the new ORDBMS concepts can be made available for technical applications at the client side. Such an extensibility technology may work for encapsulated function execution in the DBMS server, but is not applicable in a straightforward way to achieve, e. g., version handling or workspace management at the client side. These issues are further explored in [7].

6. Outlook on ORDBMSs

In this section, we briefly summarize the state-of-the-art of the object-relational database technology and give our outlook on future developments.

6.1 State-of-the-art

As we have seen, object-relational or relational database vendors are demonstrating that the new database technology can be implemented, but they take significantly different ways to do so. For example, as the first ones in this area, Illustra [10] advocates an extended relational approach, whereas UniSQL [23] is actually an OODBMS that happens to support the relational model as a special case, that is, in addition to the object-oriented concepts it provides relational access capabilities. Like most of the OODBMSs, UniSQL supports an abundance of APIs, including C, C++, Smalltalk, ODBC, and CLI interfaces. This is made possible through UniSQL's navigational computational model attributed by workspace management and pointer swizzling techniques, which are missing in RDBMSs. However, UniSQL's deficit in supporting schema-level type extensibility makes data modeling more unwieldy, especially in the presence of multimedia data. This is the main reason why we changed to Illustra after some attempts with UniSQL.

At the time being, there is no generally accepted definition what "object-relational" means. The international standard committee has been showing its effort and intent when defining SQL3 [12] with object-relational features. SQL3 could serve as a "limited" future standard and is more or less closely followed by most of the database products [9, 10, 20]. Nevertheless, it is not available today and, in addition, it is also very relation-centric despite its recent adoption of some suggestion from UniSQL, e. g., *row types* as a comparable feature to UniSQL's identifying object attributes with separate table columns.

6.2 Our expectations

What should an ORDBMS really be? To us, ORDBMSs, trying to unify the relational database (RDB) technology and object-oriented technology, are expected to preserve the merits of both worlds and also to provide some added values as illustrated in Fig. 4.

That means, firstly, it should retain the value-based data representation which enables set-oriented queries, system-based optimization, and parallel query evaluation. Moreover, all salient DBMS properties, approved in mission-critical environments, must be preserved. At the same time, it should adopt some of the attractive data model features from the object-oriented world as well as support reference-based navigation and APIs compatible with ODMG or other object-oriented models. However, the fact that relational and object-oriented paradigms are somewhat antagonistic to each other poses more challenges on the unifying work. As an example, taking Illustra, the query and programming language statements cannot be freely combined: It is impossible to pose a query on the result returned by a set-valued function in the program or to pass a query to a function expecting a set-valued parameter. In UniSQL, although the query result can be placed in variables and be used in other queries or method calls, the result must be exactly one instance. ODMG OQL's ability to name query statements and use them in other queries could be very useful, i. e., to name the extent of a maybe completely new data type.

Secondly, some features orthogonal to relational or object-oriented ones, including semantic relationships, more mature version concepts, active mechanisms, as well as set-valued attributes, are also essential for advanced applications. Together with a framework of nested transactions and enhanced support of client/server computing, they should be further addressed in the context of ORDBMSs.

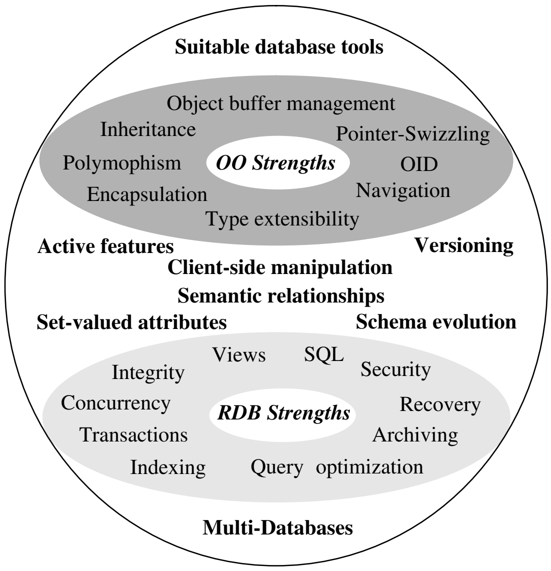

Fig. 4: Makeup of ORDBMSs

Thirdly, being confronted with such a richness in modeling constructs and functionality, users should have the possibility to deal with data from different perspectives and to migrate data between relational and object-oriented worlds without having to worry about implementation details. This will be impossible if there is no binding defined between ADT extensions (or row types) and object types (or classes) in the object-oriented world as in the current SQL3 and some products like Illustra. Schema evolution becomes also a much more important issue. For reasons of modeling accuracy as well as performance, it should be possible to alter a database schema in a flexible way during the lifetime of the underlying databases. Consequently, a database could dynamically migrate to more "relational" or to more "object-oriented", depending on the application demands. Furthermore, new database tools for DBAs, schema designers, and application developers are also needed for ORDBMSs.

7. Conclusion

Our discussion in this article was based on the prototypical modeling performed so far and on our current understanding of the critical issues. As the project progresses, we will surely gain even deeper insight into the modeling power of ORDMs. At present, we hold the opinion, according to our experiences using an ORDBMS to build an integrated information system for technical applications, that object-relational is the right way to go. In most cases, as we have seen, the object-relational extensions are powerful enough to result in both DDL and DML statements that are succinct, natural, and easy to read. However, there are still some cases that ORDBMSs, in their current status, can not handle ingeniously. As for processing strategies, server-side data management based on query shipping approach is greatly improved, while set-oriented data shipping with descriptive client-side object manipulation remains a challenging task for the future. The ultimate goal of ORDBMS extensibility should include the support of tailored processing models, thereby giving the system designer the opportunity to freely choose the location of a function execution.

Shortcomings of today's ORDMs need to be overcome before the full promise of this technology can be enjoyed. There exist definitely many difficulties to do so. However, it is not unappreciated for real applications to dream of the ultimate goal and it is not impossible for researchers and vendors to try to approach this goal, as long as the endeavor of definition is made more a matter of technology than of marketing, which is just the opposite as it is the case today.

Our work to remedy some deficits of ORDMs can be reflected by an ongoing project ORIENT [24] aiming at integrating relationship semantics into an ORDBMS and another project being planned to enhance ORDBMSs with mature version concepts.

Bibliography

[1] Bancilhon, F., Kim, W., Korth, H. F.: *A Model of CAD Transactions*, Proc. 11th Int. Conf. on Very Large Databases (VLDB'85), Stockholm, 1985, 25-33.

[2] Chamberlin, D.: *Using the New DB2: IBM's Object-Relational Database System*, Morgan Kaufmann, 1996.

[3] Cheng, J. M., Mattos, N. M., Chamberlin, D. D., DeMichiel, L. G.: *Extending relational database technology for new applications*, IBM Systems Journal 33:2, 1994, 264-279.

[4] Härder, T., Hübel, C., Meyer-Wegener, K., Mitschang, B.: *Processing and Transaction Concepts for Cooperation of Engineering Workstations and a Database Server*, Data and Knowledge Engineering 3, 1988, 87-107.

[5] Härder, T., Meyer-Wegener, K., Mitschang, B., Sikeler, A.: *PRIMA - a DBMS Prototype Supporting Engineering Applications*, Proc. 13th Int. Conf. on Very Large Databases (VLDB'87), Brighton, 1987, 433-442.

[6] Hübel, C., Sutter, B.: *Supporting Engineering Applications by New Data Base Processing Concepts — an Experience Report*, Engineering with Computers 8, Springer, 1992, 31-49.

[7] Härder, T., Ritter, N.: *DBMS Support in Client/Server Environments - How can CAD Applications Profit?*, submitted for publication, 1997.

[8] Härder, T., Thomas, J.: *RITA — ein rechnergestüztes Informationssystem für technische Anwendungen* (in German), ITG-Fachbericht 137 (STAK'96), Munich, 1996, 111-126.

[9] *IBM DB2 SQL Reference — for common servers (Version 2)*, IBM Corp., 1995.

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

[11] *Informix and Illustra Merge to Create Universal Server*, Whitepaper, Informix Software, Inc., 1996.

[12] ISO/IEC CD 9075 Committee Draft, *Database Language SQL*, Jim Melton (eds.), July 1996.

[13] Katz, R.: *Towards a Unified Framework for Version Modeling in Engineering Databases*, ACM Computing Surveys 22:4, 1990, 375-408.

[14] Kim, W.: *Object-Relational — The Unification of Object and Relational Database Technology*, Whitepaper, UniSQL Inc., Austin, 1996.

[15] Kim, W., Lorie, R., McNabb, D., Plouffe, W.: *Nested Transactions for Engineering Design Databases*, Proc. 10th Int. Conf. on Very Large Databases (VLDB'84), Singapore, 1984, 355-362.

[16] Käfer, W., Schöning H.: *Mapping a Version Model to a Complex-Object Data Model*, Proc. 8th Int. Conf. on Data Engineering, Tempe, Arizona, 1992, 348-357.

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

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

[19] Mitschang, B.: *Extending the Relational Algebra to Capture Complex Objects*, Proc. 15th Int. Conf. on Very Large Databases (VLDB'89), Amsterdam, 1989, 297-305

[20] *Oracle7 Sever SQL Reference (Release 7.3)*, Oracle Corp., Feb. 1996.

[21] Ritter, N., Mitschang, B., Härder, T., Gesmann, M., Schöning, H.: *Capturing Design Dynamics - The CONCORD Approach*, Proc. 10th Int. Conf. on Data Engineering, Houston, Texas, 1994, 440-451.

[22] Stonebraker, M.: *Object-Relational DBMSs — The Next Great Wave*, Morgen Kaufmann, 1996.

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

[24] Zhang, N., Härder, T., Thomas, J.: *Enriching Object-Relational Databases with Relationship Semantics*, 3rd Int. Workshop on Next Generation Information Technologies and Systems (NGITS'97), Israel, 1997.