# Of Objects and Databases: A Decade of Turmoil

Michael J. Carey
IBM Almaden Research Center
650 Harry Road, K55/B1
San Jose, CA 95120
carey@almaden.ibm.com

David J. DeWitt
Computer Sciences Department
University of Wisconsin-Madison
Madison, WI 53706
dewitt@cs.wisc.edu

## Abstract

A decade ago, the connection between objects and databases was new and was being explored in a number of different ways within our community. Driven by the perception that managing traditional business data was largely a solved problem, projects were investigating ideas such as adding abstract data types to relational databases and building extensible database systems, object-oriented database systems, and toolkits for constructing special-purpose database systems. In addition, work was underway elsewhere in the computer science research community on extending programming languages with database-inspired features such as persistence and transactions.

In this paper, we take a look at where our field was a decade ago and where it is now in terms of database support for objects (and vice versa). We look both at research projects and at commercial database products. We share our vision and our biases about the future of objects and databases, and we identify a number of research challenges that remain to be addressed in order to ultimately achieve our vision.



**Proceedings of the 22nd VLDB Conference**
**Mumbai(Bombay), India, 1996**

## 1 Introduction

Ten years ago, the database field was on the verge of an interesting but confusing new era – the era of objects and databases. As in much of the rest of computer science, the term "object" meant different things to different people in the database community. In addition, there are multiple ways in which object-oriented technology can impact database systems (both internally and externally), and explorations of many of them were newly underway at that time. Most of the work going on then can be grouped into four rough areas:

1. Extended relational database systems
2. Persistent programming languages
3. Object-oriented database systems
4. Database system toolkits/components

Research on extended relational database systems had been in progress for several years in 1986, and it was beginning to bear significant fruit [Ston86a]. Although work on persistent programming languages had also been underway for some time in the programming languages community [Atki87], work on applying those ideas to object-oriented languages was just taking off. Object-oriented database systems were a brand new idea, having just been born [Cope84], and nobody was quite sure what they should be yet. Finally, work on database system toolkits and component architectures, including our own EXODUS project [Care86b], had also just begun. For those wishing to gain a deeper perspective into the field at that time, [Ditt86] provides an interesting and reasonably accurate snapshot of objects and databases as of 1986.

A decade later, things are much clearer, at least in our view. Not all of the above areas have panned out, though each has produced an interesting stream of research results that will likely outlast their specific areas. In fact, we anticipate a not-too-distant future with only one real survivor remaining from the above list; this survivor, of course, will have benefited from both the successes and the mistakes of the others. The goal of this paper is to take an informal look at where we have been, where we are now, and where we are headed – in our "unbiased" opinion, of course! We caution the reader that this is *not* intended as a scholarly work, and as such, our references are spotty and incomplete. We refer interested readers to resources such as [Zdon90, Ston94, Kim95] as well as to the proceedings of conference series like *VLDB* and *SIGMOD* for more information about topics and results that we can only touch upon here.

The remainder of this paper is organized as follows. Section 2 reviews the state of objects and databases as of a decade ago, examining the various ways in which these two technologies were being combined. Section 3 looks at where we are today, looking at how the different combinations have panned out in terms of research accomplishments, commercial database systems, and standards. In Section 4, we present our views about where objects and databases are heading and how we as researchers can help the field to get there from here. Finally, Section 5 concludes the paper.

## 2 Objects & Databases in 1986

As we have just explained, the world of objects and databases was an exciting but confusing place in 1986. Traditional database researchers were extending their favorite data model – relational – to incorporate new, more complex types of data. Programming language researchers were busily adding persistence (permanent data storage) to their favorite programming languages, and object-oriented languages were rapidly gaining their favor. Other researchers were proposing more radical approaches to accommodating the data management needs of new applications: One camp believed that combining key features from both object-oriented programming languages and database management systems would yield a new generation of one-size-fits-all database systems. Another camp felt that the right answer was a toolkit to aid system developers in building domain-specific database management systems; this camp saw objects as an important contributing technology to such a toolkit.

### 2.1 Extended Relational Database Systems

The first approach proposed for moving databases beyond the realm of traditional business applications was an evolutionary approach: open up the type system of a relational database system to allow for the addition of new, user-defined abstract data types (ADTs). To define a new ADT, a user was required to implement the type – defining its representation and writing its functions – in an external programming language (e.g., C). The type would then be registered with the database system, making the system aware of its size and its available functions; included among the functions provided would be functions to input and output instances of the new ADT. Once registered with the system, an ADT could be used – just like a built-in type – in defining the type of an attribute of a relation. ADT functions could be used in queries and would be dynamically loaded as needed at runtime. This approach was pioneered by the ADT-Ingres effort at UC-Berkeley in the early 1980's [Ong84].[1]

In the mid-1980's, the Postgres project began as a follow-on to Ingres, initially laying out an approach to providing query optimizers with information about the properties of ADTs and their functions [Ston86a]. Another goal of Postgres was to provide support for storing and querying complex objects. Here, the Postgres project advocated a somewhat radical "procedure as a data type" approach [Ston86b], and rejected the CODASYL-like "pointer spaghetti" that they felt characterized systems that supported inter-object references (such as object-oriented database systems). Precomputation and query rewriting techniques were held up as possible approaches to avoiding the overheads that might otherwise cause problems for their procedure-centered proposal.

### 2.2 Persistent Programming Languages

A different approach for addressing the needs of complex, data-intensive applications was advocated by the programming language community: take the type system and programming model of an object-oriented programming language such as Smalltalk, CLOS, CLU, Trellis/Owl (a CLU descendent), or C++ and add features to make its data persistent and its program executions atomic. The argument for this approach was that some applications just need to manage permanent data, and would be happy with the imperative programming model of such a language if only its type system were available for use in constructing complex persistent data structures; in particular, such applications would benefit significantly from losing the "impedance mismatch" that arises at the boundary when a programming language type system meets a (relational) database type system. A good survey of the state of this area as of a decade ago, when objects were entering the persistent language scene, can be found in [Atki87].

[1] In many respects, the recent "Third Manifesto" of Darwen and Date [Darw95] seems to be some combination of a rediscovery and an elaboration of this approach.

Work in this area involved addressing alternative solutions to a number of problems: orthogonality (e.g., can any type be made persistent?), persistence models (e.g., persistence by reachability versus persistence by allocation), binding and namespace management for persistent roots, type systems and type safety, and alternative implementation techniques for supporting transparent navigation, maintenance, and garbage collection of persistent data structures.

### 2.3 Object-Oriented Database Systems

A more radical approach to addressing the perceived needs of non-traditional database applications, particularly engineering applications, emerged at this time in the database community: combine *all* of the features of a modern database system with those of an object-oriented programming language, yielding an object-oriented database (OODB) system. Three early OODB projects laid the foundation in this area – Gemstone [Cope84, Maie86], which was based on Smalltalk, Vbase [Andr87], which was based on a CLU-like language, and Orion [Bane87], which was based on CLOS. Again, a major motivation was to reduce or eliminate the impedance mismatch cited in our discussion of persistent programming language work. What distinguished the work on object-oriented databases from work on persistent languages was a focus on support for queries and indexing as well as navigation, as well as a focus on addressing the version management needs of engineering applications.

A decade earlier, in the early days of relational database systems, there was a single, clearly defined data model – relations, which were sets of tuples with simple attributes. Similarly, two competing relational query languages emerged early on – Quel and SQL. The early days of OODB systems were very different; there was no agreement on the details of the data model (e.g., on the underlying language type system), nor on a query model/language, nor on the version management features to be provided by such systems. However, there was a general agreement within the OODB community that this was the right direction to support engineering applications; there was also quite a bit of commonality among the approaches if viewed from the appropriate altitude. The emerging OODB revolution spawned work on many aspects of these systems, including data model details, query languages, indexing techniques, query optimization and processing techniques, system architectures, user interfaces, and pretty much every other aspect of database systems that one could readily imagine.

### 2.4 Database System Toolkits/Components

The last major approach proposed at the time was based on the belief that it was unlikely that any one type of DBMS would be able to meet the functionality and performance requirements of a broad range of next-generation applications. Instead, this camp advocated a different approach: provide a DBMS that can be extended at almost any level, e.g., an extensible DBMS, often based on a set of kernel facilities plus tools to aid developers in "rapidly" building a domain-appropriate DBMS. Members of this camp envisioned database systems specialized to application domains – documents would likely be managed by document-oriented database systems, while geographic data would be managed by geographic information systems. Such domain-appropriate DBMSs would likely have a number of fundamental differences, such as different query languages, different access methods, different storage organizations, and perhaps even different transaction mechanisms.

Key projects representing this approach were our EXODUS project [Care86b], the GENESIS project [Bato86], and the DASDBS project [Depp86]. EXODUS provided a storage manager for objects [Care86a] and provided a persistent programming language (E, based on C++) that was to be used for writing new access methods and query operators; it also provided a query optimizer generator for generating an optimizer for a domain-appropriate query language from a rule-based language specification. GENESIS provided a set of composable storage and indexing primitives and a "database system compiler" for assembling an appropriate storage manager from a specification. DASDBS provided a complex object storage manager with a novel, multi-layered transaction facility as a kernel upon which a domain-appropriate data model and query layer could be built; this layering was modeled after the RSS/RDS separation used in System R.

Another important project that started at this time was the IBM Starburst project [Schw86]. Starburst can be classified partly as a component-based DBMS, as support for new storage and indexing components was a major goal, but it can also be classified as an extended relational DBMS, as it was still centered on the relational model in terms of both its query language (SQL) and its models for query optimization and execution. One of the key goals of Starburst was to develop a clean architectural model to facilitate storage and indexing (and related) extensions; it also explored the use of a rule-based approach to providing an extensible query processing subsystem.

### 2.5 Summary

It should be clear that there was much turmoil in the database research community in 1986 – there were various competing approaches for leveraging object-oriented ideas in the database world, each with its own believers. Meanwhile, the commercial world was, for the most part, about ten years behind. Relational database system technology was finally maturing, in terms of commercial products, and relational systems were finally starting to be adopted for use in serious enterprise-scale applications. There were no extended relational, persistent programming language, or database toolkit products, and only two tiny OODB companies – Servio Logic (of Gemstone fame) and Ontologic (of Vbase fame, now Ontos) – who were attempting to market and refine their first product offerings. Over the next few years, panels at database research conferences debated the virtues and evils of objects and argued about how they should be utilized and packaged in the database world.

## 3 Objects & Databases in 1996

We now fast-forward to where we are today, roughly ten years after the initial object explosion in the database field. Among the different approaches being considered back then, which have died? Which appear to be wounded? What else has cropped up in the meantime? Which approaches still appear to be healthy and growing? Our view, in a nutshell, is as follows:

1. Two of the approaches, database system toolkits and persistent programming languages, generated a number of interesting results. However, both approaches have essentially proven to be dead-ends in a practical (commercial) sense.

2. Another of the approaches, object-oriented database systems, led to many research results from the academic community and product offerings from small startup companies. However, this approach has failed to live up to its original commercial expectations.

3. A new approach, based on generating language-specific object wrappers for relational databases, has emerged on the commercial database scene. This approach appears to be important – at least for now – for building object-oriented, client-side applications.

4. The last of the 1986 approaches, extended relational database systems, has been renamed. Object-relational database systems, as they are now called, appear likely to emerge as the ultimate winner in terms of providing objects for mainstream (enterprise) database applications.

In this section, we will look more closely at each of these points. We will also briefly touch upon some related developments, namely CORBA, OLE, Java, and middleware, that have appeared on the scene in the past few years.

### 3.1 Two Casualties

As mentioned above, one of the casualties of the past decade – perhaps ironically, given the reason for *this* paper – was the database toolkit approach. Toolkits such as EXODUS, GENESIS, and DASDBS have essentially fallen by the wayside; we are aware of no serious ongoing work in this area. One reason for this is that too much expertise ended up being required to use them; another is that each ended up being a bit too inflexible, awkward, or incomplete in certain dimensions of the database system design space. Still another reason is that object-oriented and object-relational database systems have managed to provide enough extensibility that it has not proven worthwhile for most builders of domain-oriented data management facilities to simply start from scratch, even given a toolkit to simplify the process.

To illustrate these problems, we look briefly at our own EXODUS experiences. EXODUS provided a full-function client/server storage manager for handling object storage, a persistent variant of C++ to simplify the management of objects as well as the construction of new access methods and query operations, and a rule-based query optimizer generator to simplify the development of efficient query processors. A number of other research projects, plus one startup company that we know of, made use of the EXODUS storage manager. One common problem was that most wanted to use EXODUS to implement their own object servers; thus, the EXODUS storage manager's client/server architecture tended to get in the way, introducing an unwanted level of indirection for their systems [Care94]. Some projects also made use of the E programming language, but "serious" database implementors would have preferred more control over low-level details (e.g., buffering, concurrency, recovery), and application-oriented programmers found it a bit too low-level (no collections, queries, etc.). The EXODUS query optimizer generator was very general, but it was inefficient and too hard to use; it still left too much (e.g., predicates and logical query rewrites) to the implementor when applied to a full-function language such as SQL.

We ourselves put EXODUS to the test on what would now be characterized as an object-relational data model (EXTRA) and query language (EXCESS) that we designed mid-way through the EXODUS project [Care88]. To make a long story short, we found that there was still way too much to do in building such a system to declare EXODUS as having succeeded in that regard. Luckily, the good news is that the EXODUS project nonetheless managed to produce a number of interesting research by-products – including system design, implementation, and performance studies of direct relevance to object-oriented and object-relational database systems – as well as a steady stream of first-rate graduate students.

We have also characterized the persistent programming language area as one of the casualties of the decade-long object wars. Unlike toolkits, this area of research is still active, at least in academia. Our characterization stems from the fact that we are aware of no commercial implementation of what could properly be characterized as a pure persistent programming language. This bad news is balanced by good news, however – as work in persistent languages has had a significant impact on the navigational programming interfaces of many of today's object-oriented database products. In addition, the results from this area on topics such as persistence models, pointer swizzling schemes, and garbage collection schemes for persistent data have also been directly transferable to object-oriented databases. But, there's more bad news, as the next subsection of this paper will discuss – because the transfer target, the object-oriented database field, has not expanded commercially at the rate or in the way that its founders and contributors had expected.

### 3.2 Object-Oriented Database Systems

A tremendous amount has happened in the object-oriented database area in the past decade. One early milestone, which helped to focus both research and development, was a general prescription – developed by a collection of leading database system and language researchers – for what constitutes an object-oriented database system [Atki90]. It was agreed that such systems must support all of the following: complex objects, object identity, encapsulation, inheritance and substitutability, late binding, computationally complete methods, an extensible type system, persistence, secondary storage management, concurrency control, recovery, and ad hoc queries. Optionally, they might also choose to support: multiple (versus only single) inheritance, static versus dynamic type checking, distribution, long transactions, and version management. Open to individual choice were: the programming paradigm (and language choice, obviously), the exact details of the type system, the degree of fanciness of the type system (e.g., templates), and the degree of uniformity (or purity) of the object model.

Much research energy has been expended in the object-oriented database area, and many interesting results have been produced. A variety of data model issues have been examined, including basic object models, support for composite objects, and schema evolution. Quite a few object-oriented query language proposals have appeared, as have a number of papers on the query processing issues for such languages (e.g., handling of path expressions and nesting). Many schemes have been designed for indexing object-oriented databases, addressing issues such as efficient handling of path expressions (and updates along paths) and queries over portions of a class hierarchy. Pointer-based join methods and complex object assembly schemes have been studied for queries over large object bases. Alternative client/server architectures have been proposed and studied, including schemes for transactional data caching, and several client/server crash recovery algorithms have been designed for use in the data-shipping environments common to object-oriented database systems. A series of OODB benchmarks have been published and used to characterize the performance of such systems. Finally, a number of version and configuration management models have been proposed and implemented.

On the systems side, several other significant systems followed the three ground-breaking prototypes; important later systems included O2, ObjectStore, and ODE, all of which have had significant degrees of OODB research impact. There are numerous commercial OODB products available on the market today; current players in the OODB marketplace include GemStone, Objectivity, ObjectStore, Ontos, O2, Poet, and Versant. In addition, a consortium of the OODB product vendors banded together – under the leadership of Rick Cattell of SunSoft – in the early 1990's and formed the Object Database Management Group (ODMG). This group has worked to draft OODB standards for an object data language (ODL), an object query language (OQL), and a C++ programming interface for manipulating and querying object databases. The latest version of their specification, called ODMG-93 Release 1.2, was published earlier this year [Catt96].

With all that research, and all these companies – plus a draft standard – what could possibly be wrong? Several things. First, despite a decade of hard work, it has been nearly impossible to gain complete agreement on anything having to do with object-oriented database systems. Today, there are still differences between many of the OODB products in terms of details of their programming interfaces, implementation twists, and query support. Although the ODMG standard has been out in some form for approximately three years, vendors do not truly support it as a standard – it has been divided into pieces, corresponding to the chapters of the standard, and many vendors are choosing to support only this piece or that piece. As an example, we are aware of only one vendor (O2) who supports the object query portion of the standard (OQL). Second, object-oriented database products are still quite a bit behind relational database products in some respects – e.g., none provides a view facility, and in fact the prerequisite research in that area is still unfinished in our opinion. Schema evolution is much more painful in the OODB world, and many of the OODB products still rely on a CODASYL-like schema/application compilation cycle. The coupling between an OODB and its application programming language tends to be tight; most are single-language (most commonly C++) systems for all intents and purposes. In addition, the robustness, scalability, and fault-tolerance of most OODB products still cannot match those of relational database products.

Other problems have to do with the availability of application development tools and how client/server computing environments have evolved. In the tools area, there are obviously fewer end-user tools and application development tools available in the object-oriented database world. Such tools are widely used today for application development in the relational world. Moreover, PC-based applications talking ODBC to relational servers have emerged as a common architecture for database applications; this has dramatically reduced the number of programmers writing lower-level database code (e.g., embedded SQL). This in turn has diminished the impact of the impedance mismatch, and has also led the dominant computing environment to be one with thin clients and fat servers – which is the opposite of the design point for object-oriented database systems. In response, most OODB system vendors have developed ODBC connectivity solutions, but such solutions cannot exploit the object-oriented features of their underlying database systems.

Partly as a result of the aforementioned problems, the commercial OODB market has grown quite a bit more slowly than expected. Some of the expected consumers of OODB technology, e.g., CAD system vendors, have been slower than expected to move away from relying on files to store their data. Meanwhile, in the commercial database world, some large enterprises have barely finished embracing relational technology wholeheartedly, and are therefore not anxious to undertake yet another major paradigm shift.

### 3.3 Object-Relational Database Systems

In parallel with the explosion of work in the object-oriented database system area, extended relational database systems have matured. Products are available today from several vendors (e.g., CA-Ingres, IBM, Illustra, and UniSQL). In fact, over time these systems have adopted some of the more attractive data model and query language features from the OODB world; this trend will no doubt continue.

The path that object-relational database systems (as extended relational database systems are now known [Ston96]) have followed was foretold in a document drafted by a different set of leading database researchers [Comm90] in response to the "OODB Manifesto" cited earlier [Atki90]. This document gave three main tenets for so-called "third-generation" database systems: provide support for richer object structures and rules, subsume second generation (i.e., relational) DBMSs, and be open to other subsystems, e.g., tools and multidatabase middleware products. It then laid out a set of more detailed propositions about what third-generation database systems should provide: a rich type system, inheritance, functions and encapsulation, optional unique ids, and rules/triggers; a high-level query-based interface, stored and virtual collections, updatable views, and separation of data model and performance features; accessibility from multiple languages, layered persistence-oriented language bindings, SQL support, and a query-shipping client/server interface.[2]

Object-relational systems differ from object-oriented database systems in many of the above ways. They start with the relational model and its query language, SQL, and build from there. In terms of object features, current (early!) products provide support for two types of objects – *ADTs*, a la Section 2.1, and *row types* (or *composite types*). ADTs are user-defined base types, as discussed earlier. Their role is to enable the set of built-in types of the DBMS to be extended with new data types such as text, image, audio, video, time series, point, line, polygon, and so on. They enable the DBMS to manage new kinds of facts about the entities in the enterprise that the database is intended to model; e.g., an Employee can have a resume and a photograph in addition to a name and a salary.

Row types are a direct and natural extension of the type system for tuples. They make it possible for rows in tables to enjoy object-like properties (such as named types, and functions/methods). In addition to other base type attributes, "row objects" are permitted to contain reference-valued attributes. Model-wise, such typed row references (e.g., *ref(Dept)* are treated as an-

[2] Another paper that we would recommend to interested readers is [Kim93]; it explains how object-oriented and relational database technologies should be combined from the perspective of UniSQL's founder.

other flavor of base type.[3] Also supported are multi-valued attributes, i.e., attributes whose values can be sets, bags, arrays, or lists of base type elements.[4] Lastly, inheritance is also supported to enable natural variations among row types to be captured in the schema (e.g., Persons, Students, Employees, and WorkStudyStudents have much in common). The topmost level of an object-relational database schema is still a collection of named relations. However, the objects in the relations can now be as rich as those supported by OODB systems. SQL extensions for object queries include such features as path expressions, method-like function invocation syntax, and support for nested sets in the *from*-clause.

As for OODB systems, the past decade has seen both research results and work on prototypes of object-relational database systems. Since they build upon relational database technology, their basic foundation already existed. Many of the required system extensions have been explored in the contexts of the Postgres project at UC Berkeley, the EXTRA/EXCESS effort within the EXODUS project at Wisconsin, the Starburst project at IBM Almaden, and the current Paradise project at Wisconsin. Moreover, in some dimensions, such as object query support, object-relational systems will be able to benefit rather directly from work in the OODB area. As mentioned above, vendors are already offering products with degrees of object-relational functionality: IBM's DB2/CS V2 system supports user-defined base types and functions, rules, and large objects, as does the CA-Ingres system (which was the first commercial system to offer those features). UniSQL does not support user-defined base types, but it does provide support for row objects and inheritance (including view support, in fact). Illustra is currently the most complete product, functionality-wise, in the object/relational market; it supports all of these features in some form. In addition to these server products, vendors are starting to market ready-made, ADT-based type extension packages for managing data types such as image and text (e.g., Illustra's DataBlades and IBM's Database Extenders); some predict that these add-on packages will be the primary early driver for the acceptance of object-relational database technology.

Object-relational database systems today suffer to some extent from the same problem that plagues OODB systems – there are too many differences from vendor to vendor. However, the SQL3 standards effort is working hard to standardize most of these features; it has included an ADT concept for some time, and was amended recently to include support for row objects and references as well. Moreover, unlike the OODB marketplace, the major database vendors are all pushing in this direction (and are very concerned about standards). Among the major vendors, IBM currently provides some significant object-relational features and is working on implementing further extensions; Informix recently purchased Illustra, and is promising customers a merged "universal server" product in late 1996; and, Oracle is promising that Version 8 will be out, with substantial object support, in a similar timeframe.

### 3.4 Object-Oriented Client Wrappers

In addition to the approaches being actively studied a decade ago, another approach has recently been gaining favor in the commercial world – the use of object wrappers for relational databases to support the development of object-oriented, client-side applications working against legacy databases. A number of vendors offer such products today, including Persistence Software, Ontologic, HP, Next, and others. Most of these products are language-specific; they generate either C++ or Smalltalk classes that act as proxies for data in the underlying database, permitting programmers to interact with the data in a more natural way for their programming tools. They come with tools to aid developers in defining/constructing objects from the underlying database, and rely on key-to-OID mappings to maintain correspondences between programming objects and database data.

The attractiveness of the wrapper approach is that it enables object-oriented applications to be written *today* against enterprise data, making an object-oriented design methodology available for implementing business objects. There are disadvantages as well, however. One is that these products tend to be very weak on the query side, requiring ad hoc queries to be posed in SQL against the underlying relational schema; this creates a paradigm mismatch for application programming and querying. Another is that they force both design and paradigm choices when it comes to representing business logic – should one utilize the underlying database system's triggers, procedures, and constraints to enforce data integrity, or code the business rules and procedures outside the DBMS in C++ or Smalltalk on the client side? The need to make such a decision is an unfortunate reality of 1996.

[3] Notice that object-relational database systems have ended up adopting the "pointer spaghetti" approach that Postgres worked to avoid, but that were advocated and studied elsewhere, e.g., [Zani83, Care88].

[4] It is interesting to note that the foundation for many of the row type extensions predates the initial object revolution by several years, having been laid in part by efforts such as Daplex [Ship81] and GEM [Zani83].

### 3.5 Related Developments

Before moving on to what lies ahead, we must mention that there have been a number of related developments in the object area that are likely to have some impact on the database world. Namely, technologies such as CORBA, OLE, and Java have been attracting much industry attention. Another recent trend is the growing importance of database middleware. We will discuss each of these technologies briefly here.

The CORBA standards, developed by the Object Management Group (OMG, not to be confused with ODMG!), are focused on solving problems that arise when developing large, distributed, object-oriented applications. Their biggest success has been in defining the standards for an interoperable object RPC mechanism; in addition, they have developed useful standards for services like registering and locating named resources in a distributed environment. We expect that CORBA will continue to be important in these respects. In addition, OMG has attempted to define a number of factorable object services, including a persistence service, a collection service, an indexing service, a transaction service, and so on. Here, we predict that OMG will fail, as years of database research have not been able to separate the majority of these services (e.g., indexing and transactions, or collections and queries) in a manner capable of providing anything approaching reasonable performance. Also, there are some who advocate the use of CORBA for fine-grain access to database data, making each database object a CORBA object. We expect such approaches will perform poorly and will ultimately fail as well. CORBA should stick to coarse-grained object RPC and related support services in our opinion.

Another set of de facto object standards are Microsoft interfaces such as OLE and its underlying object model, COM and Distributed COM (the Microsoft answer to CORBA). Given their source, it goes without saying that these standards cannot be ignored. OLE is a key technology for those who build and manage desktop data in the Wintel world; all major vendors are working to integrate OLE support into their database engines and tools, and it is clear that support for OLE/COM ADTs will be important in the future. Also looming on the horizon is Microsoft's OLE DB work [Blak96], which offers an approach to separating query optimization from execution in a world where data lives elsewhere in addition to databases. We will comment further on OLE DB in Section 4.2.4.

Obviously, no discussion of object trends would be complete without touching upon the Java furor that has recently been sweeping the computer industry for the past year or two. Java is essentially a safe subset of C++ together with a standard, machine-independent, pcode-like representation for executable Java programs. Java was designed to enable safety checks and guarantees that make shipping Java code (applets) around the Internet both possible and reasonable; this is the reason for the current Java furor. How will Java impact database systems? We will mention Java again in Section 4, but certain potential impacts are clear – for example, Java would be an ideal language for writing ADTs that can be executed on either the server side or the client side of the database world.

One last trend of importance is the growing market for database middleware products – products that provide a uniform interface to multiple backend database systems. On the client side, the best example today is undoubtedly Microsoft's Access product, which provides relational query access to any backend DBMS that speaks ODBC, and which permits queries that draw data from multiple backends. As another example, on the server side, IBM's DataJoiner product provides a full-function relational DBMS engine, with facilities for accessing a variety of backend database products, plus a cost-based, distributed query optimizer. An early object-relational offering in this area, called UniSQL/M, is available as a middleware version of the UniSQL system. Middleware query products such as these – and there are a number of relational products available – lie in what is expected to be one of the fastest growing segments of the database market.

## 4 Objects & Databases in 2006

We began by looking at the past, and now we have seen the present. But what does the future hold for objects and databases? In this section, we share our vision in that regard. Since predicting the future is always difficult, we will "cheat" – by describing what *commercial* database products will look like (if done "right," i.e., our way!). Since the latency from research prototypes to robust products is often on the order of 5-10 years, that makes our job here easier. After describing our vision for the next decade's database products, we will discuss the challenges that we – as researchers – now face in helping the field to get there from here.

So what will the database solution of the year 2006 look like? We envision large enterprises reaping the benefits of families of products that offer...

### 4.1 A Fully Integrated Solution

We predict that object-relational database systems will mature, and will end up delivering – scalably and robustly – most of what object-oriented database systems have been promising to deliver. Object-relational servers will provide full support for object-oriented

ADTs, including inheritance among ADTs and the ability to implement them in any of a number of programming languages. They will also provide full object-oriented support for row types, with the extended SQL features in this area being integrated with all of SQL's important other features – including object-oriented views, authorization, triggers, constraints, and so on – as per the SQL standard of the day (SQL4 or SQL5). To support middle-tier and desktop applications well, these servers will work together with high-function, object-oriented, caching client front-ends to provide a development environment where the same object model is used to describe a database at all levels, both for querying and for navigational programming. Methods will be able to run on cached data at the client, or on the server, depending on which is cheaper; likewise for queries and fragments thereof. And, the same will be true for triggers, referential integrity constraints, and other types of constraints – the business rules of the year 2006 will be specified and implemented just once, in SQL, with methods written in SQL and/or the imperative object-oriented language of choice, and will simply run wherever it makes the most sense for them to run.

Where does this leave object-oriented databases as we know and love them today? They will probably remain niche solutions – e.g., embedded within prepackaged solution packages for problems in areas like engineering design, telecommunications, on-line trading, and web page management – serving applications that demand a level of seamlessness and high-performance, for moderate databases, that a more heavyweight object-relational solution cannot address as effectively. This niche may shrink over time, especially if the object-relational vendors offer "lite" versions of their products. What about object-oriented client wrappers – where does this leave them? They will have been the first step in the client-side direction that we have sketched out above. Since the server will be object-relational in its data model, they will have much less object-mapping work to do; their job will be to map object-relational concepts into Java, Smalltalk, C++, or another object-oriented language of choice. And, they will be more tightly integrated with the engine, in the sense that they will assist in the early, client-side enforcement of business rules and execution of business logic. They will still cache data and updates, but they will have to become more sophisticated in order to cooperatively process queries.

### 4.2 Research Challenges

If one believes this view of the world – and we certainly do – a number of problems remain to be solved in order to get to the year 2006 from here. Areas needing work include server functionality and algorithm improvements, integrated clients, parallelization, and provisions for legacy data access. In addition, the world of database standards will need to fill certain holes in order to realize the full potential of our vision. We briefly explain the open problems in each of these areas.

#### 4.2.1 Server Functionality & Performance

Object-relational servers will continue to evolve from today's SQL-based relational servers and early object-relational servers. Research is still needed on object query processing; we believe that this work will need to draw more fully on the large body of work on SQL query processing from the world of relational databases in order to yield industrial-strength solutions in the year 2006. We did some initial object-relational benchmarking work for a consulting client of ours a little over a year ago[5], and it was clear from our work that there is still "room for improvement" in today's systems. Also, applied research is needed to complete the task of "object-ifying" SQL in the best possible way. Some of this can be drawn from work on OQL and similar languages; other work, e.g., on properly extending SQL's support for views, updates, constraints, triggers, and so on, remains to be done. Object-relational servers of the future can also draw upon work done in the object-oriented database world on topics such as path indices and object clustering. On the ADT side, an important open problem is support for extensible access methods – this has been talked about since the early days of the object revolution, but there is still no ideal solution in sight. Industrial-strength support for new data types will demand solutions to this problem.

#### 4.2.2 Client Integration

As mentioned above, the solution of the year 2006 will include a highly integrated client component. At the surface of this component, good mappings and programming interfaces will be needed to serve object-relational objects to programs in C++, Smalltalk, Java, and other such languages – with fully integrated object query support in addition to navigation. Open problems in this area include querying over the cache plus the database in an intelligent way (e.g., not just flushing the cache or else shipping all objects to the client, as most systems do today). Cached objects can come from base tables or views; the same caching programming interface must be available for both, and must work correctly in the face of updates. Updates

[5] This benchmarking work was performed while the first author was employed at the University of Wisconsin-Madison. We are hoping to publish the benchmark's design in the not-too-distant future.

on the client side must trigger the appropriate triggers in a timely, consistent manner so that the client/server boundary becomes only a performance boundary and not a wall separating two disjoint worlds. The same must be true for the enforcement of constraints of all types. Methods must execute properly on either side, client or server; perhaps Java will help here. The solutions to these many problems are not obvious, given the interfaces that servers provide today; one potential solution might be for servers to provide various "cooperation hooks" that can be exploited by high-function client tools.

### 4.2.3 Parallelization

Another place where research is needed is parallelism. Parallel database systems today can successfully parallelize query execution for relational queries. What of object-relational queries? Little has been done in this area, and since large enterprises need parallel database systems, this will be important to the ultimate success of object-relational database systems. To the extent that relational query processing technology can be extended to object queries, the same should be true of parallel execution techniques. However, some potentially sticky issues await, most arising from the fact that many important ADTs, such as large multimedia (image, video, and audio) and GIS data types, will involve expensive operations that should themselves somehow be parallelized – otherwise, load imbalances and large execution times for queries over these types will plague large enterprises in 2006.[6] Parallelizing ADT operations on these and other interesting data types – from both I/O and CPU perspectives – is an open problem, as is providing a framework to make this easier to do. Some of these issues are currently being explored for image and GIS data types in the context of the Paradise project at Wisconsin [DeWi94].

### 4.2.4 Legacy Data Sources

Although it has always been our fond hope that databases will someday achieve world domination, we recognize that solutions requiring all the world's data to be moved into a database system will never be universally accepted. As a result, another important area of research has to do with providing access to legacy data sources – in older database systems (both relational and pre-relational) and in other kinds of systems, such as document or image management systems. We envision a middleware solution here – we believe that a promising approach to solving this problem is to place a distributed, object-relational query engine in between end users and their disparate data sources, with the goal being to make the data *look* as if it were stored in a centralized object-relational database system. This approach would make all the data available through a common query interface, and would also make available all of the nice client-side tools that we expect object-relational database systems to support. The result is a three-layer architecture, with clients on top and a wide variety of data sources on the bottom, including object-relational DBMSs, relational DBMSs, pre-relational DBMSs (e.g., IMS), and various non-database data sources; in the middle, providing the "glue," is an object-relational middleware query engine.

Research issues related to this approach to legacy data access include distributed query optimization for a mix of object-relational, relational, and other ("dumber") data sources; effective handling of semi-structured data (e.g., structured documents and web pages); query semantics and query processing for data sources that yield relevance-ranked results; and so on. This approach to legacy data is being explored in the Garlic project at the IBM Almaden Research Center [Care95], while semi-structured data is a focus of the TSIMMIS project at Stanford [Garc95]. Meanwhile, the Microsoft OLE DB work [Blak96] is also addressing this problem space, e.g., by defining the protocol for a query processor to use in talking to non-database data sources.

### 4.2.5 Standards

In the area of standards, SQL3 is moving in the direction that we have outlined, and it is drawing on OQL for inspiration in some areas. However, we are concerned that certain important areas (where standards are a must) are currently *not* being addressed anywhere – so we expect to see work in this area between now and 2006. One key example is in the area of ADTs, particularly those ADTs defined in external programming languages. To process queries involving such ADTs well, an object-relational query processor must be aware of information about properties of the type and of its operations, selectivity estimates, and function costs [Ston96]. Object-relational database systems bring an opportunity for third-party vendors to provide libraries of ADTs (and also row types) that address certain problem domains – i.e., instead of providing special-purpose data managers, future domain-specialists could make a business out of providing extensions for object-relational systems. The situation today is that each vendor who supports user-defined data types has their own interface for defining and registering these types – fortunately, SQL3 is addressing this. Unfortunately, each such vendor also has differ-

[6] For some initial thoughts on the difficulties associated with parallelizing an object-relational DBMS, see `http://www.cs.wisc.edu/~dewitt/dewitt.html/vldbsum.ps`.

ent provisions for the additional information needed for query optimization, and SQL3 is not addressing this aspect of the ADT registration process. The ultimate success of a third-party data type market will depend on the emergence of suitable standards in this area. A related area where we would like to see an eventual standard is the access method interface, so that the same vendors could be the providers of appropriate index structures for the data types in which they specialize.

There are other areas where standards would be beneficial as well in the year 2006. One is in the area of client/server interfaces. It would be nice if the cooperative client/server interface alluded to above could be defined in a standard way, enabling multiple providers and competition in this area (rather than requiring each database vendor to provide full, top-to-bottom solutions). As mentioned in the previous subsection, standardizing on an interface for query tools to use in accessing non-database sources could also be beneficial. Finally – though not necessary for success – wouldn't it be nice if we could someday have a fresh new query language standard, where some of the legacy design quirks of SQL could simply be left behind rather than supported forever?

## 5 Conclusions

In this paper, we have taken a look (albeit biased!) at what has transpired in the area of objects and databases in the period from 1986-1996. We have explained why we believe that some approaches have fared better than others, and we have tried to predict the future, at least commercially. We believe that we are on the verge of an era where object-relational database systems will begin taking over the enterprise, and that their ultimate success will be due to the work – past, present, and future – that our community has been doing in areas including extensibility, object data models and query languages, persistent languages, object mapping, etc. While not all of the approaches themselves have, or will, survive, we believe that many of their results will prove to have a lasting impact on the shape of the highly integrated, client/server, object-relational database solutions of the year 2006. To encourage more work towards this goal, we have attempted to identify some of the key research and development challenges that we believe lie ahead if this takeover is to be successful over the next ten years. The future of objects and databases appears bright, yet much is left to do...

## 6 Acknowledgements

Joel Richardson and Eugene Shekita were co-authors of the 1986 VLDB paper [Care86a] that led to our being invited to write this paper. We wish to thank them and the many other superb University of Wisconsin students, staff, and faculty colleagues who we've had the pleasure of working with on projects related to the topic of this paper. We would also like to thank our industrial colleagues, especially those at IBM Almaden and IBM Santa Teresa, for many interesting discussions on these and related issues. Finally, we would like to thank Nelson Mattos of IBM for reading and providing helpful feedback on an earlier draft of this paper.

## References

[Andr87] T. Andrews and C. Harris, "Combining Language and Database Advances in an Object-Oriented Development Environment," *Proc. 1987 ACM OOPSLA Conference*, Orlando, FL, Oct. 1987.

[Atki87] M. Atkinson and P. Buneman, "Types and Persistence in Database Programming Languages," *ACM Computing Surveys* 19(2), June 1987.

[Atki90] M. Atkinson *et al*, "The Object-Oriented Database System Manifesto," *Proc. 1st DOOD Conf.*, Kyoto, Japan, 1989.

[Bane87] J. Banerjee *et al*, "Data Model Issues for Object-Oriented Applications," *ACM Trans. on Office Info. Sys.* 5(1), Jan. 1987.

[Bato86] D. Batory *et al*, "GENESIS: A Project to Develop an Extensible Database Management System," in [Ditt86].

[Blak96] J. Blakeley, "Data Access for the Masses through OLE DB," *Proc. 1996 ACM SIGMOD Conference*, Montreal, Canada, June 1996.

[Care86a] M. Carey, D. DeWitt, J. Richardson, and E. Shekita, "Object and File Management in the EXODUS Extensible Database System," *Proc. 12th VLDB Conf.*, Kyoto, Japan, Aug. 1986.

[Care86b] M. Carey *et al*, "The Architecture of the EXODUS Extensible DBMS," in [Ditt86].

[Care88] M. Carey, D. DeWitt, and S. Vandenberg, "A Data Model and Query Language for EXODUS," *Proc. 1988 ACM SIGMOD Conference*, Chicago, IL, June 1988.

[Care94] M. Carey *et al*, "Shoring Up Persistent Applications," *Proc. 1994 ACM SIGMOD Conf.*, Minneapolis, MN, May 1994.

[Care95] M. Carey *et al*, "Towards Heterogeneous Multimedia Information Systems: The Garlic Approach," *Proc. 1995 IEEE RIDE Workshop*, Taipei, Taiwan, March 1995.

[Catt96] R. Cattell (ed.), *The Object Database Standard: ODMG-93 (Release 1.2)*, Morgan Kaufman Publishers, 1996.

[Comm90] Committee for Advanced DBMS Function, "Third-Generation Database System Manifesto," *SIGMOD Record* 19(3), July 1990.

[Cope84] G. Copeland and D. Maier, "Making Smalltalk a Database System,", *Proc. 1984 ACM SIGMOD Conference*, Boston, MA, June 1984.

[Darw95] H. Darwen and C. Date, "The Third Manifesto," *SIGMOD Record* 24(1), March 1995.

[Depp86] U. Deppisch, H. Paul, and J. Schek, "A Storage System for Complex Objects," in [Ditt86].

[DeWi94] D. DeWitt, N. Kabra, J. Luo, J Patel, and J. Yu, "Client-Server Paradise," *Proc. 20th VLDB Conference*, Santiago, Chile, Sept. 1994.

[Ditt86] K. Dittrich and U. Dayal (eds.), *Proc. 1st Int'l. Workshop on Object-Oriented Database Systems*, Pacific Grove, CA, Sept. 1986.

[Garc95] H. Garcia-Molina *et al*, "The TSIMMIS Approach to Mediation: Data Models and Languages," *Proc. 2nd Int'l. Workshop on Next Generation Info. Technologies and Systems*, Naharia, Israel, June 1995.

[Kim93] W. Kim, "Object-Oriented Database Systems: Promises, Reality, and Future," *Proc. 19th VLDB Conference*, Dublin, Ireland, August 1993.

[Kim95] W. Kim (Ed.), *Modern Database Systems: The Object Model, Interoperability, and Beyond*, ACM Press, 1995.

[Maie86] D. Maier *et al*, "Development of an Object-Oriented DBMS,", *Proc. 1986 ACM OOPSLA Conference*, Portland, OR, Oct. 1986.

[Ong84] J. Ong, D. Fogg, and M. Stonebraker, "Implementation of Data Abstraction in the Relational Database System Ingres," *SIGMOD Record 14(1)*, March 1984.

[Schw86] P. Schwarz *et al*, "Extensibility in the Starburst Database System," in [Ditt86].

[Ship81] D. Shipman, "The Functional Data Model and the Data Language DAPLEX," *ACM Trans. on Database Sys.* 6(1), March 1981.

[Ston86a] M. Stonebraker, "Inclusion of New Types in Relational Data Base Systems," *Proc. 2nd IEEE Data Eng. Conf.*, Los Angeles, CA, Feb. 1986.

[Ston86b] M. Stonebraker, "Object Management in Postgres Using Procedures," in [Ditt86].

[Ston94] M. Stonebraker, *Readings in Database Systems (2nd Edition)*, Morgan Kaufmann Publishers, 1994.

[Ston96] M. Stonebraker, *Object-Relational DBMSs: The Next Great Wave*, Morgan Kaufmann Publishers, 1996.

[Zani83] Carlo Zaniolo, "The Database Language GEM," *Proc. 1983 ACM SIGMOD Conference*, San Jose, CA, May 1983.

[Zdon90] Z. Zdonik and D. Maier, *Readings in Object-Oriented Database Systems*, Morgan Kaufmann Publishers, 1990.