Proc. Advanced Computer Technology, Reliable Systems and Applications, 5th Annual European Computer Conference, CompEuro 91, Bologna, May 1991, pp. 398-405

Invited Paper

Extending Database Technology Towards Object Orientation: For Whom, Why, and for What?

Peter Dadam
University of Ulm, Faculty of Informatics
Oberer Eselsberg, D-7900 Ulm, Germany
DADAM @ RZ.UNI-ULM.DBP.DE

Abstract

New application areas like Computer Integrated Manufacturing and others are demanding for "object-oriented" database technology. The right approach to extend database technology towards object-orientation is therefore one of the hot research and development issues in the database research community. Sometimes very controversial and confusing discussions are taking place, partially simply due to the fact that the point of view under which the argumentation takes place is not made clear. After a discussion of the different approaches and their underlying motivation, a possible combined approach which takes advantage from the benefits of either approach is outlined and some of the remain open issues are addressed. In total, the paper is attempting to give some answers to the questions *why* an extension of database technology towards object-orientation is needed and *for whom* it would be beneficial. It also tries to give some indications *for what kind* of demands it should be prepared.

1. Introduction and Background

Today's SQL-based [1,2] relational database management systems (DBMSs) have been developed having traditional, *data intensive* applications in mind. They been designed to provide adequate support for searching in large amounts of data for the information on entities or real world objects to be retrieved. In this scenario, the objects of the real world usually can be represented by one tuple or a small number of tuples respectively, e.g. to store a multi-valued attribute like a repeating group, for example. Nowadays, the focus of interest is moving towards the integration of non-traditional application areas like Computer Integrated Manufacturing (CIM), office, geographical applications, etc. Here the problem is to integrate complex applications using large, complex structured data objects, each of these representing different phases of an overall engineering design task to be performed. Figure 1 illustrates the state of the art in the design of robot based manufacturing processes. Very often, different hardware and software systems are used to perform the different steps of the overall design task. As a result, whenever one step is completed, the data has to be moved via data exchange (file transfer + data conversion) to the next system to perform the subsequent design step.

Unfortunately, there are a couple of problems related with this procedure. At first, in many cases not all the data produced at one system can be automatically carried over to the subsequent system due to limitations in the data exchange programs. As a consequence, manual follow-up treatment is often required, creating additional efforts and potential sources of errors. A second and much more serious problem is the total absence of system enforced global control to ensure that always the most current data is submitted to and used in the subsequent step. Consider for example the case where a potential assembly problem is detected in a later phase of the manufacturing design process like, e.g. during robot motion planning. The problem detected may require to change the initial product design (that is the initial CAD data) to eliminate the problem. This, in turn, causes all the data produced during the intermediate design and planning steps and which are depending on this CAD data to become obsolete. Usually there is no system enforced control mechanism to automatically "mark" all these data (versions, representations) accordingly. It, therefore, can rather easily happen (and it happens!) that people continue to work with obsolete versions of data without recognizing it. This is especially "funny" if that kind of problem is only detected at manufacturing time at the shop floor.

A scenario where the local data stores shown in Figure 2 are replaced by a *common engineering database* would therefore certainly be much more desirable. In the ideal case all application systems would directly fetch from and store their data into the common engineering database. Although consistency and redundance-free management of data is also not automatically achieved, the preconditions for coming closer to that goal are much better. Unfortunately, to use today's relational DBMSs in a clean and proper way for this purpose would cause a lot of problems. At first, relational DBMSs require to represent all information in simple tabular form (with rows (called *tuples*) and columns (called *attributes*)) in which only elementary (*atomic*) entries (attribute values) are allowed.

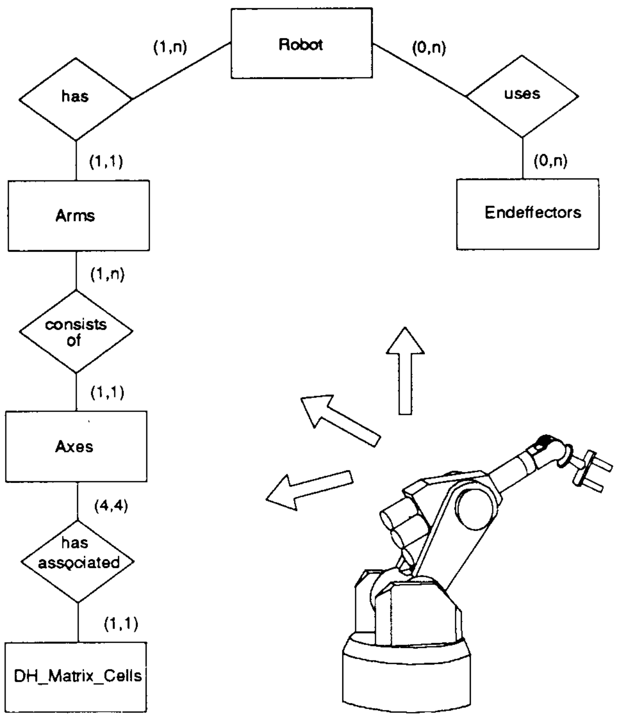

Figure 1: (Simplified) Modelling of Robot Information

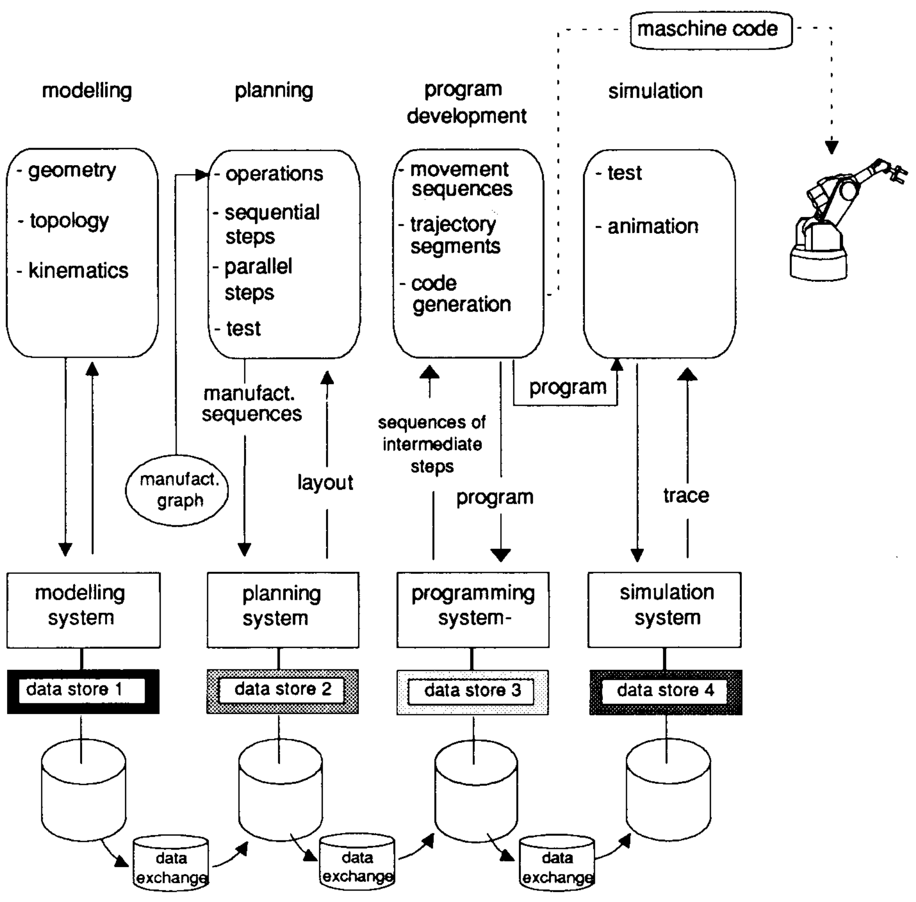

Figure 2: Isolated Engineering Applications

(This is called the *First Normal Form (1NF) condition* [3].) As a consequence many different tables are needed to represent *one* engineering object like a robot, for example, in proper relational form.

Figure 1 is showing some of the information which would be needed to support a computer based robot programming and simulation system (a more detailed treatment of this issue can be found in [4] and [5]). To be represented in proper relational form several tables (relations) similar to those shown in Figure 3 would be necessary (in reality we would even need 15 and more tables to represent the robot information properly; cf. [4]). As the information has to be spread over many tables already simple queries are leading to complex SQL expressions (see Query 1). As one can easily imagine, the complexity of such queries is far beyond that what a "normal" engineer is usually willing to accept.

Unfortunately, not only the complexity of the query is prohibitive but also the resulting response time would be orders of magnitude larger than with the file based solutions of today. Moreover, due to the nature of the join operation which has to be used to combine the tuples (records) from the different tables into one result table, this result table will be very large and will contain many redundant entries; in many cases probably even more than 90%. - Alternatively, an application program could be written to "manually navigate" between the tables using SQL. This is certainly also no attractive solution.

Query 1: *"Show all information on robot Rob1"*

```
SQL:  SELECT  r.Rob_ID, r.Rob_Descr, ar.Arm_ID,
              ac.AxisNo, m.Row, m.Col1, m.Col2,
              m.Col3, m.Col4, ac.JA_min, ac.JA_max,
              ac.Mass, ac.Accel, re.Eff_ID, e.Function
      FROM    Robot r, Robot_Arms ar, Axes ac,
              Matrices m, Robot_Endeffectors re,
              Endeffectors e
      WHERE   r.Rob_ID = 'Rob1' AND
              r.Rob_ID = ar.Rob_ID AND
              ar.ROB_ID = ac.Rob_ID AND
              ar.Arm_ID = ac.Arm_ID AND
              ac.Arm_ID = m.Arm_ID AND
              ac.Rob_ID = m.Rob_ID AND
              ac.AxisNo = m.AxisNo AND
              r.Rob_ID = re.Rob_ID AND
              re.Eff_ID = e.Eff_ID
```

Regardless of the alternative which is selected, processing of the result of this query by the application program becomes slow and cumbersome. For these and other reasons the relational DBMSs of today can not be adequately used for engineering applications. If they are used at all, they are usually managing "byte containers" which contain the real data. Some information about the contents

Robots	
Rob_ID	Rob_Descr
Rob1	Speedy 400 ....
Rob2	Speedy 600 ....
Rob3	Colossus MX-3 ....
:	:

Robot_Arms	
Rob_ID	ArmID
Rob1	left
Rob1	right
Rob2	solo
Rob3	left
Rob3	middle
Rob3	right
:	:

Endeffectors	
Eff_ID	Function
GR700	Gripper Type 600 ....
GR700	Gripper Type 700 ....
LW1	Laser Welding Equipment Type 1 ..
PW1350	Point Welder Type 1350 ....
PW1380	Point Welder Type 1380 ....
PW1510	Point Welder Type 1510 ....
SD200	Screw Driver Type 200 ....
SD300	Screw Driver Type 300 ....
:	:

Robot_Endeffectors	
Rob_ID	Eff_ID
Rob1	SD200
Rob1	SD300
Rob1	PW1380
Rob1	GR700
:	:
Rob2	SD300
Rob2	SD300
Rob2	PW1510
Rob2	LW1
:	:

Axes						
Rob_ID	Arm_ID	AxisNo	JA_min	JA_max	Mass	Accel
Rob1	left	1	-90	90	40,0	1,0
Rob1	left	2	-170	180	30,5	1,5
Rob1	left	3	-180	180	20,0	3,0
:	:	:			:	:

Matrices							
Rob_ID	Arm_ID	AxisNo	Row	Col1	Col2	Col3	Col4
Rob1	left	1	1	1	0	0	1
Rob1	left	1	2	0	0	1	0
Rob1	left	1	3	0	-1	0	80
Rob1	left	1	4	0	0	1	1
Rob1	left	2	1	0	0	0	60
:	:	:	:	:	:	:	:
Rob1	left	3	4	0	-1	0	70
Rob1	right	4	1	0	-1	0	70
:	:	:	:	:	:	:	:
Rob2	solo	1	1	1	0	0	1
:	:	:	:	:	:	:	:

Figure 3: Representing the Robot Information in Relational Tables

of the byte containers is stored in proper relational form allowing at least some limited types of queries while the major portion of the data is stored in "cryptic" binary form which can only be understood by the application program that created it. A not very desirable solution with respect to the implied redundancy of data and the related consistency problems as well as to the resulting data dependence of application programs which one wanted to reduce with relational database (DB) technology (cf. Sect. 2). In order to provide better support for *complex objects* of this kind possible extensions of the relational data model are under discussion already since the early eighties [6, 7]. The goal of these and related approaches is to provide *structural object-orientation* [8] to a certain degree within a general purpose DBMS to improve efficiency as well as to simplify complex object related queries.

Missing adequate data structuring capabilities is not the only problem in this context, however. Assume, for example, one wants to store the geometrical representation of a cuboid using the x-, y-, and z-coordinates of the corners (see [9] for a more detailed treatment of this subject). In this case 8 x 3 = 24 real numbers would be needed. A possible database representation would be to store these 24 real numbers as attribute values of one tuple (e.g. as attributes v111, v112, v113, v121, ..., v888). Every modify operation applied on the cuboid like to move it in space, to rotate it, or to change its size would mean to update these 24 real numbers accordingly. No update operations should be allowed, however, which would violate the cuboid property of this geometric object (integrity constraint). Unfortunately, in most cases of this kind the *semantics* of such interrelated attributes are not known to the DBMS. Usually it will be completely hidden in the application programs which are accessing and manipulating these numbers. As a consequence, the DBMS is usually not able to perform any serious consistency checking in such cases. Even worse, because there are no DBMS provided operations for manipulating such special types like a cuboid, every programmer is practically free to implement his/her own algorithms. As one can easily imagine, this is a very error-prone way to deal with data of such type of data.

Having already problems with such relatively simple things like the adequate structural and behavioural representation of a cuboid, the situation becomes really bad when looking at more complex objects like the arm of a robot or the robot as a whole. Not only to keep the related data consistent is a hard problem, also the formulation of expressive queries like *"Find all robots whose arms (end points) could be moved along a given sequence of trajectories"* would be far beyond the expressive power of today's SQL. Therefore, for keeping complex object data consistent as well as for formulating expressive queries mechanisms are needed to reflect the *behaviour* of the objects in the DBMS as well. That is structural object-orientation and *behavioural object-orientation* [8] have to be provided by the DBMS in order to support complex objects adequately.

Although there is no disagreement on these aspects of in general, there are different standpoints and controversial discussions on *how* and *to which degree* structural and behavioural object-orientation should be supported by the DBMS [10, 11]. Depending on the individual standpoint different, contradictory goals are emphasized leading sometimes to some confusion - not only outside of the DB research and development (R&D) community. The main reason for this confusion is that one can look at the purpose of DBMSs from different points of view. One point of view is to look at DBMSs as being tools to speed up application programming. If looking only at the implementation of *individual* (isolated) *applications*, the convenience and adequacy of the provided DBMS data structures and functions is most important. Certainly, also the performance should be comparable to using conventional (file based) programming techniques. If looking at applications which have to cooperatively work in an *integrated application environment*, however, aspects like data sharing, independence of application programs from database data structures, a common database representation of data which is independent from any specific programming language, query capabilities to efficiently search in large collections of data, and other related issues are usually dominating. Unfortunately, these isolated points of view are not very helpful to achieve much progress towards a new DB technology which is broadly applicable because only parts of the total spectrum of problems are taken into consideration. The purpose of this paper therefore is to outline how a possible common approach to the overall problem could look like which could incorporate a lot of the good technology developed at either side.

According to that goal the remainder of the paper is organized as follows: In the next section we outline the development of DBMS technology and its related achievements and remaining weaknesses. In Sect. 3 the different approaches for object-oriented databases pursued in the DBMS R&D community are presented and discussed. How a combined approach taking advantage from the strong technologies at either side is outlined in Sect. 4. The paper ends with a summary in Sect. 5.

2. Achievements of Relational Database Technology

The first generation of DBMSs which based on the hierarchical [12] or network [13] data model was (more or less) directly exposing its internal linked record database structures to the application programmer. As a consequence, the application programs were rather heavily dependent on the underlying physical database structures. The introduction of the second generation of DBMS products based on the *relational data model* [14] in the early eighties has significantly simplified application programming and usage of database technology in total. In these DB systems in all information has to be stored in simply structured ("flat") tables and all relationships between tuples of the same or another table have to be represented by appropriate attribute values, e.g. so-called *primary key - foreign key* relationships [3]. By doing so, the degree of separation between database and application program has been enlarged significantly compared to the first generation DBMSs. In those DBMSs nearly every kind of schema change (e.g. adding a new record type or adding a new attribute to an existing record type) requires program modifications or at least re-compilation of all application programs referring to this schema. Opposed to that, in relational DBMSs tables as well as indexes can be dynamically created and dropped. *Virtual tables* (*views*) are an additional mechanism for providing a higher degree of independence of application programs from physical database structures.

The relational SQL database language which has become an ISO standard in 1987 [1] is additionally contributing to shield the application program from physical database structures. Instead of procedural navigation between records (tuples), the desired information to be retrieved is described in a declarative way. As SQL is based on a solid formal basis (relational algebra, relational calculus), the DBMS can internally transform SQL query expressions into equivalent query expressions to find the optimal way for executing the query. That is, a relational DBMS is able to perform *query optimization* by itself. The application programmers, therefore, can formulate their queries such that they appear most natural to them and need not (at least should not have) to worry about how these queries will be executed. This aspect of system provided query optimization becomes especially important if database queries are not directly formulated by humans but are generated by application programs [15] or in the context of distributed databases when queries have to be decomposed and executed at remote sites [16]. Over the last ten years relational DBMS technology has also been significantly improved to meet even high performance requirements making it possible to use these systems also for operational applications [17]. In addition, powerful concepts like referential integrity features supporting cascading delete and cascading update operations, assertions, triggers, and other things are already available or will be available in the near future [18] making SQL-based DBMSs very powerful for a large variety of applications.

Despite of all these achievements the relational DBMSs of today reveal severe weaknesses when to be applied in so-called *non-standard application areas* as already pointed out above. When working on extensions DB technology towards object-orientation we should try as hard as we can to preserve the technological achievements of this technology. This target will also be the guideline for the subsequent discussions.

3. Approaches Towards Object-Orientation

Depending on whether DBMS technology is seen as tool to support application development or the aspect of integration of applications is in the foreground, the discussion about object-orientation of DBMSs or *object-oriented DBMSs* (OODBMSs) is performed from a *programming language oriented* point of view of from a (DB-)*system-oriented* point of view. In the first case the behavioural aspect of OODBMS is usually more emphasized. In the second case the concentration is more on the structural aspect of OODBMSs in general. One could also say that the goal (the OODBMS) is approached from two different sides (see Figure 4). We will first discuss the two approaches separately in the following two subsections and then try to outline a possible combined approach in Sect. 4.

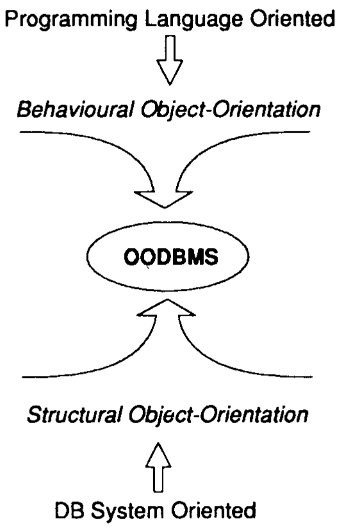
Programming Language Oriented
Behavioural Object-Orientation
OODBMS
Structural Object-Orientation
DB System Oriented

Figure 4: Approaches Towards OODBMS Technology

3.1 Programming Language Oriented Approaches

Compared to the query languages of the first generation DBMSs, SQL is a very elegant, easy to use, and powerful query language. Unfortunately much of this elegance of SQL gets lost when using the application program interface (API). One reason is that SQL is not fully integrated into the programming language used for application programming. That is the SQL statements used within application programs are not an integral part of the (host) programming language but are only *embedded*. A special host-language dependent SQL pre-compiler has to be used to translate these embedded SQL statements into respective statements of the host programming language. The other reason is that SQL is a set-oriented query language returning a *set* of tuples as the result of a DB query while the usually used host programming languages only support one-record-at-a-time processing. *Cursors* and corresponding FETCH operations are to be used for transferring the result of an SQL query, one tuple after another, into respective programming variables of the host program.

To overcome these problems extensions of programming languages with fully (transparently) integrated DB interfaces have been proposed and implemented. The earliest and best-known proposal of this category is Pascal/R [19] which has stimulated a lot of research activities, today usually labelled as *persistent programming* (languages) or *persistent data types* [20]. Nowadays the focus has moved from embedding simple relational structures and related operations, however, towards more complex structured entities (*objects, object classes*) and their related operations (*methods*). Objects and their related operations are considered as being one unit. Application programs are accessing objects for reading or modification purposes only via methods associated with these objects in general. Consequently, the (internal) object *structure* is usually not visible to the applications using it. As opposed to embedded SQL the database interface is (at least to a certain degree) transparently integrated into the object-oriented programming language by declaring an object type to be persistent of by providing respective methods to read and write objects. The "host" programming languages discussed in this context are usually offering very advanced object-oriented programming features like powerful type and class concepts, inheritance, polymorphism, etc. [20]. Consequently, when talking about the required features of a DBMS to be called "object-oriented" groups following this approach are emphasizing the object-oriented programming language features (cf. [10] for example). First commercial products basing on C^{++} [21] and Smalltalk [22] respectively which follow this philosophy are currently already appearing at the market place.

3.2 Database System Oriented Approaches

When talking about "efficiency" of OODBMSs in the context of the programming language oriented approaches described above, usually either the saving in application development time is meant or the performance (e.g, access time) of the resulting solution compared to a "hand coded" one. Usually, also only a relatively small number of objects (which may be rather complex ones however) is assumed. As a consequence, sophisticated query capabilities, algebraic query optimization, and automatic access path selection are not the major issues under consideration there. Opposed to that, more "system near" DBMS R&D groups are concentrating on the *integration aspect* and, therefore, are usually much more concerned about the DBMS related performance and internal DBMS functionality issues. An OODBMS which is behaving only well for retrieving some of the few complex objects it is managing but is behaving badly when having to search in large collections of data ("the commercial bread and butter applications") is considered to be not acceptable. On the other side, to have totally different types of database systems for either scenario is not desirable as well because of the resulting redundancy (consistency) and integration problems. In order to keep the DBMSs efficient the system-oriented approaches to OODBMS tend to keep a much lower profile with respect to proposing fancy object-oriented database or data model extensions compared to groups who are favouring the programming language oriented approach described above.

The challenge is to find appropriate DBMS enhancements for the data representation and query capabilities such that "orthogonality" and "closure" of language constructs (with respect to the underlying data model) are achieved or preserved respectively and that APIs can be developed for these more powerful data structures which permit safe application programming. *Orthogonality* means that query language expressions can be (nearly) arbitrary combined or nested. Wherever a constant value of a certain type is syntactically allowed, a variable, a function, or an expression returning a value of this type is allowed as well. Today's SQL is not orthogonal in this sense (no table "expressions" are allowed in the from-clause, for example), but future SQL standards will very likely improve this situation to certain extent [18]. Orthogonality is a very important factor for making database as well as ordinary programming languages easier to understand because no complex exception rules have to be obeyed. *Closure* means that one never "leaves" the data model. That is every query expression produces a result type which is also a legal object type (structure type) of the underlying data model and thus could either be directly stored in the database or could be input for a subsequent query step (nested queries; more detailed discussions of these aspects can be found in [23, 24]).

Among the various proposals for extending database technology to support *"new" data models* providing at least a certain degree of structural object-orientation, *nested relations* (also called *NF^2 Relations* [7]) are certainly having the broadest theoretical basis. The reason is that work on generalizing the relational theory has already been started in the early eighties and has led to many results with respect to appropriate extensions of the relational algebra and its underlying theory [7, 25, 26], appropriate SQL language extensions [27, 28], and database design theory [29, 30, 31], to name just a few out of each area (see [32], for example, for further references). AIM-P [33] (in the initial phase) and DASDBS [34] are examples of experimental DBMSs based on this approach. Unfortunately, also nested relations are offering only rather limited facilities to structure data (or object). Already a simple thing like a vector of integer numbers, e.g. a series of measurement values, can not be represented adequately.

Therefore further developments of the NF^2 data model have taken place which finally resulted in a data model supporting lists, sets, tuples, and atomic values in practically every combination as legal database objects (cf. Figure 5). AIM-P [35, 36] and also O_2 [37] are both based on this type of data model. Beside offering a more powerful data model and a DB query language supporting it, the support of user defined data types and functions played a major role in the development of AIM-P [38].

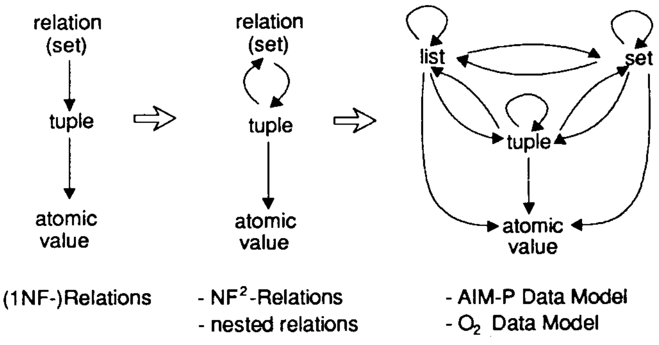

Figure 5: Evolution of Relational Data Models

One target was to enable users to enhance the system's query capabilities by implementing their own functions which could be used within AIM-P's SQL-like query language called HDBL (Heidelberg Data Base Language [27, 35]). Another target was to support these data types and functions also in the application programming interface. By doing so, the same user defined functions applicable within HDBL become applicable in the application program as well. More interesting however: the DBMS can be used to efficiently load complex objects into predefined main memory structures for further processing within the application program. As those "external" data representations are the same as being used to implement new functions for HDBL, additional application specific functions and procedures can be provided to implement a high-performance main memory based *object cache* as been done in conjunction with the R^2D^2 project

[39, 40, 41]. This mechanism has been used for example to provide abstract data types (hiding the object cache internals) for a robot programming environment like the one illustrated in Figure 2 (but with an integrated engineering database based on AIM-P [5]).

In AIM-P it is left up to the type implementor whether the (internal) structure of the type (which is constructed using HDBL's basic data types and its set, list, and tuple constructors) or whether it shall be hidden (called *encapsulated types* [38]). By doing so, the type implementor can decide whether just an additional built-in function is wanted or an abstract data type behaviour shall be enforced. To be able to adequately support hidden types in AIM-P's interactive ad hoc query interface the type implementor can provide display functions which will automatically replace AIM-P's standard output screen representation for all data of this type [42]. AIM-P also allows the application programmer to decide himself/herself whether to use the type mechanism to move database objects into respective main memory structures, or to use nested, hierarchical cursors (one cursor for every level of the hierarchy) to transfer the object attribute wise or tuple wise into appropriate host program variables, or to mix both concepts [43]. In the first case a so-called "external type representation" to be used in the application program is generated by AIM-P's type compiler [38]. This external type representation is used in the sequel to pass the object from the database to the application program and vice versa. In the second case the application programmer decides how the main memory representation shall look like but a sequence of cursor operations will be necessary in general to fill the application program's data structure.

Updates can be (like in SQL) performed by either using the respective HDBL commands or by using the API. If the API is used all updates are performed locally at first in the so-called *object buffer* which is under the control of the API run-time system. An object buffer is created when a database retrieval operation is issued from the API. The query result (the object(s)) is passed to the API run-time system using the object buffer. The object buffer is also used in the sequel to reflect all modifications performed by the application program using the API. Last but not least the object buffer is used to communicate the changes to the DBMS at commit time. The implementation of the object buffer is done such that only the modified parts are communicated by the API run-time system to the DBMS (cf. [44] for further details on this subject).

Unfortunately, due to the rather limited support for dynamic data structures in most conventional programming languages (HDBL has been embedded in Pascal), much of the elegance of this data model is lost when using the API. Also the function mechanism supported to enhance AIM-P's query capabilities is rather limited compared to the powerful methods (in conjunction with the inheritance of methods) proposed by programming language oriented groups. That is many of the arguments which have led to the programming language oriented approaches for OODBMS mentioned in Sect. 3.1 are still valid. On the other hand, many of the programming language oriented proposals are rather weak in the database part, especially under the aspects of adequate structural representation of objects, concurrency control, high-level query languages, query optimization, automatic access path selection, etc. It would therefore be desirable to combine both approaches to benefit from the good technologies at either side. How such a combined approach could be look like shall be outlined in the following.

4. Towards a Combined Approach for OODBMS

Complex design tasks like e.g. computer aided robot assembly or computer aided parts design are touching the objects to be worked with very often more than once. That is usually *sequences* of operations (methods) are applied rather than finishing the task with just one method invocation. To cross a DBMS query interface multiple times to apply the desired sequence of methods is both awkward and inefficient in most cases. For such kind of application *local processing* (within the application program) is therefore desirable. In such cases the database related part should be reduced to fetch the object(s) wanted and to propagate the changes back to the database after the modifications have been locally performed. This means that the sophisticated methods for modifying objects have to be available in the application programming environment in the first place. There is no such strong need to have all of these methods also available as part of the DBMS kernel or as part of the DB query language respectively.

This separation can only properly work, however, if the DBMS can "trust" the application programming environment that all modifications performed have been done using the related methods. That is no application program has illegally "by-passed" any method. Otherwise costly integrity checks would have to be performed during check-in processing by the DBMS which would probably completely eliminate the advantages of this approach. DBMS based integrity checking should be reduced to such cases where no predefined methods which are enforcing the respective integrity constraints can be applied. To go that way means that the DBMS based integrity checking mechanism has to be coordinated somehow with the method implementation to avoid double checking or missed checking. This may also require to use special coding mechanisms such that the DBMS can determine with safety whether a certain modification has been performed by using the respective method or not. Superimposed coding (that is some kind of check sum computing) plus cryptographical methods may be helpful techniques for doing this. This separation of processing does not mean that only simple query capabilities have to be provided by the DBMS. In order to efficiently determine the objects wanted within may be large collections of "similar" objects, adequate search capabilities have to be offered as well. This leads rather naturally to the requirement to support user defined data types and functions (methods) at the DBMS side as well. They can be reduced however to a large extent to the support of new types of query predicates or visualization of new data types when using the interactive ad hoc query interface (if provided).

A certain problem is the identification of object modifications when following this approach. For efficient back propagation of changes to the DBMS (especially in the context of versioned objects) the modified objects should not (or even can not) require to simply overwrite the old objects in the database. Instead, only the changes should be communicated (or should be at least easily detectable) to allow the DBMS to select the most appropriate update strategy. As this is crucial for consistency reasons (no modifications should be "forgotten") as well as for performance reasons, the DBMS provided type representations in the object buffer and the respective usage in the application programming environment (e.g. to implement new methods) have to be carefully coordinated. This is certainly a point which deserves further attention.

Last but not least a similar (at best the same) "look and feel" should be provided for all host programming languages using the database. To avoid a "semantic break" between the database data model and the respective programming language representation appropriate "embedded" data structures and operations defined on it have to be provided in a uniform way. How this could look like is demonstrated in O_2 for example [45, 46]. That is the application programmer should not have to worry about how a "list of list of set of ..." is physically represented in the host programming language used. He/she should only work with appropriate high-level structures and operations within the application program. The mapping to real data structures and operations of the host programming language should be done by the pre-compiler. This is also a point which deserves some more attention. The discussion about data structures and operations is also leading to the question which kind of data model should be supported by the DBMS. Shall it be set-oriented and value-based, or shall it support explicit links and navigation?

When discussing about *navigation* one has to carefully discriminate between navigational access *within* objects and navigation *between* objects. Navigation *within* objects is a must without any question. This is nothing new, however. The cursor construct provided by today's SQL API is nothing else than a mechanism to navigate in the result table (the result "object"). More complex result structures, once supported, will certainly require more powerful navigation mechanisms than are provided today. Opposed to that, navigation *between* objects, especially in conjunction in a record-oriented way of database access would be a major step backward into the direction of first generation DBMSs. As far as flexible object structures and shared subobjects are concerned, one could think of appropriately extended SQL query capabilities based on a higher level (e.g. ER-like) data model allowing to describe dynamically and to retrieve even network-like result (object) structures based on flat relations [47, 48, 49]. With such extensions there should only be very few cases left over where "manual" navigation remains to be necessary.

There are sometimes also rather dogmatic discussion on either side about the differences between a primary key and an object-id. It would certainly be preferable to have not more different concepts to support than necessary. A more general treatment of the attribute concept might be the right way to achieve this goal. One could, for instance, distinguish between attributes whose values can be *user-provided, system-provided, or both.* This would allow the DBMS to automatically generate attribute values on demand and depending on the attribute type and further characteristics (e.g. primary key). Orthogonal to this concept one should be able to define whether an attribute value, once provided, may be changed (updated) or not. By doing so, the concept of an object identifier and the concept of a primary key are just variants of the same basic concept. Normal SQL-like query mechanisms could then be used to select such objects. Clearly, appropriate language constructs have to be provided to discriminate between retrieving only the object identifier and the object itself. But this is trivial (cf. [27], for example).

5. Summary

Without any doubts database technology is one of the key technologies for performing the complex integration tasks many industrial and other companies are faced with already today or in the foreseeable future. As has been illustrated in the paper, today's database technology is not really prepared yet to adequately support the demands arising from non-standard application areas like, e.g. Computer Integrated Manufacturing which we have also used to study the problem. The same kinds of problems are also appearing in other application areas like office, geographical, medical, and other non-traditional application areas. That is, the current discussion about extending database technology towards object-orientation has its real practical roots. This answers also the question in the title *why* database technology should be extended towards object-orientation.

We have also elaborated the different points of view under which the current controversial discussion on OODBMSs are taking place at present and have pointed out some the strengths and weaknesses of the related proposals. We have discussed the different requirements or priorities respectively when focussing on the aspect of speeding up application development and when focussing on the aspect of implementing integrated applications. This was an attempt to give some more insights for *whom* database technology has to be extended and *for what* kind of problems the new technology has to be prepared.

Finally, we have tried to outline how a combined approach could look like which takes advantage of the good technologies available or under development at either side. In this context we have also addressed some of the open issues which deserve further attention.

Acknowledgements

The valuable comments of A. Dyballa, K. Gaßner, and Chr. Kalus on an earlier version of this paper are gratefully acknowledged. Thanks also to my former colleagues at the IBM Heidelberg Scientific Center, especially the members of the AIM research group, which contributed at a lot over all the years (1982-1990) to gain the insights and experiences expressed in this paper.

6. Literature

[1] International Standards Organization (ISO), "Database Language SQL", Document ISO-9075-1987(E).

[2] International Standards Organization (ISO), "Database Language SQL with Integrity Enhancement", Document ISO-9075-1989(E).

[3] C.J. Date, An Introduction to Database Systems, vol. 1, 4th ed., Addison-Wesley Publ. Comp., 1986.

[4] P. Dadam, R. Dillmann, A. Kemper, P.C. Lockemann, "Objektorientierte Datenhaltung für die Roboterprogrammierung", Informatik Forschung und Entwicklung, Springer-Verlag, Heidelberg, vol. 2, 1987, pp. 151-170 (in German).

[5] R. Dillmann, M. Huck: "R^2D^2: An Integration Tool for CIM", in [53], pp. 355-372.

[6] R.L. Haskin, R.A. Lorie, "On Extending the Functions of a Relational Database System", in Proc. ACM-SIGMOD, Int. Conf. on Management of Data, Orlando, Florida, 1982, pp. 207-212.

[7] G. Jaeschke, H.-J. Schek: "Remarks on the Algebra of Non First Normal Form Relations", in Proc. ACM-SIGACT-SIGMOD Symposium on Principles of Database Systems, Los Angeles, March 1982, pp. 124-138.

[8] K.R. Dittrich, "Object-oriented Database Systems: The Notions and the Issues", in [50], pp. 2-4.

[9] A. Kemper, M. Wallrath, "An Analysis of Geometric Modeling in Database Systems", ACM Computing Surveys, vol. 19, no. 1, pp. 47-91, March 1987.

[10] M. Atkinson et al, "The Object-Oriented Database Manifesto", Altair Technical Report No. 30-89, GIP Altair, LeChesnay, France, Sept. 1989

[11] M. Stonebraker et al., "Third-Generation Database System Manifesto", ACM SIGMOD Record, vol. 19, no. 3, 1990, pp. 31-44.

[12] J. Strickland, P. Uhrowczik, V. Watts: "IMS/VS: An Evolving System", IBM Systems Journal, vol. 21, no. 4, 1982.

[13] CODASYL: "Data Description Language Committee Report", Information Systems, vol. 3, no. 4, 1978, pp. 247-320.

[14] E.F. Codd: "A Relational Model for Large Shared Data Banks", Communications of the ACM, vol. 13, no. 6, 1970, pp. 377-387.

[15] N. Ott, K. Horlaender: "Removing Redundant Join Operations in Queries Involving Views", Information Systems, vol. 10, no. 3, 1985, pp. 279-288.

[16] S. Ceri, G. Pelagatti, Distributed Databases, Principles and Systems. McGraw-Hill Book Comp., 1984.

[17] D.J. Haderle, "Database Role in Information Systems: The Evolution of Database Technology and its Impact on Enterprise Information Systems", in [54], pp. 1-14.

[18] Ph. Shaw, "Database Language Standards: Past, Present, and Future", in [54] , pp. 55-80.

[19] J.W. Schmidt: "Some High-Level Language Constructs for Data of Type Relation", ACM Transactions on Database Systems, vol. 2, no. 3, Sept. 1977, pp. 247-261.

[20] M.P. Atkinson, P. Buneman, R. Morrison (Eds.): "Data Types and Persistence", Topics in Information Systems, Springer-Verlag, 1988

[21] Ontologic Inc.: ONTOS - Object Database Programmer's Guide, 1989

[22] R. Bretl et al.: "The GemStone Data Management System", in [52], pp. 283-308.

[23] S. Abiteboul, C. Beeri, M. Gyssens, D. Van Gucht: "An Introduction to the Completeness of Languages for Complex Objects and Nested Relations", in [32], pp. 117-138.

[24] A. Heuer, M.H. Scholl, "Principles of Object-Oriented Query Languages", in [55], pp. 178-197.

[25] H.-J. Schek, M. Scholl: "The Relational Model with Relation-Valued Attributes. Information Systems, vol. 11, no. 2, 1986, pp. 137-147.

[26] F. Bancilhon, P. Richard, M. Scholl: "On Line Processing of Compacted Relations", in Proc. Int. Conf. on Very Large Data Bases, Mexico, Sept. 1982, pp. 263-269.

[27] P. Pistor, R. Traunmüller, "A Database Language for Sets, Lists, and Tables", Information Systems, vol. 11, no. 4, 1986, pp. 323-336.

[28] H.F. Korth, M.A. Roth, "Query Languages for Nested Relational Databases", in [32], pp. 190-204.

[29] Z.M. Ozsoyogly, L.Y. Yuan, "A New Normal Form for Nested Relations", ACM Transactions on Database Systems, vol. 12, no. 1, 1987, pp. 111-136.

[30] Z.M. Ozsoyogly, L.-Y. Yuan: "On the Normalization in Nested Relational Databases", in [32], pp. 243-271.

[31] M.A. Roth, H.F. Korth: "The Design of Non 1NF Databases into Nested Normal Form", in Proc. ACM-SIGMOD Conf., San Francisco, May 1987, pp. 143-159.

[32] S. Abiteboul, P.C. Fischer, H.-J. Schek (Eds.): Nested Relations and Complex Objects in Databases. Lecture Notes in Computer Science 361, Springer-Verlag, 1989

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

[34] H.-B. Paul et al., "Architecture and Implementation of the Darmstadt Database Kernel System", in Proc. ACM-SIGMOD Conf., San Francisco, 1987.

[35] P. Pistor, P. Dadam, "The Advanced Information Management Prototype", in [32], pp. 3-26.

[36] P. Dadam, V. Linnemann, "Advanced Information Management (AIM): Advanced Database Technology for Integrated Applications", IBM Systems Journal, vol. 28, no. 4, 1989, pp. 661-681.

[37] Ch. Lecluse, P. Richard, F. Velez, "O_2 - An Object Oriented Data Model", in: F. Bancilhon, P. Buneman (Eds.): Advances in Database Programming Languages, ACM Press, 1986, pp. 257-276.

[38] V. Linnemann et al., "Design and Implementation of an Extensible Database Management System Supporting User Defined Data Types and Functions, in Proc. Int. Conf. on Very Large Databases, Los Angeles, Aug. 1988, pp. 294-305.

[39] P. Dadam et al., "Managing Complex Objects in R^2D^2", in [53], pp. 304-331.

[40] A. Kemper, M. Wallrath, M. Dürr, "Object Orientation in R^2D^2", in [53], pp. 332-353.

[41] A. Kemper, M. Wallrath, "A Uniform Concept for Storing and Manipulating Engineering Objects", in [51], pp. 292-297.

[42] S. Hartig, N. Südkamp, "User Defined Procedures for Displaying Database Objects", IBM Heidelberg Scientific Center, Technical Note, August 1988.

[43] R. Erbe, N. Südkamp, G. Walch, "An Application Program Interface for a Complex Object Database", in Proc. 3rd Int. Conf. on Data and Knowledge Bases, Jerusalem, June 1988.

[44] K. Küspert, P. Dadam, J. Günauer, "Cooperative Object Buffer Management in the Advanced Information Management Prototype", in Proc. Int. Conf. on Very Large Databases, Brighton, U.K., Sept. 1987, pp. 483-492.

[45] Ch. Lécluse, Ph. Richard, "The O_2 Database Programming Language", Altair, Rapport Technique, 26-89.

[46] F. Bancilhon et al., "The Design and Implementation of O_2, an Object-Oriented Database System", in [51], pp. 1-22.

[47] R. Lorie, H.-J. Schek, "On Dynamically Defined Complex Objects and SQL", in [51], pp. 323-328.

[48] H. Pirahesh, C. Mohan, "Evolution of Relational DBMSs Toward Object Support: A Practical Viewpoint", in [55], pp. 16-37.

[49] Th. Härder et al., "PRIMA - DBMS Prototype Supporting Engineering Applications", in Proc. Int. Conf. on Very Large Databases, Brighton, U.K., 1987, pp. 433-442.

[50] K. Dittrich, U. Dayal (Eds.): Proc. Int. Workshop on Object-Oriented Database Systems, Asilomar, Pacific Grove, Cal. 1986

[51] K.R. Dittrich (Ed.): "Advances in Object-Oriented Database Systems", ACM-IEEE Int. Workshop on Object-Oriented Database Systems, Bad Münster, Germany, Sept. 1988. Lecture Notes in Computer Science 334, Springer-Verlag, 1988.

[52] W. Kim, F.H. Lochovsky (Eds.): Object-Oriented Concepts, Databases, and Applications. ACM Press, 1989.

[53] G. Krüger, G. Müller (Eds.): HECTOR, Volume II: Basic Projects, Springer-Verlag, 1988

[54] A. Blaser (Ed.), Proc. Int. Symp. Database Systems of the 90s, Müggelsee, Berlin, Nov. 1990, Lecture Notes in Computer Science 466, Springer-Verlag 1990.

[55] H.J. Appelrath (Ed.), Proc. "Datenbanksysteme in Büro, Technik und Wissenschaft", Kaiserslautern, Germany, March 1991 (Informatik-Fachberichte 270, Springer-Verlag, 1991).