Object Database Management Systems: An Overview

Akmal B. Chaudhri[1]
The City University, London

> *"Consider all the classes that are not members of themselves, such as the class of dogs, which is not itself a dog, and try to combine them into a big class of their own, the class of classes that are not self-members: then you have the result that this class is a member of itself only if it is not and is not only if it is. Contradiction! Red Alert!"*
>
> (Russell Paradox)
>
> Advertisement for London Review of Books, London Underground.

Abstract

The aim of this paper is to provide a broad overview of Object Database Management Systems (ODBMSs). It discusses the shortcomings of existing database technology to meet the requirements for "next-generation" applications that require support for richer data types and complex structures. Object and data management requirements for ODBMSs are also briefly reviewed. Six major classification models are presented including *Data Models*, *Architectures*, *Storage Server Models*, *Alternative Object Database Strategies*, *Language Data Models*, and *Evolution versus Revolution*. Areas for standards are briefly considered and, finally, the main strengths and weaknesses are discussed.

1. Introduction

The object paradigm has been applied to programming languages, user interfaces, design methodologies, operating systems, hardware architecture, and database systems. This has been prompted by the desire to control complexity and harness the expanding system environment (e.g. multimedia information, end-user computing, distributed processing) into more useful and exciting applications [Winbl90]. These "next-generation" applications include (the often cited) Computer Aided Software Engineering (CASE), Computer Integrated Manufacturing (CIM), Computer Aided Instruction (CAI), Computer Aided Publishing (CAP), and other (CAx) applications.

For several years now, there has been intense research activity in many of these areas, resulting in the publication of numerous articles and papers. As a result of this research, many new products have also appeared.

Within the field of databases, some authors have also noted that the "everything with objects" approach has been permeating the research and commercial database community. This has resulted in articles such as "My Cat is Object-Oriented" [King89].

[1] Present address: Systems Architecture Research Centre (SARC), Computer Science Department, The City University, Northampton Square, London EC1V 0HB, United Kingdom, (email: akmal@cs.city.ac.uk).

It has been noted that there are difficulties in using existing relational technology for these new applications areas:

> *"Everyone agrees that traditional relational database systems do great on business data processing, and lay an egg if you ask them to do anything else ... If you want them to store documents or CAD-related information, they just don't do very well."*
> Michael Stonebraker, cited in [Hazza90]

> *"You can do anything with an RDBMS that you can with an OODBMS, except you have to roll your own."*
> Mark Hanner, cited in [Hodge89]

[Codd92] has suggested that relational technology can form the *foundation* for these new application areas. This may be possible with extended relational (or non-first normal form) models, which provide better support for complex data by allowing column attributes themselves to be vectors (arrays, lists, tables). However, it is difficult to see how the semantics of multimedia data types (for example) can be adequately captured by the relational model, since it cannot support this type of complex data - largely a by-product of relational theory according to [Hazza90].

Object Database Management Systems (ODBMSs), also variously referred to as Object-Oriented Databases (OODBs), Object-Oriented Database Management Systems (OODBMS) or Next-Generation Database Systems, try to address some of the shortcomings of existing (relational) database technology. Briefly, some of these shortcomings are now discussed.

1.1 Lack of Expressive Power

The only data structure (aggregate or object) in the relational model is the table. This reduces all data to flat two-dimensional form, with relationships between tables being dynamically re-imposed at run-time. The lack of a facility for establishing relationships between tables at design time is probably the main backwards step from the structured database [White89]. [Khosh90a] have also noted that the *"semantic eloquence of complex object composition is lost"* when real-world objects are mapped to tables, resulting in the so-called "semantic gap."

1.2 Simple Data Types

To support next-generation applications a richer set of data types is required (e.g. arrays, lists, multimedia data types).

Some popular RDBMS vendors have enhanced their products to support new data types [Compu91b]. Examples include INGRES that allows the incorporation of user-defined Abstract Data Types (ADTs) into database tables. However, these enhancements are still not comparable to the facilities provided by Extended Relational Database Systems such as POSTGRES [Rowe90; Stone90; Stone91].

1.3 Loss of Data Protection

Most RDBMSs support simple data types (e.g. integer, char). Many programming languages, however, enable new enumerated types to be defined. When these enumerated types need to be mapped to the database types, data protection is lost [Hughe91].

1.4 Impedance Mismatch

> *"Impedance mismatch ... the metaphor comes from the field of electrical engineering, and refers to the fact that an impedance mismatch in an electrical circuit will prevent the maximum power transfer from being achieved."*
> [Gray92]

Mapping data structures between programs and the database can account for a considerable amount of code - figures of up to 30% have been suggested [Atkin83]. There are several reasons why this mapping is necessary.

Firstly, SQL is relationally complete (it supports the operations select, project and join), but is computationally incomplete (in the sense of a programming language such as C, Pascal, etc.).

Secondly, computer programming languages such as C, Pascal, etc. are procedural and record-oriented, whilst database query languages such as SQL are more declarative and set-oriented.

Thirdly, computer programming languages support rich data types (e.g. arrays) whilst SQL is a closed language, operating on two-dimensional tables with simple types (e.g. integer, char).

This mix of paradigms (SQL and host language) results in this "impedance mismatch."

However, the mismatch aside, some attempts have been undertaken to extend SQL to provide more functionality and support for object concepts. Examples include Intelligent SQL [Khosh90b], Objects in SQL [Beech90], and the current efforts of the ANSI X3H2 committee (SQL3).

1.5 Performance

Within the literature, it is widely accepted that certain classes of applications are particularly well suited to ODBMSs (the ubiquitous CAD, CAM, CIM, GIS, Multimedia, etc.). These are primarily applications involving the storage and manipulation of complex and heavily inter-related data. Traditionally, RDBMSs have been unable to provide the necessary performance for applications with complex relationships and multi-way joins, as discussed by [Maier89].

By its very nature, the relational model "flattens" entities. Putting these entities together in a meaningful way requires joining (described as "unnatural" in [Khosh90a]) and sorting. These operations compete for being the slowest in relational systems [Loomi92]. In addition, commercial language optimisers are often unable to cope effectively with queries involving large numbers of joins [Gray92], which has resulted in static binding strategies being developed by some relational database vendors (e.g. IBM’s DB2).

In many ODBMSs, the equivalent of relational joins are "pre-computed" through the use of explicit pointers. Additionally, since ODBMSs can store methods, this incremental knowledge about the operations being performed provides the ability to better control the concurrent execution of transactions (and hence provide better performance). This is possible

since, as [Dawso89] has noted, the operations are not simply reads or writes, but have more semantics, as illustrated by the following example.

For a queue data type, it is possible to have operators such as **enque** and **deque**. From one point of view these can be viewed as write and read operations respectively. However, when considering the special semantics of these operators, a higher degree of concurrency can be achieved.

Consider a queue object Q and two transactions, T1 and T2. If T1 wished to perform an enque on Q, then T2 would be prevented from performing a deque by common read/write semantics until T1 had committed. However, for non-empty queues, it is possible to execute both operations without conflict, since they do not affect each other's results.

[Atwoo92] has also noted that in an ODBMS, the operations that objects of a given type are capable of performing are known to the DBMS. The DBMS can therefore exploit this knowledge to optimise performance. Performance tools could be built to monitor reference trace patterns for frequently used operations. Anticipatory "pre-fetching" is then possible for objects that an application is about to reference.

In contrast, according to [Ketab90], in an RDBMS, the set of operations is limited and fixed (operations on the sets of tuples and two operations of select and project). Therefore, any operation that an application wishes to perform must be mapped onto this limited set. As applications become complex, so does this mapping, leading to large application programs. To retrieve an application specific object, several DBMS operations are required. This makes applications slow.

Evidence to back the claims in improved performance for engineering applications have been produced by the OO1 Benchmark [Catte92], and more recently by the OO7 Benchmark [Carey93]. The major features of OO1 are:

- Measures response-time for single-user access.
- Bill of Materials (BOM) parts and connections (one part, one relationship).
- Local and remote access.
- Three operations (look-up, traversal, insert).
- Cold and warm measures.

The OO1 Benchmark, however, does have some recognised shortcomings [Butte92]:

1. Typically, databases are much larger than those used in the benchmark being quoted by the ODBMS vendors.

2. The queries are fairly simple and are randomly distributed. Few applications actually have random distribution.

3. The benchmark is a single-user benchmark. This not only leaves the concurrency control systems untested, but allows the vendors to exploit the benchmark by disabling concurrency and recovery features essential to a commercial application.

Better metrics are needed not only for engineering applications, but MIS and business applications.

The remainder of this paper will examine a number of issues of importance to ODBMSs - terminology, classification, standards, strengths and weaknesses. The paper is organised as follows. Section 2 will examine some of the key object concepts required for ODBMSs. Section 3 will examine some of the key data management requirements for ODBMSs. Section 4 will examine a number of definitions that have been suggested for ODBMSs, and will also discuss various classification models that have been proposed. Section 5 will discuss the progress in the area of standards, and whether a common object data model seems to be emerging. Section 6 will conclude with a summary of the main strengths and weaknesses of ODBMSs.

2. Object Requirements for ODBMSs

The discussion is not exhaustive, and the terminology here should be compared with [OODBT91] which provides general characteristics of object models. An additional useful source is [Stefi86], which provides a comprehensive discussion of object concepts from a programming language perspective.

2.1 Object

An object is a software representation of a real-world object. Objects can be physical items (e.g. computer, disk drive, etc.) or abstract concepts (e.g. University Board of Studies meeting).

Software objects are self-contained modules that include data (instance variables) and code that acts on that data (methods). Instance variables are *"everything an object knows"*, and methods are *"everything an object can do"* [Taylo91].

An object is an instance of a class, where *"classes are descriptions of things, objects are the things themselves"* [Winbl90].

The state of an object can be defined as the values of an object's instance variables at any point in time. Alternatively, it may be defined as the result(s) returned after an object has performed certain operations. An object's state can be modified by sending it messages to invoke one of it methods.

An object's instance variables can be simple base types (e.g. integer, char), more complex types (e.g. arrays, sets, other aggregates) or complex objects themselves (e.g. computer aided design drawing).

An object can participate in a number of different (orthogonal) relationships. Some examples [Kim90a] include class and object (instance-of), generalisation (is-a), aggregation (class-composition), composition (part-of), and versioning (version-of). Each object, therefore, can have a number of different "roles" (depending on which relationship we view it from).

[Taylo91] has noted that an ODBMS can support any number of alternative structures for the same set of data, and that these structures are not simply "views" of the data superimposed on a single underlying model (viz. relational), but are all equally valid and exist independently of each other. [Prabh92] has also commented that existing data models provide poor relativism (alternative ways of looking at the same thing). With ODBMSs, better support for multiple ways of viewing the same information have become possible.

2.2 Object Identity

A unique identifier that is associated with every object, and distinguishes one object from another. This logical identifier is invariant across all possible modifications of an object's state [Dawso89], and additionally address, structure or name.

Given the uniqueness of an object identifier (OID), it is then possible to compare objects for several forms of equality.

The OIDs, instance variables, methods or even object hierarchies of two objects can be compared to determine if they are the same. If the OIDs of two objects are the same - it is the same object.

OIDs permit the referential sharing of objects. In addition, because a relationship between objects is explicitly defined, the existence of the relationship is ontologically dependent on the prior existence of the objects that participate in the relationship [Atwoo90]. In contrast, the relational model supports implicit relationships, whose semantics need to be re-created at run-time. SQL also incorporates additional capabilities, such as foreign key constraints to recapture lost semantics [Khosh92].

The OID is system generated, in contrast to the relational model where explicit identifiers need to be created by users. However, a compromise has been suggested [Commi90], where identifiers are system generated only when meaningful user generated values (e.g. social security number, student number, employee number) are not available.

Some ODBMSs do not use OIDs. ObjectStore [Lamb91], for example, does not directly manipulate objects, but manipulates virtual memory pages on which objects reside.

[Ullma88] has argued that pre-relational database languages can be classified as object-oriented, since they support the notion of object identity. Although we can view hierarchical, network and relational models as special cases of the object model, object identity is a semantic concept, whereas virtual addresses or pointers (as used in hierarchical and network systems) represent memory locations on an underlying von Neumann machine [Khosh92]. Additionally, pre-relational languages do no support concepts such as methods, classes or inheritance. A more detailed discussion of the similarities and differences between object-oriented and earlier generations of database systems can be found in [Kim90a; Kim90b].

2.3 Method

A method is equivalent to a procedure, function or subroutine in traditional programming environments. An object's methods define its interface (or protocol).

2.4 Message

A message is a request to a specific object to invoke one of its methods to perform some operation. It is equivalent to a procedure, function or subroutine call in traditional programming. As noted by [Atkin89], the syntax to send a message (invoke a method) may vary between programming paradigms. For example, the following would be equivalent - "john hire", "john.hire", "hire john", "hire(john)."

2.5 Type and Class

These are often used interchangeably. However, one of the few ODBMSs that differentiates between a type and a class is O_2 [Deux90; Deux91]. In O_2, a class encapsulates attributes (instance variables) and behaviour (methods). Objects are instances of a class. Types are components of a class and describe the structure of its instances. Values are instances of a type. Values are not encapsulated.

The class construct can be either intentional (object factory) or extensional (object warehouse). ODBMSs that attempt to provide the seamless integration of a programming language and database system by extending an object-oriented programming language with DBMS capabilities fall into the former category. This is because languages such as C++ and Smalltalk do not specify extensional capabilities with the class construct. Examples include GemStone [Bretl89; Maier90; Butte91], which extends Smalltalk.

2.6 Inheritance

Classes can inherit the attributes and behaviour of other classes. They are then organised into an "inheritance hierarchy", where classes further down the hierarchy (subclasses) are specialisations of those above them, and classes higher in the hierarchy (superclasses) are generalisations of those below them.

New subclasses can be created by programming only the differences from their superclass(es). Also, subclasses can override attributes and methods inherited from superclasses.

A subclass may have only one superclass (single inheritance) or two or more superclasses (multiple inheritance).

Single inheritance is similar to the concept of parent/child records in hierarchical database systems. Multiple inheritance can similarly be compared to owner/member records and sets in network databases.

Multiple inheritance provides the capability to model greater complexity, but at the expense of more complex software. Conflict resolution, error logging or some other strategy will be necessary, if the superclasses of a subclass contain attributes and/or methods with the same names.

2.7 Encapsulation

The manipulation of an object is only possible through its defined external interface. The implementation of the instance variables and methods is hidden. As a result, the implementation can be changed without affecting existing program code that uses the object’s interface. This provides "logical data independence" [Atkin89]. Encapsulation, to paraphrase [Winbl90], makes boundaries among objects clear, communication among objects explicit, and hides implementation details.

It has been suggested by [Atkin89], that strict encapsulation may not be desirable for ODBMSs, and that access to implementation details may be required (e.g. by the query optimiser, which is a "trusted" component of the DBMS). Also, structural access to instance variables may be needed for performance reasons when using an ad-hoc query language.

2.8 Abstract Data Type (ADT)

The ability to distinguish between an object's interface and its implementation results in an ADT. The class construct specifies an ADT.

3. Data Management Requirements for ODBMSs

This section examines how the data management paradigm has been influenced by the object paradigm, and some of the problems and challenges that have appeared.

3.1 Persistence

Within object-oriented programming languages (and languages in general), most data are of a transient nature and cease to exist when the process that created them terminates (unless explicit commands are issued to make the data persistent). An ODBMS, however, would enable data to be saved implicitly.

[Khosh90a] have discussed a number of alternative strategies for persistence - persistence extensions and persistence through reachability.

ODBMSs that define persistence extensions use the class construct to specify structure, extension and persistence (similar to a relational table).

ODBMSs using persistence through reachability specify an object space with a persistent root. Objects that can be reached from this root are also persistent.

3.2 Transactions

The properties of a transaction are [Loomi90]:

1. It is application-defined. It obeys the applications rules of consistency.

2. It is all-or-nothing. All parts of a transaction complete and any updates are committed to the database, otherwise the entire transaction aborts.

3. Once a transaction commits, changes to the database cannot be undone, except by running another transaction.

The second property has certainly had to be re-examined for ODBMSs, since in many application areas (e.g. CAD), transactions could be very lengthy. For example, data could be "checked out" of a central database into a local workstation, worked on for hours or even days, and then "checked in" again. To lock data for such lengthy periods of time would result in poor levels of performance if concurrent access to the data was required by many users.

A new model to support these "long transactions" for ODBMSs has been suggested by [Rotze90].

[Brown91] has also recognised that new models are required to support conversational, long duration, complex and non-atomic transactions.

3.3 Concurrency

One of the major benefits of a DBMS is that data can be shared by multiple users and applications. However, with multiple transactions attempting to access the same data at the same time, some form of control is required to ensure that the database is always in a consistent state. To support these two opposite demands upon the database, a serialisable order of execution for transactions is usually imposed.

[Khosh90a] discuss three possible strategies (time-stamp ordering, optimistic algorithms, pessimistic algorithms) to ensure the serialisability of transactions. [OODBT91] note some additional important characteristics of transactions.

[SH90] note that ODBMSs offer *"long transactions, gaining optimistic concurrency"*, although some commercial ODBMS systems, for example GemStone, offer hybrid schemes that combine several of the above mentioned strategies.

3.4 Recovery

To return the database to a consistent state, after a failure (e.g. transaction, system, media).

This is another area that has required re-examination, since many of the current commercial ODBMS systems use a client/server architecture. According to Michael Stonebraker (cited in [Hazza90]) most ODBMSs run their data manager in the same protection environment as the user program (usually the client workspace). Consequently, a protection boundary does not have to be crossed for data lookup. Although this improves performance, it results in the loss of a protected database. This could cause potential problems for recovery if there is a failure.

3.5 Querying

A criticism of data manipulation in object databases is that record-at-a-time navigational access is required [Date90]. However, many ODBMSs provide declarative SQL-like DDL/DMLs, e.g. OPAL (GemStone), OSQL (IRIS), OQL[C++] (Zeitgeist). These languages vary in their support for data abstraction, ranging from the direct manipulation of attributes to access via an objects' public interface. Some of these languages are also computationally complete (e.g. OPAL). Additionally, commercial ODBMS vendors provide rich graphical tools for database design, administration, examination, and the development of user applications, thus excluding the need to directly manipulate the underlying structures.

3.6 Security and Authorisation

[Barry91] has stated that some ODBMSs provide very extensive support for security, whilst others do not even provide a log-on password. This has been confirmed by a review of some of the commercial products in [OOS92].

Security and authorisation have not been adequately addressed by object database vendors, since many commercial products were developed for group-work applications, where such issues were perhaps not considered to be of prime importance. However, such issues need to be addressed for other application domains, where access to sensitive data must be restricted. In contrast, many relational systems offer certifiable security levels [Loomi92].

Further discussion of these issues can be found in [Kim90b].

4. Taxonomy

4.1 Definitions of ODBMSs

ODBMSs can be considered as the convergence of a number of technologies, most notably, Object-Oriented Programming Languages, Semantic Models, and Complex Object Models [Khosh90a]. Additional influences have come from Artificial Intelligence [Jeffc89; Paton91].

Despite this convergence, there is currently no single object data model for ODBMSs (in contrast to the relational model for RDBMSs). The reason for this has been noted by [Catte91a]:

> *"There is no single object-oriented paradigm, and therefore there are a variety of object-oriented data models."*

Most research and commercial efforts have, therefore, resulted in various interpretations and implementations of the paradigm. As a result of this, there is considerable confusion as to exactly what constitutes an ODBMS (e.g. [Lagun89]), although [Kim90b] has noted that many ODBMSs share a set of core concepts.

In recent years, some efforts have been undertaken to define more precisely the requirements for ODBMSs and next-generation database systems. Some "good" definitions/tenets that have been suggested include:

- Atkinson et al. [Atkin89]
- Cattell [Catte91a]
- Committee [Commi90]
- Dittrich [Dittr86]
- Joseph et al. [Josep91]
- Khoshafian & Abnous [Khosh90a]
- Kim [Kim90a]
- Lockemann [Locke92]
- OODBTG [OODBT91]
- Ullman [Ullma87]
- Unland & Schlageter [Unlan90]
- Zdonik & Maier [Zdoni90]

Strictly speaking, the Object Data Management (ODM) Reference Model [OODBT91] does not define an ODBMS, but proposes a design space for a family of models. In effect this provides an umbrella over all the current definitions, since most of the concepts suggested by the leading researchers and academics have been included in the proposed reference model. Therefore, the relative merits or drawbacks of each definition will not be discussed, and various classification models will be reviewed instead.

4.2 Classification Models of ODBMSs

A number of ODBMS classification models have been suggested in the literature. Although there is considerable overlap between the models, a summary of the major approaches is now presented.

4.2.1 Data Models

This approach has been discussed by a number of authors, e.g. [Khosh90a; Catte91a].

• *Non-First Normal Form (NF^2) Models*
Extend the relational model, whilst trying to maintain a strong mathematical foundation. Such models relax the First Normal Form constraint of the relational model and allow repeating groups. They also try to provide better support for complex objects. POSTGRES is perhaps the closest example to this approach. Other examples include DASDBS [Schek90] where a DBMS kernel is augmented with application-specific front-ends.

• *Object-Oriented Languages*
There are more than 80 object-oriented languages in the world [McClu92]. These differ significantly in many respects (for example, compare C++ with Smalltalk). ODBMSs that are based on object-oriented language models include O_2, ORION [Kim90b; Kim90c], ObjectStore, ONTOS, and GemStone.

• *Functional Models*
A number of ODBMSs have been built, based on the DAPLEX functional model [Shipm81]. Attributes, methods, and relationships are all represented by functions in these models. Subtypes and referential integrity are also supported. Examples of ODBMSs include OpenODB [Ahad92] based upon the IRIS research prototype [Fishm89; Wilki90], PROBE [Manol86], and VISION.

Functional and semantic models have been distinguished using the definition of object databases suggested by [Dittr86]. Semantic models are here considered as *structurally* object-oriented and functional models as *operationally* object-oriented.

• *Semantic Models*
There are many semantic models, but the Semantic Data Model (SDM), developed by [Hamme81] was one of the earliest attempts to add more semantics (e.g. objects, type hierarchy, etc.) to the relational model. SDM models structural abstractions (like frames in AI systems), and does not support behavioural abstractions (i.e. ADTs). SDM uses a diagrammatic representation similar to semantic networks, with nodes and links, to depict entity types and relationships. SIM [Fritc90] is perhaps the best example of a database system based on a semantic model (SDM).

Further discussion of semantic models and database implementations can be found in [Prabh92].

4.2.2 Architectures

Classification by data models is a useful way to distinguish the genealogy of a system, i.e. which previous systems and philosophies have contributed, but that a better method is by architecture [Catte91a].

[Catte91a; Catte91b] has suggested the following architectural classification.

• *Object Managers*
These systems are extensions of existing file systems or virtual memory, have a limited data model, no query language, and provide the most basic functionality (storage of persistent objects). Examples include POMS, Mneme and ObServer.

Persistent-Data Servers [Simme92] are a recent development akin to Object Managers. They attempt to provide an alternative persistence mechanism to ODBMSs and file systems. They offer some DBMS features and perform like file systems, whilst maintaining the structure of the data on the disk, independent of the application that created them.

• *Extended Database Systems*
These systems attempt to provide multi-programming language access by supporting a database language neutral model (sacrificing performance for data independence). New or extended database query languages that provide richer modelling concepts such as classes, inheritance, types, and functions are provided by these systems, e.g. POSTQUEL, OSQL (IRIS). In addition, there is full support for DBMS features such as concurrency control, transaction management, etc. Examples include POSTGRES, Starburst [Haas90; Lohma91], PROBE, VISION, IRIS, and SIM.

• *Database Programming Languages*
Extend existing programming languages (e.g. C++, Smalltalk, Lisp) to provide DBMS features such as persistence, concurrency, etc. The database query language and application programming language execute in the same workspace and share the same type system. Examples include O_2, ORION, ObjectStore, ONTOS, and GemStone.

• *Database System Generators*
Since no DBMS can provide all the specialised functions and operations needed by a specific application area (e.g. text processing, GIS), these "toolkits" enable customisable DBMSs to be built. Examples include GENESIS [Bator86] and EXODUS [Carey86; Carey89].

Further discussion of the data management requirements for environments such as project management, office documents, geographical and spatial information can be found in [Oxbor91].

• *Relational Object Shells*
Backward compatibility with RDBMSs is the main motivation for the development of such systems [Catte91b]. [Rasmu92] has also noted that *"to prove useful, object-oriented databases must be able to be integrated into a relational world."* This could be achieved by interfacing or encapsulation. The latter approach is more elegant and requires a "wrapper" to be built around a relational database, thus providing object-level interfaces and hiding the syntax and semantics of the underlying database. In this scenario, the relational database simply becomes an abstract data definition [Rasmu92].

A shell would store objects as "virtual" tables [Winbl90]. There is, however, a performance trade-off, since retrieving complex structures (using search-and-match) can be time-consuming in RDBMSs. Storing the same structures in a full ODBMS would enable them to be retrieved in a single query.

Other approaches to integrating existing databases with object systems have been proposed by [Ahad88; Preme90; Nelso90].

4.2.3 Storage Server Models

This approach has been proposed by [Josep91].

• *Typeless Page Servers*
These systems do not directly manipulate objects, but manipulate virtual memory pages on which objects reside. A user application and the server share (transient and persistent) virtual memory. This approach is, therefore, architecture specific, since page formats differ between hardware platforms. Examples include EXODUS and ObjectStore.

• *Typeless Object Servers*
Object servers control access to objects or groups of objects. They have very limited knowledge about the objects themselves (e.g. OID, objects have a type, objects may be related). They cannot execute methods or access the states of objects. Examples include Mneme, ObServer, and Zeitgeist.

• *Class-Based Object Servers*
Such systems manipulate objects or groups of objects, and can interpret an object's state to provide additional services (e.g. queries). These systems are (typically) built on relational storage managers, mapping objects onto tables. Two alternatives to representing inheritance are either through horizontal or vertical partitioning. In horizontal partitioning, the inheritance graph is flattened, leading to class evolution problems. Using vertical partitioning, each inherited definition is represented by one table, requiring the use of joins.

The storage manager provides support for full DBMS features, such as backup, recovery, concurrency, etc. Examples include POSTGRES and IRIS.

• *Type-Based Object Servers*
These object servers can execute methods, and enable computations to be moved from client to server. Any of the other three server types discussed can be enhanced with extra layers of software to become type-based object servers. Examples include ORION and O_2.

4.2.4 Alternative Object Database Strategies

[Khosh90a] have proposed six different approaches to ODBMSs.

• *Novel Database Data Model/Data Language Approach*
[Khosh90a] have suggested that many research projects have pursued this approach. Of the commercial systems available, SIM has been cited because of its novel DDL/DML.

• *Extend an Existing Database Language with O-O Capabilities*
With the ANSI X3H2 committee currently working towards the definition of SQL3 (SQL with object extensions), this is the path that is most likely to be pursued by existing relational database vendors that wish to provide greater support for object concepts.

• *Extend an O-O Programming Language with Database Capabilities*
An object-oriented programming language already supports object concepts, but lacks DBMS facilities (e.g. querying, transactions, persistence, etc.). Using this approach, the language is extended with these capabilities. The best example is GemStone.

• *Extendible ODBMS Client Libraries*
An alternative to the previous approach is to provide libraries that can extend the capabilities of a language to provide classes of aggregates (sets, lists, arrays), types, and methods for transaction handling, etc. ONTOS, ObjectStore, and VERSANT are good examples of this approach.

• *Embed Object Database Language Constructs in a Host Language*
This approach is similar to using Embedded SQL (ESQL) in a host language (e.g. C, Pascal, etc.). O_2 uses this approach to provide database access from BASIC ($BASICO_2$) and C (CO_2).

• *Application-Specific with an Underlying ODBMS*
The example cited by [Khosh90a] for this approach is that of TeamOne, which is a configuration management system for engineering applications. This system supports an object repository for project design files, with access and modification being achieved by manipulating encapsulated objects.

4.2.5 Language Data Models

[Wells92] refer to three approaches to ODBMSs, based on language data models.

• *Programming Language Neutral*
The data model in this approach bears no direct relationship with the programming languages that manipulate data within programs. As mentioned earlier, such an approach aims to provide maximum data independence, though (inevitably) leads to the "impedance mismatch" problem. Examples include PROBE, POSTGRES, GEM, IRIS, ORION, and O_2.

• *Database Programming Language*
This approach provides the highest level of transparency of database access from a programming language, since the language has been specifically designed with database capabilities in mind. Examples are Galileo [Alban86] and Taxis.

• *Programming Language Specific*
The data model in this approach is an extension of the type system of an existing programming language. Although the main objective is to ameliorate the "impedance mismatch" problem, difficulties can arise when users wish to use another language for their application programs - most ODBMSs do not seem to provide language interfaces for conventional languages, such as COBOL, FORTRAN, etc.

4.2.6 Evolution vs. Revolution (Retrofit vs. Replacement or Hybrid vs. Pure)

Ultimately, perhaps all the classification models discussed so far can be generalised into either one of two approaches, namely evolutionary - extending existing (relational) databases with support for object concepts, or revolutionary - abandoning existing database technology in favour of a "fresh" start. These two approaches have received some attention in the literature, e.g. [Brodi89; Spier91].

A number of market research reports (e.g. [Jeffc91]) indicate that there is room for both types of database systems, although ultimately it may not matter, since the two database technologies appear to be converging. For example, POSTGRES uses a set-oriented query

language (POSTQUEL), but navigational access is also possible, since each record has an OID [Stone91]. With this in mind, an important point noted by [Taylo92] is that as soon as relational vendors begin adding pointer navigation to their products, it brings into question the entire mathematical foundation of relational technology.

5. ODBMS Standards

It has been reported by [OODBT91], that there are strong arguments in favour of a standard object data model. A reference model has also been defined. The purpose of the reference model is:

1. To provide a common language of ODBMS definitions.

2. To provide a means to differentiate ODBMS systems from other DBMS systems.

3. To provide a means to differentiate ODBMS systems from one another.

4. To enable future standards work in the related programming and data management areas to have a basis to work from.

A standard data model would then enable more effort to be devoted to other issues of importance, leading to the more widespread use and acceptance of ODBMS technology. Some of the areas that need to be addressed have already been identified, e.g. [Banci90; Kim90a].

5.1 Areas for Standardisation

[Catte91a] has suggested standardisation efforts in the following areas.

5.1.1 Object Data Model

It is difficult to say how many ODBMSs there are in the world today. [Evere92], for example, have cited more than 80 organisations world-wide (with perhaps as many data models) that are *"believed to have some form of an Object-Oriented Database Management system (ODM) implemented or in development."* Their report also highlights major differences in terminology. For example, what is an object?

A common industry-wide definition for what exactly constitutes an ODBMS needs to emerge. This may be almost impossible to achieve, given the many interpretations and implementations of the object paradigm (as discussed earlier). Perhaps the best (and only) way forward is through the efforts of standards bodies.

5.1.2 Object Query Language

Currently one ODBMS vendor’s query language cannot be used on another vendor’s system. A common standard (like SQL) is required. There is some debate as to what form this language should take [Works91], and whether SQL with object extensions is suitable [Beech90] or not [Orens90].

Some of the arguments in favour of using SQL as the basis of an object query language are:

- It already has an ANSI defined standard.
- SQL with object extensions would allow backward compatibility with existing applications that used standard SQL.
- It would provide a language neutral model, rather than tying down the query language to any particular programming language (e.g. C++).

Additionally:

- It is *"intergalactic dataspeak."*
  [Commi90]

- *"Our perspective is that SQL is the standard language for database access. Like democracy, it may not be perfect, but it's better than anything else that's around."*
  Ken Jacobs, cited in [Hazza90]

- *"It is socially irresponsible to invent new languages if an existing language is a good approximation to what is required."*
  [Beech88]

Some of the drawbacks of using an Object SQL (based upon DAPLEX) have been highlighted by [Gray92]:

- Syntax of OSQL may be similar to SQL, but the semantics may be quite different (e.g. implicit joins, unexpected behaviour of familiar constructs).
- Limited computational power.
- Restrictive structure.
- Awkward syntax.

In terms of query formulation, [Kim90b] notes that the structure of an object-oriented query is basically that of a relational query. There are, however, significant differences between ODBMS query languages. For example, most languages support path expressions in query formulation, but vary in the degree of encapsulation. In ORION, attributes can be directly referenced, whereas in IRIS, functions (methods) are used. Access through a behavioural interface provides better data abstraction than access through state. [Blake91] discusses other significant differences.

5.1.3 Programming Language

Portability of application code between ODBMSs is also desirable. It appears that currently C++ is becoming the *de facto* programming language standard for most commercial ODBMSs. Lack of portability of application code is something that relational systems have suffered from, since most use proprietary 4GLs.

5.1.4 SQL

With the advent of heterogeneous distributed databases, a common form of communication between various types of DBMSs is required. SQL may be the best choice, since it has already become a standard communications protocol for many client/server applications. Also, so-called "legacy" systems cannot be ignored as significant quantities of data are

already held in tables, accessed by SQL. In fact, many ODBMSs (e.g. GemStone) are already offering gateways that allow SQL access to the popular relational systems, and permit SQL queries to be viewed as objects.

Differences in SQL implementations, however, will cause problems with interoperability. For example, [Edels91] has noted that RDBMS vendor implementations of SQL are known to suffer from the following major differences:

- Syntactical differences.
- Semantic differences.
- Dictionary tables.
- Return codes.
- Host language interface.

Presumably these are in addition to other "minor problems" [Commi90].

5.2 Achieving Standards

There are a number of possibilities as to how these standards may be accomplished [Kim91].

5.2.1 Industry-Wide Efforts

• *ANSI/X3/SPARC/DBSSG/OODBTG*
An Object Data Management (ODM) Reference Model has been produced [OODBT91]. This defines an Object Data Management (ODM) design space as:

General Characteristics of Object Models
\+ Data Management Characteristics
\+ System Characteristics.

The reference model is a superset of many of the definitions mentioned earlier, and can perhaps serve as a better set of core concepts, building upon those proposed by [Kim90a].

• *Object Management Group (OMG)*

> *"OMG ... the mother of all consortia"*
> Christopher Stone, reported in [Compu91a]

This industry consortium of over 300 members has been attempting to forge an industry-wide reference model for ODBMSs. There are good reasons to be optimistic, since the OMG is not one of several rival standards bodies, and its members include all the major players in the industry [Taylo92].

• *Others*
A number of other (more specialised) efforts are also being undertaken, e.g. CAD Framework Initiative (CFI), Portable Common Tools Environment (PCTE), and X3H2 (SQL3), to name a few.

5.2.2 De Facto Standard

Of the current commercial products, Object Design Inc. claim a 35% market share (of world-wide ODBMS sales) for ObjectStore [Marsh93]. Whether this can be sustained in the long term remains to be seen, particularly with major RDBMS vendors poised to offer significant object extensions to their products within the next few years.

5.2.3 Major Companies

IBM currently has business partnership agreements with several of the commercial ODBMS vendors. It also has its own research efforts such as Starburst, and Cloris [Evere92].

[Engli92] has mentioned some other products from major systems vendors including Object/DB (DEC), OpenODB (Hewlett-Packard), OSMOS/SIM (UNISYS), and Zeitgeist (Texas Instruments).

ORACLE V8 will, according to the popular computer press, be a "full-blown" object database [Lauch92]. Exactly how object-oriented it will be remains to be seen. Most likely, the underlying storage mechanism will still be a traditional database manager, as few users wish to move to a new database management strategy [Hodge89].

6. Strengths and Weaknesses

The following is not an exhaustive list. Additional discussion can be found in [Winbl90; Brown91; Kim91].

6.1 Strengths of ODBMSs

6.1.1 Better Support for Complex Applications

ODBMSs have provided better support for certain applications requiring high performance and modelling of complex relationships and heavily inter-related data that have traditionally not been well served by other DBMSs.

6.1.2 Enhance Programmability

[Atwoo90] has stated that *"Using a highly integrated, object-oriented database management system (OODBMS) to build applications and store reusable code can save 20% to 30% of development cost beyond that achieved solely by object-oriented programming."*

The seamless integration of an application programming language and database DDL/DML enables further savings in code.

6.1.3 Improve Performance

An ODBMS has more "knowledge" of the operations being performed, and can therefore provide better control of concurrently executing transactions. This leads to better performance.

6.1.4 Improve Navigational Access

Using declarative query languages has proved to be very useful in relational systems. However, users should not be constrained in the way they interact with a database. It should be possible to support a variety of interaction styles (e.g. natural language, menus, prompts, graphical browsers, etc.) that can all be used to augment the formal command language of the database. In any case, there are many structures for which a navigational mode of access is more natural and intuitive than using a declarative query language.

6.1.5 Simplify Concurrency Control

Detailed locking strategies for some ODBMSs provide highly sophisticated control of objects (e.g. ORION). At a coarse level of granularity, it is possible to place a single lock on an entire object hierarchy. In contrast, an RDBMS would require multiple locks to get information from related but scattered data in the database. At a fine level of granularity, individual objects or attributes of objects could be locked (though the latter would violate encapsulation).

Some ODBMSs provide optimistic concurrency mechanisms, since these may be more applicable in design and engineering environments, where users may work on a part of a design, and there is less likelihood of conflict with other users.

6.1.6 Reduce Problems of Referential Integrity

One of the major problems with relational databases has been that of dangling references. This problem has been solved in object databases by giving the responsibility of pointer maintenance to the database itself, rather than to each application.

[Maier90] note that referential integrity comes "for free" in GemStone, and that *"One object refers directly to another object, not to a name for that object. The reference cannot be created if the other object does not exist. Hence, there are no dangling references."*

6.1.7 Abstraction Gain

To paraphrase [Hazza90], the relational model "hammers" the world flat. However, most of the world is not flat. The object paradigm provides a better modelling framework that captures not only data and behaviour, but raises the level of abstraction (an object is more than the sum of its attributes and methods). This was briefly noted by [Atwoo90], when he stated that *"object-oriented systems have a conceptual rather than a data model."* [Meers91] has similarly noted that it is the conceptual side or "abstraction gain" that has primarily attracted users to object databases.

6.2 Weaknesses of ODBMSs

6.2.1 Lack of Standards

[OODBT91] and the OMG have identified areas for standards efforts. Various ANSI committees are also engaged in standards work (e.g. SQL3, C++). However, there is currently no agreed standard for ODBMSs.

6.2.2 No Formal Semantics

The relational model is based on set theory and relational calculus, resulting in a "clean", mathematically proven model. As yet, no equivalent complete object theory or object calculus exists. However, [Rine92] have stated that finite state automata and formal logic can be used to represent objects mathematically. In addition, [Freyt88] have suggested that object-oriented concepts can be represented by relational concepts, and operations on objects can be mapped into operations on relations.

6.2.3 Loss of Protected Database

The loss of a protected database (mentioned earlier) can be seen as a significant weakness. In addition, some ODBMS products provide direct pointer mechanisms for accessing persistent objects, which according to [Taylo92] would increase the vulnerability of the database to wild pointers in application programs, and would *"strike terror in the hearts of most traditional database administrators."*

6.2.4 Few Successful Applications Beyond CAD/CAM/CASE, etc.

Most of the ODBMS applications cited in the literature centre on CAx. This is not really surprising, since it is precisely these areas that have led to the growth of ODBMSs. Although [Rasmu92] has noted that object databases can represent both traditional and more complex data, few ODBMSs seem to be aimed directly at MIS and business applications. [Butte91] have stated, however, that GemStone has been developed to accommodate MIS and engineering applications.

6.2.5 Object Technology Difficult to Learn

[Hazza90] states that even for experienced C programmers, C++ takes from three to six months to learn. Furthermore, the *"need to train MIS personnel in object-oriented techniques will be a hurdle for a long time to come."* Evidence to support such statements has come from a number of case studies, e.g. [Goldb92; Taylo92].

6.2.6 Loss of Relational Simplicity

One of the great strengths of the relational model is its simplicity. All data are viewed in the form of relations (tables). In contrast, object models give the opportunity of capturing more semantics by providing richer modelling constructs, but at the expense of greater complexity.

The move from computer-oriented objects to objects that attempt to mirror reality is not without its costs [Relea89] - there's no such thing as a "free lunch!" Additionally, trying to capture too much of the "real-world" may cause performance problems, similar to performance problems caused by over-normalisation in relational systems.

6.2.7 Limited Endorsement by Major Companies

Most of the commercial ODBMS products are from start-up companies. This is in many ways reminiscent of the days when network and hierarchical vendors dominated, and relational start-up companies were just emerging.

The announcement that ORACLE is committed to an object database will act as catalyst for other vendors to follow suit. However, advertising claims need to be treated with some scepticism [Codd92].

7. Summary

This paper has provided an overview of Object Database Management Systems (ODBMSs). Some of the shortcomings of existing database technology to meet the new and emerging application domains were discussed. A brief discussion of object and data management concepts was then presented. A number of definitions for ODBMSs were cited, but greater emphasis was stressed on various classification models. It was noted that there were overlaps between the classifications, but each model contributed something, which other models lacked. The efforts towards standards were briefly reviewed. It seems that there is unlikely to be one overall standard definition for ODBMSs, since there is significant disagreement between vendors and developers on even basic concepts. Finally, some of the main strengths and weaknesses were discussed. At the present time, the weaknesses seem to outweigh the strengths. However, this situation will change with time, as ODBMS technology is still evolving and developing.

8. References

[Ahad88] Ahad, R. The object shell: an extensible system to define an object-oriented view of an existing database. *Proceedings of the Second International Workshop on Object-Oriented Database Systems*. Lecture Notes in Computer Science, 334, pp. 174-192, Springer-Verlag, Berlin, 1988.

[Ahad92] Ahad, R. & Dedo, D. OpenODB from Hewlett-Packard: a commercial object-oriented database management system. *Journal of Object-Oriented Programming*. 4 (9):31-35, 1992.

[Alban86] Albano, A., et al. A strongly typed, interactive object-oriented database programming language. *Proceedings of the International Workshop on Object-Oriented Database Systems*, Asilomar, California, 1986, pp. 94-103.

[Atkin83] Atkinson, M., et al. An approach to persistent programming. *Computer Journal*. 26 (4):360-365, 1983.

[Atkin89] Atkinson, M., et al. The object-oriented database system manifesto. *Proceedings of the First International Conference on Deductive and Object-Oriented Databases*, Kyoto, Japan, 1989, pp. 223-240.

[Atwoo90] Atwood, T. Applying the object paradigm to databases. *Computer Language*. 7 (9):36-42, 1990.

[Atwoo92] Atwood, T. An introduction to ODBMSs. *JOOP Focus on ODBMS*, SIGS Publications Inc., New York, 1992.

[Banci90] Bancilhon, F. & Kim, W. Object-oriented database systems: in transition. *SIGMOD RECORD*. 19 (4):49-53, 1990.

[Barry91] Barry, D.K. Perspectives on changes for ODBMSs. *Journal of Object-Oriented Programming*. 4 (4):19-20, 1991.

[Bator86] Batory, D.S. GENESIS: a project to develop an extensible database management system. *Proceedings of the International Workshop on Object-Oriented Database Systems*, Asilomar, California, 1986, pp. 207-208.

[Beech88] Beech, D. A foundation for evolution from relational to object databases. *Advances in Database Technology - EDBT '88*. Lecture Notes in Computer Science, 303, pp. 251-270, Springer-Verlag, Berlin, 1988.

[Beech90] Beech, D. & Ozbutun, C. Object databases as generalizations of relational databases. *Proceedings of the Object-Oriented Database Task Group Workshop*, Ottawa, Canada, 1990, pp. 119-135.

[Blake91] Blakeley, J. A. ZQL[C++]: extending a persistent C++ language with a query capability. Computer Research Laboratories, Computer Science Laboratory Technical Report ITB-91-10-01. Texas Instruments Inc., 1991.

[Bretl89] Bretl, R., et al. The GemStone data management system. *Object-Oriented Concepts, Databases, and Applications*, edited by W. Kim and F.H. Lochovsky, Addison-Wesley, Reading, MA, 1989.

[Brodi89] Brodie, M.L., et al. Next generation database management systems technology. *Proceedings of the First International Conference on Deductive and Object-Oriented Databases*, Kyoto, Japan, 1989, pp. 335-346.

[Brown91] Brown, A.W. Object-oriented databases: their applications to software engineering, McGraw-Hill, New York, 1991.

[Butte91] Butterworth, P., Otis, A. & Stein, J. The GemStone object database management system. *Communications of the ACM*. 34 (10):64-77, 1991.

[Butte92] Butterworth, P. ODBMSs as database managers. *JOOP Focus on ODBMS*, SIGS Publications Inc., New York, 1992.

[Carey86] Carey, M.J., et al. The architecture of the EXODUS extensible DBMS. *Proceedings of the International Workshop on Object-Oriented Database Systems*, Asilomar, California, 1986, pp. 52-65.

[Carey89] Carey, M.J., et al. Storage management for objects in EXODUS. *Object-Oriented Concepts, Databases, and Applications*, edited by W. Kim and F.H. Lochovsky, Addison-Wesley, Reading, MA, 1989.

[Carey93] Carey, M.J., DeWitt, D.J. & Naughton, J.F. The OO7 Benchmark. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Washington DC, 1993, pp. 12-21.

[Catte91a] Cattell, R.G.G. Object data management: object-oriented and extended relational database systems, Addison-Wesley, Reading, MA, 1991.

[Catte91b] Cattell, R.G.G. What are next-generation database systems? *Communications of the ACM*. 34 (10):31-33, 1991.

[Catte92] Cattell, R.G.G. & Skeen, J. Object Operations Benchmark. *ACM Transactions on Database Systems*. 17 (1):1-31, 1992.

[Codd92] Codd, E.F. Dr. Codd on "End of Relational". *DBMS*. 5 (11):6, 1992.

[Commi90] Committee For Advanced DBMS Function. Third-generation database system manifesto. *SIGMOD RECORD*. 19 (3):31-44, 1990.

[Compu91a] Computing. OMG takes first step to an object standard. 7 November 1991, p. 11.

[Compu91b] Computing. Database giants revamp products. 14 November 1991, p. 11.

[Date90] Date, C.J. An introduction to database systems, volume 1, Addison-Wesley, Reading, MA, 1990.

[Dawso89] Dawson, J. A family of models. *BYTE*. 14 (9):277-286, 1989.

[Deux90] Deux, O., et al. The story of O_2. *IEEE Transactions on Knowledge and Data Engineering*. 2 (1):91-108, 1990.

[Deux91] Deux, O., et al. The O_2 system. *Communications of the ACM*. 34 (10):34-48, 1991.

[Dittr86] Dittrich, K.R. Object-oriented database systems: the notion and the issues. *Proceedings of the International Workshop on Object-Oriented Database Systems*, Pacific Grove, California, 1986, pp. 2-4.

[Edels91] Edelstein, H.A. Database world targets next-generation problems. *Software Magazine*. 11 (6):79-86, 1991.

[Engli92] English, L.P. Object databases at work. *DBMS*. 5 (11):44-58, 1992.

[Evere92] Everest, G.C. & Hanna, M.S. Survey of object-oriented database management systems. Technical Report, Carlson School of Management, University of Minnesota, 1992.

[Fishm89] Fishman, D.H., et al. Overview of the Iris DBMS. *Object-Oriented Concepts, Databases, and Applications*, edited by W. Kim and F.H. Lochovsky, Addison-Wesley, Reading, MA, 1989.

[Freyt88] Freytag, J.C., Manthey, R. & Wallace, M. Mapping object-oriented concepts into relational concepts by meta-compilation in a logic programming environment. *Proceedings of the Second International Workshop on Object-Oriented Database Systems*. Lecture Notes in Computer Science, 334, pp. 204-208, Springer-Verlag, Berlin, 1988.

[Fritc90] Fritchman, B.L., et al. SIM: design and implementation of a semantic database system. *Research Foundations in Object-Oriented and Semantic Database Systems*, edited by A.F. Cárdenas and D. McLeod, Prentice-Hall, Englewood Cliffs, 1990.

[Goldb92] Goldberg, A. Making a commitment to success. Keynote Speech. *Object-Expo Europe*, London, UK, 1992.

[Gray92] Gray, P.M.D., Kulkarni, K.G. & Paton, N.W. Object-oriented databases: a semantic data model approach, Prentice-Hall, New York, 1992.

[Haas90] Haas, L.M., et al. Starburst mid-flight: as the dust clears. *IEEE Transactions on Knowledge and Data Engineering*. 2 (1):143-160, 1990.

[Hamme81] Hammer, M. & McLeod, D. Database description with SDM: a semantic data model. *ACM Transactions on Database Systems*. 6 (3):351-386, 1981.

[Hazza90] Hazzah, A. Objects are taking shape in flat relational world. *Software Magazine*. 10 (7):32-42, 1990.

[Hodge89] Hodges, P. A relational successor? *Datamation*. 35 (21):47-50, 1989.

[Hughe91] Hughes, J.G. Object-oriented databases, Prentice-Hall, New York, 1991.

[Jeffc89] Jeffcoate, J., Hales, K. & Downes, V. Object-oriented systems: the commercial benefits, Ovum Ltd., London, 1989.

[Jeffc91] Jeffcoate, J. & Guilfoyle, C. Databases for objects: the market opportunity, Ovum Ltd., London, 1991.

[Josep91] Joseph, J.V., et al. Object-oriented databases: design and implementation. *Proceedings of the IEEE*. 79 (1):42-64, 1991.

[Ketab90] Ketabchi, M.A., et al. Comparative analysis of RDBMS and OODBMS: a case study. *Proceedings of COMPCON IEEE Computer Society International Conference*, San Francisco, California, 1990, pp. 528-537.

[Khosh90a] Khoshafian, S. & Abnous, R. Object orientation: concepts, languages, databases, user interfaces, John Wiley & Sons, New York, 1990.

[Khosh90b] Khoshafian, S., Blumer, R. & Abnous, R. Inheritance and generalization in Intelligent SQL. *Proceedings of the Object-Oriented Database Task Group Workshop*, Ottawa, Canada, 1990, pp. 103-118.

[Khosh92] Khoshafian, S., et al. Intelligent offices: object-oriented multi-media information management in client/server architectures, John Wiley & Sons, New York, 1992.

[Kim90a] Kim, W. Object-oriented databases: definition and research directions. *IEEE Transactions on Knowledge and Data Engineering*. 2 (3):327-341, 1990.

[Kim90b] Kim, W. Introduction to object-oriented databases, MIT Press, Cambridge, MA, 1990.

[Kim90c] Kim, W., et al. Architecture of the ORION next-generation database system. *IEEE Transactions on Knowledge and Data Engineering*. 2 (1):109-124, 1990.

[Kim91] Kim, W. Object-oriented database systems: strengths and weaknesses. *Journal of Object-Oriented Programming*. 4 (4):21-29, 1991.

[King89] King, R. My cat is object-oriented. *Object-Oriented Concepts, Databases, and Applications*, edited by W. Kim and F.H. Lochovsky, Addison-Wesley, Reading, MA, 1989.

[Lagun89] Laguna Beach Participants. Future directions in DBMS research. *SIGMOD RECORD*. 18 (1):17-26, 1989.

[Lamb91] Lamb, C., et al. The ObjectStore database system. *Communications of the ACM*. 34 (10):50-63, 1991.

[Lauch92] Lauchlan, S. Oracle goes towards objects. *Computing*. 26 March 1992, p. 1.

[Locke92] Lockemann, P.C. Object-oriented databases and deductive databases: Systems without market? Market without systems? *Proceedings of the International Conference on Database and Expert Systems Applications*, Valencia, Spain, 1992, pp. 1-7.

[Lohma91] Lohman, G.M., et al. Extensions to Starburst: objects, types, functions, and rules. *Communications of the ACM*. 34 (10):94-109, 1991.

[Loomi90] Loomis, M.E.S. Database transactions. *Journal of Object-Oriented Programming*. 3 (3):54-61, 1990.

[Loomi92] Loomis, M.E.S. Integrating objects with relational technology. *JOOP Focus on ODBMS*, SIGS Publications Inc., New York, 1992.

[Maier89] Maier, D. Making database systems fast enough for CAD applications. *Object-Oriented Concepts, Databases, and Applications*, edited by W. Kim and F.H. Lochovsky, Addison-Wesley, Reading, MA, 1989.

[Maier90] Maier, D., et al. Development of an object-oriented DBMS. *Research Foundations in Object-Oriented and Semantic Database Systems*, edited by A.F. Cárdenas and D. McLeod, Prentice-Hall, Englewood Cliffs, 1990.

[Manol86] Manola, F. & Dayal, U. PDM: an object-oriented data model. *Proceedings of the International Workshop on Object-Oriented Database Systems*, Asilomar, California, 1986, pp. 18-25.

[Marsh93] Marshall, K.E. President and CEO, Object Design (UK) Ltd., *Personal Communication*.

[McClu92] McClure, S. Object technology: a key software technology for the '90s. International Data Corporation (IDC) White Paper, 1992.

[Meers91] Meersman, R. A. Editor's foreword and introduction. *Proceedings of the IFIP TC2/WG 2.6 Working Conference on Object-Oriented Databases: Analysis, Design & Construction*, Windermere, UK, 1990, pp. v-viii.

[Nelso90] Nelson, M.L., Moshell, J.M. & Orooji, A. A relational object-oriented management system. *IEEE Ninth Annual International Phoenix Conference on Computers and Communications*, Scottsdale, Arizona, 1990, pp. 319-323.

[OODBT91] OODBTG. Object data management reference model. (ANSI/X3/SPARC/ DBSSG/OODBTG). Final Technical Report. 17 September 1991.

[OOS92] Object-Oriented Strategies. Object-oriented database management system products. February 1992, Cutter Information Corp., Arlington, MA.

[Orens90] Orenstein, J. & Bonte, E. The need for a DML: why a library interface isn't enough. *Proceedings of the Object-Oriented Database Task Group Workshop*, Ottawa, Canada, 1990, pp. 83-93.

[Oxbor91] Oxborrow, E., et al. Object-oriented data management in specialized environments. *Information and Software Technology*. 33 (1):22-30, 1991.

[Paton91] Paton, N.W. & Diaz, O. Object-oriented databases and frame-based systems: comparison. *Information and Software Technology*. 33 (5):357-365, 1991.

[Prabh92] Prabhu, C.S.R. Semantic database systems: a functional introduction, Sangam Books Ltd., London, 1992.

[Preme90] Premerlani, W.J., et al. An object-oriented relational database. *Communications of the ACM*. 33 (11):99-109, 1990.

[Rasmu92] Rasmus, D.W. Relating to objects. *BYTE*. 17 (14):161-165, 1992.

[Relea89] Release 1.0. Object-oriented database roundup. 22 September 1989, EDventure Holdings Inc., New York.

[Rine92] Rine, D.C. & Bhargava, B. Object-oriented computing. *IEEE Computer*. 25 (10):6-10, 1992.

[Rotze90] Rotzell, K. Transactions and versioning in an ODBMS. *Proceedings of the Object-Oriented Database Task Group Workshop*, Ottawa, Canada, 1990, pp. 55-61.

[Rowe90] Rowe, L.A. & Stonebraker, M.R. The POSTGRES data model. *Research Foundations in Object-Oriented and Semantic Database Systems*, edited by A.F. Cárdenas and D. McLeod, Prentice-Hall, Englewood Cliffs, 1990.

[Schek90] Schek, H.-J., et al. The DASDBS project: objectives, experiences, and future prospects. *IEEE Transactions on Knowledge and Data Engineering*. 2 (1):25-43, 1990.

[SH90] Stone, C.M. & Hentchel, D. Database wars revisited. *BYTE*. 15 (10):233-242, 1990.

[Shipm81] Shipman, D. The functional data model and the data language DAPLEX. *ACM Transactions on Database Systems*. 6 (1):140-173, 1981.

[Simme92] Simmel, S.S. & Godard, I. Objects of substance. *BYTE*. 17 (14):167-170, 1992.

[Spier91] Spiers, J. Object-oriented databases: evolution or revolution? *Hypermedia/ Hypertext and Object-Oriented Databases*, edited by H. Brown, Chapman & Hall, London, 1991.

[Stefi86] Stefik, M. & Bobrow, D.G. Object-oriented programming: themes and variations. *AI Magazine*. 6 (4):40-62, 1986.

[Stone90] Stonebraker, M., Rowe, L.A. & Hirohama, M. The implementation of POSTGRES. *IEEE Transactions on Knowledge and Data Engineering*. 2 (1):125-142, 1990.

[Stone91] Stonebraker, M. & Kemnitz, G. The POSTGRES next-generation database management system. *Communications of the ACM*. 34 (10):78-92, 1991.

[Taylo91] Taylor, D.A. Object-oriented technology: a manager's guide, Addison-Wesley, Reading, MA, 1991.

[Taylo92] Taylor, D.A. Object-oriented information systems: planning and implementation, John Wiley & Sons, New York, 1992.

[Ullma87] Ullman, J.D. Database theory: past and future. Keynote Speech. *Proceedings of the Sixth ACM Symposium on Principles of Database Systems*, 1987, pp. 1-10.

[Ullma88] Ullman, J.D. Principles of database and knowledge-base systems, volume 1, Computer Science Press Inc., Rockville, MD, 1988.

[Unlan90] Unland, R. & Schlageter, G. Object-oriented database systems: concepts and perspectives. *Proceedings of the International Symposium on Database Systems of the 90s*. Lecture Notes in Computer Science, 466, pp. 154-197, Springer-Verlag, Berlin, 1990.

[Wells92] Wells, D.L., Blakeley, J.A. & Thompson, C.W. Architecture of an open object-oriented database management system. *IEEE COMPUTER*. 25 (10):74-82, 1992.

[White89] White, D. Is it time to abandon the poor relations? *Computing*. 2 February 1989, pp. 24-25.

[Wilki90] Wilkinson, K., Lyngbaek, P. & Hasan, W. The Iris architecture and implementation. *IEEE Transactions on Knowledge and Data Engineering*. 2 (1):63-75, 1990.

[Winbl90] Winblad, A.L., Edwards, S.D. & King, D.R. Object-oriented software, Addison-Wesley, Reading, MA, 1990.

[Works91] Workshop on objects in data management. *Proceedings of the Third Joint Meeting*, Anaheim, California, 1991.

[Zdoni90] Zdonik, S.B. & Maier, D. Fundamentals of object-oriented databases. *Readings in Object-Oriented Database Systems*, edited by S.B. Zdonik and D. Maier, Morgan-Kaufmann, San Mateo, CA, 1990.