Object-Oriented Databases and Their Applications

Johan van den Akker

Dutch National Centre for Mathematics and Computer Science (CWI), Amsterdam

Object-oriented databases are considered to be the next-generation database systems after relational databases. Object-orientation offers advantages in application areas too complex for relational databases. In this article, we introduce object-oriented databases. The main systems and application areas are also outlined. Furthermore, we discuss shortcomings of current systems and potential future developments of object-orientation.

Object-orientation means the combination of data and behaviour in objects, that have a close correspondence to real-world objects. It was first found in SIMULA [3] to structure computer programs for simulations. It was taken further by languages such as Smalltalk [8], C++ [14], and Eiffel [13] . Important for the introduction of object-orientation in databases were O_2 [7] and GemStone [12].

The first occurrence of object-oriented notions in databases is in the Entity-Relationship model [5]. Until the mid-eighties, object-orientation was only found in data modelling. Implementation of an information system was mainly done using relational databases. The broadening scope of information systems brought to light a number of shortcomings of relational systems for advanced applications, like design databases, manufacturing databases, and office automation systems. In such applications, an object-oriented database offers better facilities to model complex structures. A significant advantage of object-oriented databases is the encapsulation of operations with the data. This way, operations shared between programs are specified and stored in a single place. For example, most applications using a manufacturing database need to obtain the composing parts of an assembly. In this case, it has obvious advantages to specify a single operation for this together with the data specification of an assembly. Furthermore, object-orientation offers better facilities to view data at different abstraction levels. For example, we might want to view an aircraft design at the level of the complete aircraft, split up into wings, fuselage, engines etc., or completely 'exploded' into parts.

Research in object-oriented databases has not yet yielded a single, well-defined data model, as was achieved very early for the relational model [6]. Hence, standardisation has been actively pursued by both the industrial and the academic parts of the OODBMS community. This effort resulted in the ODMG [1] model [4]. ODMG defines an interface to and a data model for an OODBMS to promote portability of applications between DBMSs.

This absence of a single well-defined data model led to the formulation of an OODBMS' key properties in the often cited object-oriented database manifesto [1]. These are the following.

1. **Complex Objects.** A complex object is an object built from simpler ones. An example of a complex object is a `car` object that exists of other objects, viz., `part` objects. Complex objects can

1. Object Data Management Group, see `www.odmg.org` .

also be recursive. For example, in a design database, a subassembly object can consist of other subassembly objects. This construction of complex objects from other objects is called aggregation.

2. **Object Identity.** Object data models are based on identity as opposed to the relational model, which is value-based. In the relational model, two tuples are the same, if their attribute values are equal. In an object-oriented data model, two objects are the same if and only if their identities are the same.

3. **Encapsulation.** This has two aspects. The first aspect originates in abstract data types. It is concerned with the separation of interface and implementation. Additionally, this allows us to hide private data of an object. The second aspect is the combined specification of data and behaviour in an object. This is the important aspect from a database point of view.

4. **Types and Classes.** Two key notions in an OODBMS are types and classes. In an object-oriented system, a type is a specification of an object's features. The type of an object is often given as a tuple of attribute and method types. A class is a collection of objects, that is used to create and store objects.
   Usually, the notions of class and type are closely associated. The relation between types and classes can be in two directions. Commonly, objects in a class conform to the associated type, because they are instances of that class. Another approach is that objects belong to a class, because they conform to the associated type. This is the case in data models allowing arbitrary addition and deletion of attributes, methods and other elements of objects, such as Self [15] and Goblin [10].

5. **Class or Type Hierarchies.** Classes and types are part of hierarchies, formed by *inheritance*. A *subclass* inherits the features from a *superclass*, which means that it has the same data and behaviour, possibly extended with its own data and behaviour. Hence, the subclass is a *specialisation* of the superclass. Likewise, a superclass is a *generalisation* of its subclasses. Everywhere an object of the superclass is required, an object of the subclass can be used.

6. **Overriding, Overloading, and Late Binding**. With the separation of interface and implementations, subclasses of a superclass might have the interface of an operation in common, but have a different implementation. A classical example is a `display` operation for a `graphic` object, which is implemented differently by its subclasses `circle`, `triangle`, and `polygon` objects. Since the name `display` denotes different operations, it is said to be *overloaded*. If the `graphic` object implements its own, generic, `display` operation, then the subclasses are said to *override* this operation with their own definition.
   Overloading and overriding operations means choosing an implementation to execute for each invocation of the operation. For example, if we invoke the `display` operation on a `graphic` object, then we would like the system to execute the most-specific implementation, e.g., the `triangle` implementation of `display` for `trangle` object. This is achieved by *late binding*, which means that the implementation is chosen at the actual exection time.

7. **Computational Completeness.** An OODBMS must allow every computable function to be computed in its data manipulation language.

8. **Extensibility.** The user of the OODBMS must be able to define his own types. Furthermore, there is no distinction between the use of system-defined types and user-defined types.

9. **An OODBMS is a DBMS.** An OODBMS must support persistence, secondary storage management, concurrency, recovery, and an ad-hoc query facility.

The final requirement is stated as five separate requirements in the Manifesto, itself. The other requirements are on the data model, that also apply to object-oriented programming languages. Actually, this close relation to programming languages is one of the main advantages of an OODBMS over a relational DBMS for complex applications. Most programming languages use a data model, that is different from the relational model. In particular, the programming language does not have a set-construct, while results from relational queries are always sets. Hence, we have the so-called *impedance mismatch* between the programming language and the relational DBMS. Since an OODBMS uses the same data model as an OO programming language, the impedance mismatch is solved here.

Object-Oriented Databases in Practice

A number of commercial OODBMSs are currently on the market. Examples are O_2 [2], Gemstone, and ObjectStore. The construction of O_2 is discussed extensively in a book [2]. O_2 originally extended C and Lisp to function as database programming language. Nowadays, it conforms to ODMG and interfaces to the three most important OO languages, C++, Smalltalk, and Java. Gemstone [3] is based on Smalltalk. Gemstone has found application in manufacturing and network management. Further-

2. See `www.o2tech.fr`.
3. See `www.gemstone.com`.

more, the Smalltalk foundation promotes its use as an experimentation platform. An example is the use in the TRiGS [9] as a foundation for an active database system. Objectstore [4] has been strongly connected to C++, but is also extended with interfaces for other languages, viz., Java and ActiveX.

Typical applications for OODBMSs are characterised by high complexity of data. Hence, we find them in areas such as design, manufacturing, telecommunications, and financial services. For example, Texas Instruments uses Gemstone for a semiconductor fabrication management system. An example in telecommunications is the choice of ObjectStore by Deutsche Telekom as the database for their ISDN system management. In the financial sector, Swiss Life uses Informix in their life insurance management system. The websites of various database vendors all offer more application studies of their systems.

Shortcomings

This article has discussed object-oriented databases and their applications. However, OODBMSs have been relatively slow to take off in practice. This is due to a number of shortcomings in current OODBMS. Here, we discuss a number of these.

1. **Query Languages.** OODBMSs have mainly focussed on applications needing navigational access, such as design systems. Hence, set-oriented queries as found in relational databases have received less attention. Powerful features like nested queries, aggregation, and group-by are often not present. This issue needs to be tackled by the development of query languages for OODBMSs, preferably through an extension of SQL.
2. **Interoperability.** In most application environments, a new information system based on an OODBMS will need to interact with legacy systems. A particular concern is the cooperation with relational databases. A transitional data model is offered by object-relational database systems, such as Informix [5]. These combine object oriented features, such as an identity-based model and encapsulation of behaviour, with a relational model based on tables.
3. **Interface with Programming Languages.** Initially, most OODBMSs were closely integrated with one programming language. Although the close integration has its advantages in view of the impedance mismatch, it also means that the applications for the database must be written in that language. This shortcoming is countered by database vendors through integration off DBMSs with multiple programming languages and by standardisation efforts such as ODMG.
4. **Lack of Standards.** As discussed above, standardisation is only a recent development in object-oriented databases. There is not a common database interface, like SQL for relational databases. As a consequence, applications developed for one OODBMS do not work with another OODBMS. This shortcoming is being tackled by the ODMG standard, which most main database vendors intend to support.

4. See `www.odi.com`.
5. See `www.informix.com`.

Extensions of Object-Orientation

The current state-of-the-art in OODBMSs provides basic object-oriented features. In the future, OODBMSs will include new features currently developed in the database research community. The main focus is to create self-contained objects that define a complete piece of domain knowledge. These are also known as *business objects*. In data modelling terms, this leads to a focus on encapsulation of object properties. In fact, the encapsulation of methods is only a first step. For example, active databases [16] incorporate a production rule system in a database. Next-generation object-oriented databases will encapsulate production rules in objects as an additional feature of the object. Other candidates for encapsulation are constraints and lifecycle descriptions. Thus, an object becomes a self-contained description of part of an application domain.

In the long term, objects will evolve through active objects to intelligent agents. A database will no longer be a large piece of software. Instead, each piece of data is managed by its own agent. Specialised tasks in a database, like executing a query, will be the responsibility of a specialised agent. This paradigm is very appropriate for *ubiquitous computing*. If every artefact contains processing power, the natural place to store data on the artefact is the artefact itself. In such a computing environment, a large centralised database is not essential anymore for most applications.

Further Reading

For a general discussion of the object model in databases the reader is referred to the collection edited by Won Kim [11]. The Story of O_2 [2] gives a good overview of all issues in building an object-oriented database management system.

References

[1] Malcolm Atkinson, François Bancilhon, David DeWitt, Klaus Dittrich, David Maier, and Stanley Zdonik, 'The object-oriented database system manifesto'. In Won Kim, Jean-Marie Nicolas, and Shojiro Nishio, editors, *Deductive and Object-Oriented Databases: Proc. of the 1st Intl. Conf. (DOOD'89)*, pages 223-240, Kyoto, Japan, 1989. North-Holland.

[2] François Bancilhon, Claude Delobel, and Paris Kanellakis,

Building an Object-Oriented Database System: The Story of O2. Morgan Kaufmann, San Mateo, CA, USA, 1992.

[3] G.M. Birtwistle, O.-J. Dahl, and B. Myhrhaug, *{SIMULA} begin*. Studentlitteratur, Lund, Sweden, 1974.

[4] R.G.G. Catell, *The Object Database Standard: ODMG-93*. Morgan Kaufmann, San Mateo, CA, USA, 1994.

[5] Peter Chen, 'The entity-relationship model: Towards a unified view of data'. *ACM Transactions on Database Systems*, 1(1):9-36, 1976.

[6] E.F. Codd, 'A relational model for large shared data banks'. *Communications of the ACM*, 13(6):377-387, 1970.

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

[8] Adele Goldberg and David Robson, *Smalltalk-80: the language and its implementation*. Addison-Wesley, Reading, MA, USA, 1983.

[9] G. Kappel et.al., 'TriGS - making a passive object-oriented database system active'. *Journal of Object-Oriented Programming*, July/August 1994.

[10] Martin L. Kersten, 'Goblin: a dbpl designed for advanced database applications'. In Dimitris Karagiannis, editor, *Database and Expert Systems Applications, Proc. of the Intl. Conf.*, pages 354-349, Berlin, Germany, 1991. Springer.

[11] Won Kim, *Modern Database Systems: The Object Model, Interoperability, and Beyond*. ACM Press/Addison-Wesley, New York, USA, 1995.

[12] David Maier and Jacob Stein, 'Development and implementation of an object-oriented dbms'. In Bruce Shriver and Peter Wegner, editors, *Research Directions in Object-Oriented Programming*. MIT Press, 1987.

[13] Bertrand Meyer, *Object-oriented Software Construction*. Prentice Hall International, 1988.

[14] Bjarne Stroustrup, *The C++ Programming Language*. Addison-Wesley, Reading, MA, USA, second edition, 1991.

[15] David Ungar and Randall B. Smith, 'Self: The power of simplicity'. In Norman K. Meyrowitz, editor, *Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'87)*, volume 22 of *SIGPLAN Notices*, pages 227-242, Orlando, FL, USA, December 1987.

[16] Jennifer Widom and Stefano Ceri, *Active Database Systems: Triggers and Rules for Advanced Database Processing*. Morgan Kaufmann, San Francisco, CA, USA, 1995.

Drs. Johan van den Akker is a Ph.D. candidate at CWI, the Dutch National Centre for Mathematics and Computer Science, in Amsterdam. His main research interest is active object-oriented databases.