Research Directions in Object–Oriented Database Systems

Won Kim

Microelectronics and Computer Technology Corporation
3500 West Balcones Center Drive
Austin, Texas 78750

ABSTRACT

The set of object–oriented concepts found in object–oriented programming languages forms a good basis for a data model for post–relational database systems which will extend the domain of database applications beyond conventional business data processing. However, despite the high level of research and development activities during the past several years, there is no standard object–oriented data model, and criticisms and concerns about the field still remain. In this paper, I will first provide a historical perspective on the emergence of object–oriented database systems in order to derive a definition of object–oriented database systems. I will then examine a number of major challenge which remain for researchers and implementors of object–oriented database systems.

1. INTRODUCTION

The rapidly increasing use of object–oriented concepts in various components of software technology has naturally necessitated object–oriented database systems. There is a flurry of activities to build and market object–oriented database systems [RELE89, KIM90d]. There are several commercial products in various stages of readiness. There are at least a dozen industrial research projects around the world to prototype object–oriented database systems, and object–oriented database systems are also under development in a comparable number of university laboratories.

Despite all these activities, currently there is no standard for object–oriented databases; that is, there is no standard object–oriented database language in which to program applications. Further, the lack of standard semantics for object–oriented programming and the apparent lack of a computational formalism for object–oriented database systems have given rise to concerns about the validity of object–oriented database systems. These concerns certainly must and, I believe, can be addressed. The recent proposals for high–level rules for object–oriented database systems, [ATKI89] and [KIM90a, KIM90e], reflect the view that the field of object–oriented databases has matured enough during the past several years. I believe that database systems based on an object–oriented data model will take root in the near future as one form of post–relational database systems. The primary reason is that object–oriented programming and the object–oriented approach is the most promising solution available today to the problems of developing, modifying, and extending complex, data–intensive software systems [GOLD83, STEF86, MICA88, STRO88]; the use of object–oriented concepts in applications which generate and manage a very large number of objects and complex relationships among them will have to turn to database systems which directly capture the object semantics [KIM89a]. Further, there is a long list of shortcomings of current relational database systems; these shortcomings must be removed if the utility of database systems is to extend beyond the domain of business data processing.

In this paper, I will first provide a historical perspective on the emergence of object–oriented database systems. This will lead to a working definition of object–oriented database systems, and will allow some light to be shed on the current concerns about and prospects for object–oriented database systems. I will then outline a num-

ber of important topics of research which I hope other researchers will take up.

2. HISTORICAL BACKGROUND

I believe that the emergence of object-oriented database systems during the second half of the 1980s is due to the simultaneous coming of age of object-oriented programming and the push for post-relational database technology. The discovery of the limitations of the relational database systems and the need to manage a large volume of objects with object semantics found in object-oriented programming languages led to the introduction of commercial object-oriented database systems in the mid- to late-1980s. Let us briefly review the evolutionary history of database systems and the rationale for object-oriented programming.

2.1 Push for Object-Oriented Concepts

Object-oriented concepts have evolved in three different disciplines: first in programming languages, then in artificial intelligence, and then in databases. Simula-67 is generally regarded as the first object-oriented programming language. Since Simula-67, researchers in programming languages have taken two different paths to promote object-oriented programming [MICA88]. One was the development of new object-oriented languages, most notably Smalltalk. Another was the extension of conventional languages: Flavors and LOOPS as extensions of LISP; Objective C and C++ as extensions of C; and so on. After Marvin Minsky's introduction of frames [MINS75] as a knowledge representation scheme, researchers in artificial intelligence have developed such frame-based knowledge representation languages as KEE, ART, and so on. In the database area, research into semantic data models has led to data modeling concepts similar to those embedded in object-oriented programming and knowledge representation languages. The class concept captures the instance-of relationship between an object and the class to which it belongs; the concept of a subclass specializing its superclass captures the generalization (IS-A) relationship; and the composition of an object in terms of attributes captures the aggregation relationship. Some of the better-known semantic data models include E/R (entity-relationship) [CHEN76], SDM (semantic data model) [HAMM81], and DAPLEX [SHIP81]. Indeed, object-oriented concepts are the common thread linking frame-based knowledge representation and reasoning systems, object-oriented programming (application development) environments, and object-oriented advanced human interface systems. Therefore, they may be the key to building one type of intelligent high-performance programming system of the foreseeable future.

An object-oriented approach to programming is based on the concepts of encapsulation and extensibility. A program in general consists of data and code that operates on the data. Object-oriented programming encapsulates in an object some data, and programs to operate on the data: the data is the state of the object, and the code is the behavior of the object. The state of an object is the values of the attributes defined for the object, and the behavior is a set of programs with a well-defined interface for their invocation. Object-oriented programming requires the behavior of an object to be invoked via messages to the object through the interface defined for the object. For example, one may define a Shape object, whose state consists of the values of the attributes Center-Point and Bounding-Box for the object, and whose behavior includes a program called Display-Shape to display the object on a screen. One may cause a Shape object to Display itself on a screen by sending a message to invoke the Display-Shape behavior of the object. The notion of encapsulation has proven to be a natural and easy paradigm for various application environments, such as graphical user-interface systems.

Extensibility refers to the ability to extend an existing system without introducing changes to it. Extensibility is an especially powerful concept for building and evolving very large and complex software systems. An object-oriented approach to programming offers extensibility in two ways: behavioral extension and inheritance. The behavior of an object may be extended by simply including additional programs. A behavioral extension of an object does not affect the validity of any existing program. For example, a new program named Rotate-Shape may be added to the Shape object defined above. An object-oriented approach further promotes extensibility through reuse or inheritance. The behavior, and even the attributes, defined for an object may be reused in the definition of more specialized objects, that is, they may be inherited by new objects. For example, a new object named Triangle may be defined as a specialized Shape object. The Triangle object inherits (reuses) the behavior (Display-Shape and Rotate-Shape) and attributes (Center-Point and Bounding Box) defined for the Shape object. Further, one may define additional behavior and attributes for the Triangle object, and even re-define some of the inherited behavior and attributes.

2.2 Push for Next-Generation Database Systems

During the past three decades, database technology has undergone four generations of evolution, and the fifth generation of database technology is currently under development. The first generation was the file systems; the second generation was the hierarchical database systems; the third generation was the CODASYL database systems; and the fourth generation is the relational database systems. The second and third generation systems realized the sharing of an integrated database among many users within an application environment. The lack of data independence and the tedious navigational access to the database gave rise to the fourth-generation database technology. All four generations of database systems have been designed to meet the requirements of business data-processing applications, such as inventory control, payroll, accounts, and so on. The fifth-generation database technology will be characterized by a richer data model and a richer set of database facilities necessary to meet the requirements of applications beyond the business data-processing applications for which the first four generations of database technology have been developed.

The transition from one generation of database technology to the next has been marked by the offloading of some tedious and repetitive bookkeeping functions from the applications into the database system. This has made it easy for the application programmers to program database applications; however, it made the performance of database systems a major problem, and required considerable research and development efforts to increase the performance of the new generation databases to an acceptable level. This point is particularly true for the transition into the era of relational databases. The introduction of declarative queries in relational databases relieved application programmers of the tedious chore of programming navigational retrieval of records from the database. However, a major new component, namely the query optimizer, had to be added to the database system to automatically arrive at an optimal plan for executing any given query, such that the plan will make use of appropriate access methods available in the system.

During the 1970s research and development activities in databases were focused on realizing the relational database technology. These efforts culminated in the introduction of commercial relational database systems in the late 70s and early 80s. However, attempts to make use of the relational database technology in a wide variety of types of application have exposed several serious shortcomings of the relational and past-generation database technology. The shortcomings of conventional database technology became the subjects of database research and development during the 80s. The new applications include computer-aided design, engineering, software engineering, and manufacturing (CAD, CAE, CASE, and CAM - hereafter abbreviated as CAx) systems and applications that run on them; knowledge-based systems (expert systems and expert system shells); multimedia systems which deal with images, voice, and textual documents; real-time systems; programming language systems; and so on. Some of these applications require facilities for modelling and managing complex nested entities (such as design and engineering objects, and compound documents); a richer set of data types, that is, user-defined data types, and long unstructured data (such as images, audio, and textual documents); frequently useful semantic concepts (such as generalization and aggregation relationships); the concept of temporal evolution of data (i.e., temporal dimension of data, and versioning of data); and so on. Some of the applications are highly compute-intensive on a large volume of memory-resident data, and impose performance demands that cannot be met by conventional database systems. The environment of some of the applications requires long-duration, interactive, and cooperative transactions. Further, conventional database systems require application programs to be implemented in some algorithmic programming language and some database language embedded in it; the database languages are very different from the programming languages, in both the data model and data structures.

3. OBJECT-ORIENTED DATABASE SYTEMS

In this section, I will first set forth broad minimum requirements for object-oriented database systems. Then, on the basis of the historical background I outlined in Section 2, I will examine a broader definition, and address important reasons for the current concerns about the definition and validity of object-oriented database systems.

3.1 Minimum Requirements

I take the view that a database system must satisfy the following two broad conditions before it can be called an object-oriented database system.

1. It supports a core object-oriented data model.

2. It supports all the database features found in conventional

database systems, with their semantics extended/modified to be consistent with the semantics of the core object-oriented data model.

An object-oriented database system is a persistent and sharable repository and manager of an object-oriented database; and an object-oriented database is a collection of objects defined by an object-oriented data model, that is, objects that capture the semantics of objects supported in object-oriented programming. I note that [MAIE89b] questions whether it makes sense to even talk about a data model for object-oriented database systems. I, like most others [LECL88], will nevertheless use the term object-oriented data model to mean a logical organization of real-world objects (entities), constraints on them, and relationships among objects. A set of core object-oriented concepts forms the basis of an object-oriented data model for an object-oriented database system. I regard, as does [ATKI89], the following as core object-oriented concepts; as they are familiar concepts by now, I will not elaborate on them here.

1. Any real-world entity is uniformly modeled as an object, and is associated with a unique identifier.

2. Every object encapsulates a state and a behavior, where the state of an object is the set of values for the attributes of the object, and the behavior of an object is the set of methods (program code) which operate on the state of the object. The value of an attribute of an object is also an object in its own right. Further, an attribute of an object may take on a single value or a set of values.

3. All objects which share the same set of attributes and methods are grouped together as a class*, such that an object belongs to only one class as an instance of that class.

4. The domain (type) of an attribute of a class may be any class. The domain class may be a primitive class, such as integer, string, or boolean. It may be a general class with its own set of attributes and methods. The domain of an attribute of a class C may be the class C.

5. All the classes are organized as a rooted directed acyclic graph or a hierarchy (called a class hierarchy). A class inherits all the attributes and methods from its direct and indirect ancestors in the class hierarchy. Semantically, a class is a specialization (subclass) of the class(es) from which it inherits attributes and methods; conversely, a class is a generalization (superclass) of the classes which inherit attributes and methods from it. The class hierarchy must be dynamically extensible; that is, a new subclass can be derived from one or more existing classes.

6. The state and behavior encapsulated in an object may be accessed or invoked from outside the object only through explicit message passing. Further, it must support the run-time binding of a message to its corresponding method, since the method may have been inherited into the object from a superclass.

The evolutionary history of database systems has shown that the transition from one generation to the next has never resulted in the elimination of essential database features. Conventional database systems provide database languages to allow application programmers to define and manipulate the database. A conventional database language consists of three components (or sublanguages): data definition language for specifying the schema; query and data manipulation language for querying and updating the database; and data control language for transaction management, integrity control, authorization, and resource management. All these facilities must be provided for object-oriented database systems [KIM90a, KIM90d]. Beyond the features expressed in these database sublanguages, database systems provide various facilities, including concurrency control, recovery, error reporting, and so on. The semantics of these facilities in object-oriented database systems must be extended/modified to be consistent with the semantics of the core object-oriented concepts. I will elaborate on this point in Section 3.2.

3.2 Core Object-Oriented Concepts and Database System Architecture

The fact that under the core object-oriented data model, a class is both the root of an aggregation hierarchy and a class hierarchy has far-reaching consequences on the architecture of a database system. It impacts the query model [KIM89d], indexing [MAIE86b, KIM89b], authorization [RABI90], schema evolution [BANE87, ZICA89], concurrency control [GARZ88], and physical clustering [KIM90d]. In this section, I will make this concrete only for the query model and indexing. (The re-

---

* In this paper, I will use the terms class and type interchangeably. Although it is correct to differentiate them, the two concepts are often combined into one term.

mainder of this subsection is taken largely from [KIM90a].)

Query Model

A query is formulated against a portion of the database schema. Since the schema of an object-oriented database consists of a class hierarchy and an aggregation hierarchy rooted at each of the classes in the database, a query against an object-oriented database can be very rich. The semantics of a query in object-oriented databases is significantly different from and richer than a query in standard relational databases.

Let us consider a query against a single class; and for the time being let us ignore the class hierarchy. A query against a class will result in the retrieval of a set of instances of the class that satisfy user-specified search conditions. However, an instance of a class is a nested object consisting recursively of instances of the domains of the attributes of the instance. This means that a query against a class is really a query against the class and some of the domains of the attributes of the class. In other words, a query against a class is formulated against the nested definition of the class.

As an example, let us consider the schema of an example database as shown in Figure 1, and the query "Find all vehicles that weigh more than 7500 lbs, and that are manufactured by a company located in Detroit." This query is targeted against the class Vehicle, and associates the predicate 'Location = Detroit' with the class Company, and the predicate 'Weight > 7500' with the class Vehicle.

Next, let us introduce the concept of a class hierarchy to a query, but ignore the nested definition of the class. There are two meaningful interpretations for the scope of evaluation of such a query. One is obviously all instances of the target class. The other is all instances of the classes in the class hierarchy rooted at the target class. This interpretation is correct when the user means the specified target class to be the generalization of all direct and indirect subclasses of the target class. For example, the user may specify the class Vehicle as the target class in a query intending to retrieve all types of vehicle, including Vehicle, Automobile, Truck, etc. The interpretation of a class as the generalization of all its subclasses may be extended to the domain of an attribute. For example, when the user specifies the class Company as the domain of the Manufacturer attribute of the class Vehicle, the attribute may take on as its values objects from the class Company and any direct or indirect subclass of Company.

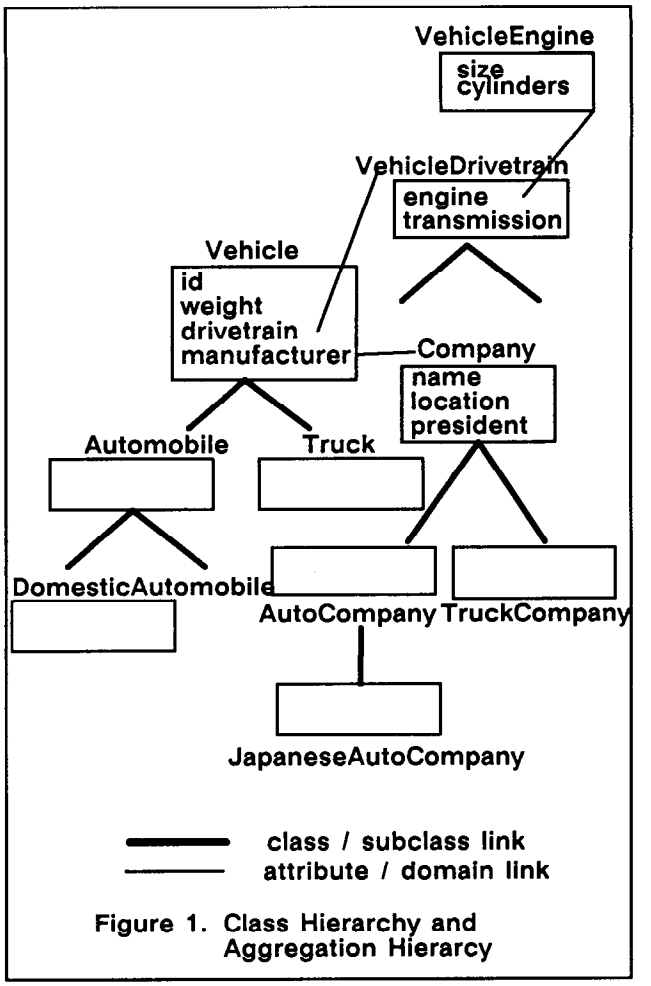

Figure 1. Class Hierarchy and Aggregation Hierarcy

Indexing

The aggregation and generalization relationships captured in an object-oriented data model require changes to the semantics of indexes. As I will show below, these relationships suggest different types of indexing: class-hierarchy indexing along a class hierarchy, and nested indexing along an aggregation hierarchy.

An index is maintained on an attribute (or a combination of attributes) of a class; an index is logically a list of pairs < key-value, list of object identifiers >, where the key-value is a value in the indexed attribute(s), and each identifier in the list of object identifiers is the identifier of an object in which the indexed attribute holds the key-value. For example, an index may be maintained on the Weight attribute of the class Vehicle to expedite the evaluation of queries with a search predicate involving the Weight attribute (e.g., Weight = 8000).

In relational database systems, one index is maintained on an attribute (or a combination of attributes) of one relation. This technique, if applied directly to an ob-

ject-oriented database, will mean that one index is needed for an attribute of each class. However, since the indexed attribute is common to all classes in the class hierarchy rooted at the user-specified target class, it makes sense to maintain one index on the attribute for all the classes in the class hierarchy rooted at the target class. An index on an attribute of a class and the class hierarchy rooted at the class is called a *class-hierarchy index.*

As we have seen earlier, although a query returns a set of instances of the target class, search predicates may be specified on any nested attribute of the class. Just as an index on an attribute of a class is useful for evaluating a query involving a predicate on the attribute, an index on a nested attribute of a class should be useful for a query involving a predicate on the attribute. An index on a nested attribute of a class is called a *nested-attribute index.* For example, the domain of the Manufacturer attribute of the class Vehicle is the class Company; and the class Company has the Location attribute. Then Location is a nested attribute of the class Vehicle.

3.3 Concerns

I believe that the current concerns about the definition and validity of object-oriented database systems are based on the following four major reasons. After stating them, I will discuss each of them in turn.

1. There is no standard object semantics.
2. Object-oriented database systems today are 'pot-boilers.'
3. The vendors of early systems offered systems which seriously lack in database features.
4. There is apparently no formal foundation for object-oriented database systems.

1. There is no standard object semantics.

Indeed, the only consensus about object semantics is that there is no consensus. However, there is a surprisingly high degree of agreement about a number of high-level concepts; these are what I call core object-oriented concepts. These concepts are found in many object-oriented programming languages and object-oriented database systems. I believe that, until a standard semantics of objects emerges, a reasonable starting point of research into object-oriented databases is the core object-oriented concepts. I will say more about the lack of standard object semantics in Section 4.1.

2. Object-oriented database systems today are 'pot-boilers' consisting of five orthogonal elements: object-oriented concepts, database features supported in conventional database systems, many of the database features that are in the list of shortcomings of conventional database systems, management of memory-resident objects, and object-oriented database programming language.

Around the mid-1980s object-oriented database systems became a rallying point for research and development activities in databases, which had been fragmented into a number of seemingly unrelated areas, such as extended relational databases, multimedia databases, spatial databases, engineering databases, logic databases, temporal databases, imprecise databases, real-time databases, database programming languages, and so on. That is, the push for object-oriented programming and the push for solutions to a long list of deficiencies in conventional database systems were translated into object-oriented database systems. It is for this reason that object-oriented database systems have become pot-boilers. I offer the following as an extended characterization of object-oriented database systems as they are currently being developed.

Object-Oriented Database System =
Object-Oriented Concepts +
Database Features Supported in
Conventional Database Systems +
Memory-Resident Object Management +
Features Specific to CAx Applications +
Object-Oriented Database Programming Languages

Note that the definition I gave in Section 3.1 required only the first two elements. The other three elements are embellishments and are certainly orthogonal to object-oriented concepts. Let us examine how these elements found their way into object-oriented database systems.

The impetus for object-oriented database systems came primarily from two application domains: object-oriented programming environments and CAx. The combined data management requirements of these two domains represent a significant part of the data management requirements unfulfilled by conventional database systems. The direct support of object semantics for a database of persistent and sharable objects is one of the data management requirements that are shared by the domains of object-oriented programming and CAx. The crying need for enhancing programmer productivity and for controlling exploding software complexity has led to the rise in interest in an object-oriented approach to the design and implementation of CAx systems, and to the integration of CAx artifacts and tools. The need to directly support object semantics in a database system to enhance programmer productivity and to control software com-

plexity is as strong a rationale as any which led to the transition in the past from one generation of database systems to the next.

The CAx applications demanded many of the data management requirements which cannot be satisfactorily met by conventional database systems. The core object-oriented concepts satisfy some of the data-modelling requirements: the aggregation relationship (complex nested objects) and generalization relationship. Additional data modeling requirements, which are not part of the core object-oriented concepts, include versions [CHOU86, KATZ86] and composite objects [KIM89c] which capture the version-of relationship and part-of relationship, respectively. Additional requirements, which are not related to data modeling, include long-duration transactions, checkout and checkin of objects between a shared database and private databases, change notification, and so on. Various object-oriented database systems operational today support or plan to support some or all of these features.

The two domains of applications which galvanized the activities in object-oriented database systems share one other important requirement, namely, management of memory-resident objects to support extensive computations. Some applications, such as interactive simulators and editors for VLSI designs, must traverse a large collection of objects, recursively from one object to other objects related to it. If, for example, relational database systems are used to manage objects for such applications, the applications have to use joins to express the traversal from one object to other objects [MAIE89a]. Obviously, the combined cost of crossing the boundary between the application and the database system, and joining one object with related objects, is simply intolerably expensive for such applications. A much better solution is to store logical object identifiers within the objects in the database, and convert them to memory pointers to related objects. That is, as an object is fetched from the database, the object identifiers embedded in the object are converted to memory pointers that will point to some descriptors for the objects that the object references (refers to). The referenced objects may later be fetched as necessary, if their descriptors are accessed. This is the approach developed to make objects persistent in the LOOM system [KAEH81]; this approach has been adopted and refined in ORION [KIM88b].

This particular aspect of object-oriented database systems has led some implementors of object-oriented database systems to the rather meaningless exercise of comparing the performance of memory-resident object management of their systems with relational database systems. It has also contributed mightily to the largely false impression that object-oriented database systems can only use navigational access to objects, unnecessarily inviting the criticism that object-oriented database systems are at odds with declarative queries and thus represent a throw-back to the days of hierarchical and CODASYL database systems [LAGU88]. I emphasize three points to clarify this concern. First, declarative queries can certainly augment the navigational access in object-oriented database systems, as evidenced by the declarative query languages which have been proposed and implemented in more recent object-oriented database systems, such as ORION [KIM90b], EXTRA/EXCESS [CARE88], and O2 [DEUX90]. Second, systems that manage memory-resident objects allow applications to directly access memory-resident objects; one should remember that conventional database systems do not allow applications to directly access objects in the page buffers. Third, systems that manage memory-resident objects extend the capabilities of database systems to these objects. Simply put, object-oriented database systems which manage memory-resident objects extend the capabilities of database systems to the virtual-memory workspace for the applications; these functions include queries, transaction management, integrity enforcement, and so on.

The database languages for (programmatic interfaces to) object-oriented database systems offered a seemingly excellent opportunity to remove the impedance mismatch [COPE84] found in the traditional embedding of a database language in some programming language. The reason is that object-oriented concepts already include such data modeling concepts, found in conventional database languages, as grouping objects into a class (corresponding to grouping records into a record type, or tuples into a relation), aggregation relationships between an object and objects it consists of (nested objects), and generalization relationships between a class of objects and classes of objects specialized from it. The idea of extending an object-oriented programming language into a unified (object-oriented) programming and database language naturally leads to the notion of an (object-oriented) database programming language [BLOO87, ATKI87, DANF88]. The fact that an object-oriented data model includes the aggregation and generalization relationships means that an object-oriented database system provides a user interface for the definition and manipulation of the relationships among objects. This in turn means that application programmers need not explicitly program and manage these relationships. For example,

the aggregation relationship is the basis of the recursive definition of a complex object in terms of other objects. This makes it possible to successively refine the representation of complex objects often found in such applications as computer-aided design and engineering and compound documents.

3. The vendors of early systems offered systems which seriously lack in database features.

In my view, the vendors of early object-oriented database systems misjudged both the size of the market for object-oriented database systems and the level of readiness to install systems for production use. They offered systems that lacked a host of database features which have proven their utility during the past three decades. These features primarily include transaction management, concurrency control, declarative queries, and automatic query optimization. This situation unfortunately reinforced the impression that object-oriented database systems are somehow incompatible with the semantics of traditional database features. The next-generation database technology must necessarily build on conventional database technology to meet the requirements of today's and newly emerging database applications.

4. There is apparently no formal foundation for object-oriented database systems.

I hold the view that a core object-oriented data model is really one type of extended relational model of data. The core object-oriented data model extends the conventional relational model of data as follows. The starting point for the extension is the approximate equivalence of a relation and a class.

1. The domain of an attribute of a relation is generalized to an arbitrary type, rather than a small set of system-defined primitive types. This leads to the aggregation hierarchy of classes, and nested objects as instances of the hierarchy.
2. The properties of a relation are extended beyond the attributes and constraints on the attributes by adding methods.
3. Each relation is extended with a system-defined attribute, the unique object identifier attribute. An object identifier or a set of object identifiers is stored for the value of an attribute of a class whose domain is an arbitrary class.
4.The classes are organized in a hierarchy called a class hierarchy via the generalization / specialization relationship between pairs of classes; the conventional relational model does not organize relations into a corresponding relation hierarchy.
5. The properties of a class are allowed to be recursively inherited by its subtypes (subclases); it allows inherited properties to be modified in subclasses and additional attributes and methods to be defined for the subclasses.
6. The attributes and methods of a class are required to be accessed and invoked only by messages using an explicitly defined set of interfaces for the class. If a message sent to an instance of a class is undefined for the class, it is sent up the class hierarchy to determine the class in which it is defined: this is called late binding of messages to objects.

To be sure, it is necessary to formalize a number of aspects of the core object-oriented data model; however, the core object-oriented data model is not exactly without foundation. The familiar proposals for extending the conventional relational model of data with such concepts as aggregation, generalization, and surrogates [SMIT77, CODD79] did not engender much controversy in the past. The points 1, 3, and 4 above are merely these same proposals. The idea of introducing a new type of attribute called a procedure has been extensively studied and incorporated into the POSTGRES extended relational database system [STON87b]; this idea is essentially point 2 stated above.

4. HURDLES FOR OBJECT-ORIENTED DATABASE SYSTEMS

There are currently at least two proposed approaches for transitioning from the fourth-generation database technology to the fifth-generation technology: extended relational database technology and object-oriented database technology. The extended relational approach starts with the relational model of data and a relational query language, and extends them in various ways to allow the modeling and manipulation of additional semantic relationships and database facilities. The POSTGRES system [STON86b, STON87c, ROWE87], which has been prototyped at the University of California at Berkeley, is the best-known next-generation database system which is based on the extended relational approach. The object-oriented approach starts with an object-oriented data model and a database language that captures it, and extends them in various ways to allow additional capabilities.

An object-oriented data model is a more natural basis than an extended relational model for addressing some of the deficiencies of conventional database technology previously outlined; for example, support for general data types, nested objects, and support for compute-intensive applications. (I note, however, that the underly-

ing data model has nothing to do with other deficiencies, such as long-duration transactions, support for long data, and temporal data.) Further, an object-oriented programming language may be extended into a unified programming and database language. The resulting language is subject to the problem of impedance mismatch to a far less extent than the approach of embedding a current-generation database language in one of conventional programming languages. An object-oriented database system which supports such a unified object-oriented programming and database language will be a better platform for developing object-oriented database applications than an extended relational database system which supports an extended relational database language.

I believe that both the extended relational and object-oriented approaches are viable, and that most likely systems adopting either approach will co-exist. The case for the extended relational approach is based largely on the fact that it is rooted on the familiar current-generation database technology; for the current-generation database technology, there is already a large user/customer base, and an industry-wide standard for the database language. This is something that proponents of the object-oriented next-generation database technology cannot claim. However, object-oriented programming and object-oriented approach to designing complex software systems are to date the most promising approach to coping with increasingly complex software systems and the corresponding costs of developing, maintaining, and evolving such software systems. If the object-oriented approach truly fulfills its promise in the design and implementation of software and representation of knowledge in future knowledge-based software, object-oriented database systems will become crucial.

Once relational systems are extended with abstract data types, procedures, nested objects, and object identifiers, the distinction between extended relational database systems and object-oriented database systems can become rather blurred; however, two differences are likely to remain. One is that extended relational database systems may support only some subset of the core object-oriented concepts, and such systems will definitely stop short of supporting the full semantics of the core object-oriented database systems. Another is that extended relational database systems will continue to support SQL-like data languages on what is essentially a conventional database architecture, and as such they are not likely to reduce impedance mismatch or to provide management of memory-resident objects.

There are two major hurdles that must be cleared before object-oriented database systems can claim its place in the era of the fifth-generation database systems. These are standardization and performance. I will discuss these issues below.

4.1 Standardization

Various groups, such as the ANSI standards committee on object-oriented databases, Object Management Group, and Open Software Foundation, are now attempting to forge standards for different components of the object-oriented technology, including object-oriented databases. Recently, the Common LISP Object System (CLOS) [MOON89] has been proposed as a standard object-oriented extension to Common LISP. Further, the popularity of C++ will significantly increase, now that AT&T has announced that it will support it [STRO86], and efficient compilers are becoming available, even in the public domain.

If one examines existing object-oriented programming languages, knowledge representation languages, and semantic data models, one can identify a small set of fundamental concepts whose primary semantics are common to almost all of them, that is, a set of core object-oriented concepts. A standard object-oriented data model will most certainly include these concepts, because these concepts have proven useful already. A useful launching point for research and development in object-oriented databases today is an object-oriented data model which includes at the minimum the core object-oriented concepts. An object-oriented database language may then be specified for the data model. An object-oriented database system may then be constructed to support the language. This is essentially the approach taken in all currently operational object-oriented database systems; although they differ in varying degrees, they pretty much share the same set of core concepts. I emphasize that, even if the eventual object-oriented model standard differs in various minor ways from the core model, the technology for implementing a database system which supports the core model will still be fairly straightforwardly applicable to a database system which will support the eventual standard. The reason is that the technology is based on only the primary semantics of the core object-oriented concepts.

4.2 Performance

Various components of an object-oriented database system must be architected to both support and take advantage of the object-oriented semantics. Secondary in-

dexing, physical clustering, concurrency control, query optimization, authorization, and object directory management are the primary components of an object-oriented database system which require new architectural techniques for satisfactory performance [KIM90c]. These, except for object directory management, are also important components of conventional database systems; however, the architectural techniques used in conventional systems are inadequate or inappropriate for object-oriented database systems. The discussion of secondary indexing illustrated this point; the class hierarchy and nested objects in object-oriented databases give rise to index structures that are designed to expedite the processing of queries involving them.

Object-oriented databases have been proposed as a platform for CAx applications and environments. To be sure, object-oriented database systems can satisfy the data modeling and performance requirements of many aspects of these applications. However, some of these applications pose seemingly impossible performance requirements to database systems, object-oriented or otherwise. They perform extensive computations on a large number of interrelated objects; they typically load all necessary objects in virtual memory first and then perform necessary computations on them. The computational paradigm of conventional database systems is to deliver one object at a time to the application, and does not match that of these applications. Some object-oriented database systems have been designed to bring the full powers of database systems to bear on in-memory objects. However, they incur substantially higher performance overhead than some of these applications can tolerate, since the overhead incurred to access a memory-resident object is still an order of magnitude higher than what is necessary for these applications, running without an underlying database system, to access an object in virtual memory by a few memory lookups.

5. FUTURE R/D DIRECTIONS

Despite the high level of research and development activities during the past several years, there remain various technical challenges, beyond standardization and performance, for researchers and implementors of object-oriented database systems. These challenges include database tools, migration path from conventional databases, formalization, additional database facilities, extensible architecture, and performance modeling; these are all relatively unexplored areas. (I have tried to mention in this section as many relevant topics as reasonable; however, I may have overlooked some worthy topics.) I will not belabor the need for further work on integrating the semantics of standard database features and the semantics of core object-oriented concepts, in particular, in the areas of (possibly rule-driven) query optimization [GRAE87], indexing and clustering techniques, concurrency control to better account for semantic modeling constructs, extension of authorization to account for mandatory and context-based security [THUR89], and so on.

5.1 Database Tools

The richness of an object-oriented data model is a mixed blessing. On the one hand, it makes it easier for the users to model their applications. On the other hand, the complexity of the object-oriented database schema, with the class hierarchy and aggregation hierarchies, significantly complicates the problems of logical and physical database design. Thus the need for friendly and efficient design aids for the logical and physical design of object-oriented databases is significantly stronger than that for relational databases. [KIM89b, BERT89] examines the properties of class-hierarchy indexes and nested-attribute indexes and their use in expediting the evaluation of object-oriented queries. The current research into storage structures for non-first normal form relational databases is certainly relevant for the physical design of object-oriented databases [DESH88]. The framework for the evolution of an object-oriented database schema discussed in [SKAR86, BANE87, PENN87, ZICA89] represents important first steps towards the logical design of object-oriented databases.

A programmatic interface to a database system is too low level a tool for application development, and vendors of conventional database systems offer a number of additional tools as higher level interfaces to a database system to help non-programmers to develop applications. These interfaces include forms, menus, and graphics. The IRIS [FISH89] and O2 [DEUX90] projects have developed first-generation graphical interfaces to their database systems to allow non-programmers to browse the database schema and issue queries. However, high-level user interfaces remain an area of research not only for object-oriented databases, but also for conventional databases.

5.2 Migration Path from Conventional Databases

A system for managing a heterogeneous mix of databases is important as a migration path between relational databases to object-oriented databases, and is essential

for object-oriented database systems to take root. One viable approach is the development of an object-oriented SQL which is compatible with SQL; however, this approach will leave the impedance-mismatch problem unresolved. A preliminary proposal for such a language is reported in [BEEC88].

Another interesting line of research is the object-oriented approach to the management of a heterogeneous mix of databases, in particular, a mix of object-oriented databases and conventional databases. It is highly desirable to allow the user to access a heterogeneous mix of databases under the illusion of a single common data model [DAYA84, LITW88]. For example, suppose that an Employee database is managed by a relational database system, a Product database is managed by a hierarchical database system, and a Company database is managed by an object-oriented database system. An object-oriented data model may be used as the common data model for presenting the schemas of these different databases to the user. The richness of an object-oriented data model makes it appropriate for use as the common data model for representing a broad range of data models. Further, since object-oriented design and programming promotes extensibility, it may be used for designing and implementing a system to manage a heterogeneous mix of databases which can accommodate the addition of new types of database.

5.3 Formalization

The primary aspect of the core object-oriented data model which requires a formal basis is the query model which takes into account methods, and both the aggregation and class hierarchies. There have been a few recent proposals for such a query model which should provide a good basis for further research [KIM89d, CLUE89, SHAW89]. The query model for a core object-oriented data model is more general than that for the nested relational data model for two reasons. First, the aggregation hierarchy is actually a graph which admits cycles, whereas a nested relation is a strict hierarchy. Second, the core object-oriented data model includes a generalization hierarchy, which the nested relational data model does not have. Therefore, the lower bound for the expressive power of a query language for a core object-oriented data model is that of a nested relational query language.

The concepts of extensible class hierarchy, inheritance along the class hierarchy, and late binding along the class hierarchy are new concepts to database systems which do require precise analysis of semantics. Further, the notions of integrity constraints should be defined anew for the core object-oriented schema which now includes an aggregation hierarchy and a class hierarchy for every class in the database.

Recently, there have been worthy efforts to provide a logic-based formalism for the core object-oriented concepts by extending first-order logic with the semantics of object identity, inheritance, nested objects, and declarative functions [KIFE89a, KIFE89b, ABIT89, ZANI89]. In view of the power of predicates for expressing constraints, the discovery that the core object-oriented concepts can indeed be expressed using the logic formalism was perhaps inevitable. This line of research is also valuable to lay the foundation for the integration of rules (goal-directed reasoning) into object-oriented databases.

5.4 Additional Database Features

Views

In relational databases, a view is defined as a "virtual relation" derived by a query on one or more stored relations. The relational operations join, select, and project may be used to define a view. Views have been used for data protection and as a shorthand for queries. A query may be issued against views just as though they were relations. Further, authorization may be granted and revoked on views as on relations, allowing content-based authorization on stored relations. Views are also used as external schemas derived from an underlying schema.

Views are useful for object-oriented databases for similar reasons. Views may be used to specify logical partitioning of the instances of a class. Views may be used to define content-based authorizations in object-oriented databases; that is, views allow only the objects that satisfy specified authorization conditions to be made visible. The view mechanism may be used as one form of schema versioning [KIM88a] to allow the users of a database to experiment with schema changes without introducing permanent changes to the contents of the database. To the best of our knowledge, no object-oriented database system supports views at this time; in fact, I do not know at this time of any published account of research into views in object-oriented databases.

Semantic Modeling

One area of research, although rather tangential to object-oriented databases, is to augment the data modeling power of the core object-oriented concepts with semantic data modeling concepts, particularly, versions and composite objects. There may be additional semantic modeling concepts which are important for some major

applications; a worthy example is the modeling of roles [PERN90]. Just as versions and composite objects necessitated extensions and changes to the architecture of a database system, we expect that some of the additional semantic modeling concepts will have further impacts on the architecture of an object-oriented database system, including query evaluation, storage structures, and concurrency control.

Deductive Capabilities

Deductive databases has been an area of active research during the past decade. The objective of deductive databases is to integrate rules and traditional database of facts in support of complex reasoning which is not possible with queries against a traditional database. Much of the research has been focused on extending logic programming with database facilities [BANC86, TSUR86]. A rule system has been interfaced to ORION [BALL88], a rule manager has been incorporated into POSTGRES [STON87a], and IRIS presently provides a rudimentary support for rules [FISH89]. However, there is no object-oriented database system that can be regarded as a deductive database system. An object-oriented database system will become a deductive object-oriented database system once it can directly support rules and various reasoning concepts, such as truth maintenance and contradiction resolution. The current research is focused on extending an object-oriented database interface with rule specification and rule invocation. The issues of embedding various reasoning concepts, besides the forward and backward chaining of rules, directly in a database system have not been explored.

5.5 Extensible Architecture

Abstract Data Types

As pointed out in [BLOO87], the creation of user-defined types (often called abstract data types) has some difficult and interesting consequences on database system architecture. [STON86a] discusses how abstract data types may be incorporated into a relational database system. Abstract data types often require type-specific data structures for storing objects. Much of the past research into efficient implementation of abstract data types has been concerned with rectangular shapes in the context of VLSI layouts [STON83, BANE86]. The problem of efficiently implementing user-defined operations on user-defined types is an open-ended one, specific to the semantics of user-defined types. An interesting issue is to integrate user-defined predicates on user-defined types into the optimization framework which has been developed for standard queries, that is, queries involving involving atomic data types and standard comparison operators. To the best of my knowledge, this issue has not been seriously addressed.

Semantic Extensions

Designers of database systems have traditionally shied away from capturing the semantics of data, largely because it has been very difficult to find consensus among users. One example is versions. There is no consensus on the semantics of any type of version: versions of a single object, versions of a composite object [KIM89c], versions of a class [SKAR86], and versions of schema [KIM88a]. The few systems which have implemented versions (only of single objects), namely, ORION [CHOU88] and IRIS [FISH89], have implemented their own models of versions. Since the semantics of versions tend to differ in varying degrees from installation to installation, a worthwhile approach may be to provide a layered architecture for versions. The lower level may support a basic mechanism for low-level version semantics that are common to various proposals; the higher level may be made extensible to allow easy tailoring of installation-specific version semantics. This approach should be explored not only for versions but also for other semantic modeling concepts, such as composite objects, and even authorization; this line of research will complement the current research into extensible database systems [CARE86, BATO88, LIND87].

5.6 Performance Benchmarking

There is a clear need for performance modeling, and a meaningful and common benchmark for object-oriented database systems which will improve on the preliminary benchmarks [RUBE87]. Relational database benchmarks, such as the Wisconsin benchmark [BITT83], cannot really be used for object-oriented database systems, since operations supported in object-oriented database systems and relational database systems only partially overlap. For example, relational systems do not support operations based on concepts such as inheritance (class hierarchy), methods, object navigation (through object identifiers), and nested objects (hierarchy of inter-related objects). Further, object-oriented database systems tend to be designed to support compute-intensive applications on memory-resident objects in their in-memory data structure; relational systems have not fared well for such applications. The benchmark for object-oriented database systems should not only be meaningful for object-oriented databases, but also be useful in allowing a

meaningful comparison with conventional database systems.

6. SUMMARY

The area of object-oriented databases has come a long way during the past several years. After a few years of confusion, concerns, and high expectations, the area shows signs of reasonable maturity. In this paper, I offered both a minimal and extended definition of object-oriented database systems on the basis of a reflection on the history of evolution of database systems and the emergence of object-oriented programming. On the basis of these definitions, I addressed major reasons for the concerns and questions about the field of object-oriented database systems that have persisted, and outlined the impact of object-oriented concepts on database system architecture. Then I discussed the issues of standardization and performance, the two fundamental hurdles that exist today for object-oriented database systems, and outlined several areas of research which not been well explored thus far.

REFERENCES

[ABIT89] Abiteboul, S., and P. Kanellakis. "Object Identity as a Query Language Primitive," in *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, Portland, Oregon, May 1989.

[ATKI87] Atkinson, M., and P. Buneman. "Types and Persistence in Database Programming Languages," *ACM Computing Surveys*, vol. 19, no. 2, June 1987.

[ATKI89] Atkinson, M., et al. "Object-Oriented Database System Manifesto," in *Proc. 1st Intl. Conf. on Deductive and Object-Oriented Databases*, Kyoto, Japan, Dec. 1989.

[BALL88] Ballou, N., et al. "Coupling an Expert System Shell with an Object-Oriented Database System," Journal of Object-Oriented Programming, vol. 1, no. 2, June/July 1988, pp. 12-21.

[BANC86] Bancilhon, F., and R. Ramakrishnan. "An Amateur's Introduction to Recursive Query Processing Strategies," in *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, Washington, D.C., 1986.

[BANE86] Banerjee, J., and W. Kim. "Supporting VLSI Geometry Operations in a Database System," in *Proc. 2nd Intl Conference on Data Engineering*, Feb. 1986, Los Angeles, Calif.

[BANE87] Banerjee, J., W. Kim, H.J. Kim, and H.F. Korth. "Semantics and Implementation of Schema Evolution in Object-Oriented Databases," in *Proc. ACM SIGMOD Intl. Conf. on Management of Data,* San Francisco, Calif., May 1987.

[BATO88] Batory, D., et al. "GENESIS: An Extensible Database Management System," *IEEE Trans. on Software Engineering*, vol. 14, no. 11, Nov. 1988.

[BEEC88] Beech, D. "OSQL: A Language for Migrating from SQL to Object Databases," in Proc. Intl. Conf. on Extending Data Base Technology, Venice, Italy, March 1988.

[BERT89] Bertino, E., and W. Kim. "Indexing Techniques for Queries on Nested Objects," *IEEE Trans. on Knowledge and Data Engineering*, Oct. 1989.

[BITT83] Bitton, D., D. DeWitt, and C. Turbyfill. "Benchmarking Database Systems: A Systematic Approach," in *Proc. Intl. Conf. on Very Large Data Bases*, Oct. 1983.

[BLOO87] Bloom, T., and S. Zdonik. "Issues in the Design of Object-Oriented Database Programming Languages," in *Proc. 2nd Intl. Conf. on Object-Oriented Programming Systems, Languages, and Applications,* Orlando, Florida, Oct. 1987.

[CARE86] Carey, M., D. DeWitt, J.E. Richardson, and E.J. Shekita. "Object and File Management in the EXODUS Extensible Database System," *Proc. 12th Intl Conf. on Very Large Data Bases,* August 1986, Kyoto, Japan, pp. 91-100.

[CARE88] Carey, M., D. DeWitt, and S. Vandenberg. "A Data Model and Query Language for EXODUS," in *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, Chicago, Ill., June 1988, pp. 413-423.

[CHEN76] Chen, P. "The Entity-Relationship Model: Toward a Unified View of Data," *ACM Trans. on Database Systems*, vol. 1, no. 1, Jan. 1976, pp. 9-36.

[CHOU86] Chou, H.T., and W. Kim. "A Unifying Framework for Versions in a CAD Environment," in *Proc. Intl Conf. on Very Large Data Bases*, August 1986, Kyoto, Japan.

[CHOU88] Chou, H.T., and W. Kim. "Versions and Change Notification in an Object-Oriented Database System," in *Proc. Design Automation Conference*, June 1988.

[CLUE89] Cluet, S., C. Delobel, C. Lecluse, and P. Richard. "Reloop: An Algebra-Based Query Language for an Object-Oriented Database System," in *Proc. 1st Intl. Conf. on Deductive and Object-Oriented Databases*, Kyoto, Japan, Dec. 1989.

[CODD79] Codd, E.F. "Extending the Relational Model to Capture More Meaning," *ACM Trans. on Database Systems*, vol. 4, no. 4, Dec. 1979.

[COPE84] Copeland, G., and D. Maier. "Making Smalltalk a Database System," in *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, June 1984, pp. 316-325.

[DANF88] Danforth, S., and C. Tomlinson. "Type Theories and Object-Oriented Programming," *ACM Computing Surveys*, vol. 20, no. 1, March 1988, pp. 29-72.

[DAYA84] Dayal, U., H. Hwang. "View Definition and Generalization for Database Integration in a

Multidatabase System," *IEEE Trans. on Software Engineering*, vol. SE-10, no. 6, Nov. 1984, pp. 628-645.

[DESH88] Deshpande, A., and D. Van Gucht. "An Implementation for Nested Relational Databases," in *Proc. Intl. Conf. on Very Large Data Bases*, 1988.

[DEUX90] Deux, O., et al. "The Story of O2," *IEEE Trans. on Knowledge and Data Engineering*, March 1990.

[FISH89] Fishman, D., et al. "Overview of the IRIS DBMS," *Object-Oriented Concepts, Applications, and Databases*, (ed. W. Kim, and F. Lochovsky), Addison-Wesley, 1989.

[GARZ88] Garza, J., and W. Kim. "Transaction Management in an Object-Oriented Database System," in *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, Chicago, Ill., June 1988.

[GOLD83] Goldberg, A. and D. Robson. *Smalltalk-80: The Language and its Implementation*, Addison-Wesley, Reading, MA 1983.

[GRAE87] Graefe, G., and D. DeWitt. "The EXODUS Optimizer Generator," in *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, San Francisco, Calif., May 1987.

[HAMM81] Hammer, M., and D. McLeod. "Database Description with SDM: A Semantic Data Model," ACM Trans. on Database Systems, vol. 6, no. 3, Sept. 1981.

[HULL87] Hull, R., and R. King. "Semantic Database Modeling: Survey, Applications, and Research Issues," *ACM Computing Surveys*, vol. 19, no. 3, Sept. 1987, pp. 201-260.

[KAEH81] Kaehler, T. "Virtual Memory for an Object-Oriented Language," *BYTE*, pp. 378-387, August 1981.

[KATZ86] Katz, R., E. Chang, and R. Bhateja. "Version Modeling Concepts for Computer-Aided Design Databases," In *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, Washington, D.C., May 1986, pp. 379-386.

[KIFE89a] Kifer, M., and J. Wu. "A Logic for Object-Oriented Logic Programming," in *Proc. ACM SIGACT-SIGMOD-SIGART Sympo. on Principles of Database Systems*, March 1989.

[KIFE89b] Kifer, M., and G. Lausen. "F-Logic: A Higher-Order Language for Reasoning about Objects, Inheritance, and Scheme," in *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, Portland, Oregon, May 1989.

[KIM88a] "Versions of Schema for Object-Oriented Databases," (with H.T. Chou) in *Proc. Intl. Conf. on Very Large Data Bases*, Long Beach, Calif., Sept. 1988.

[KIM88b] Kim, W., et al. "Integrating an Object-Oriented Programming System with a Database System," in *Proc. 3rd Intl. Conf. on Object-Oriented Programming Systems, Languages, and Applications*, San Diego, Calif., Sept. 1988.

[KIM89a] Kim, W. and F. Lochovsky. *Object-Oriented Concepts, Applications, and Databases*, (ed. W. Kim, and F. Lochovsky), Addison-Wesley, 1989.

[KIM89b] Kim, W., K.C. Kim, and A. Dale. "Indexing Techniques for Object-Oriented Databases," *Object-Oriented Concepts, Applications, and Databases*, (ed. W. Kim, and F. Lochovsky), Addison-Wesley, 1989.

[KIM89c] Kim, W., E. Bertino, and J. Garza. "Composite Objects Revisited," in *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, Portland, Oregon, June 1989.

[KIM89d] Kim, W. "A Model of Queries for Object-Oriented Databases," in *Proc. Intl. Conf. on Very Large Data Bases*, August 1989, Amsterdam, Netherlands.

[KIM90a] Kim, W. "Object-Oriented Database Systems: Mandatory Rules and Prospects," *Datamation*, Jan. 15 and Feb. 1, 1990

[KIM90b] Kim, W. et al. "Architecture of the ORION Next-Generation Database System," *IEEE Trans. on Knowledge and Data Engineering*, March 1990.

[KIM90c] Kim, W. "Architectural Issues in Object-Oriented Databases," *Journal of Object-Oriented Programming*, March/April, 1990.

[KIM90d] Kim, W. *Introduction to Object-Oriented Databases*, MIT Press, May 1990.

[KIM90e] Kim, W. "Object-Oriented Databases: Definition and Research Directions," *IEEE Trans. on Knowledge and Data Engineering*, June 1990.

[LAGU88] Laguna Beach Report on the Future Directions for Database Research, presented as a panel position paper at the *Intl. Conf. on Very Large Data Bases*, Long Beach, Calif. Sept. 1988.

[LECL88] Lecluse, C., P. Richard, and F. Velez. "O2, an Object-Oriented Data Model," in *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, Chicago, Ill, June 1988, pp. 424-433.

[LIND87] Lindsay, B., J. McPherson, and H. Pirahesh. "A Data Management Extension Architecture," in *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, San Francisco, Calif., May 1987, pp. 220-226.

[LITW88] Litwin, W. "From Database Systems to Multidatabase Systems: Why and How," in *Proc. 6th British National Conf. on Databases*, July 1988, pp. 161-188.

[MAIE86a] Maier, D. "A Logic for Objects," in *Proc. Workshop on Foundations of Deductive and Logic Programming*, Washington, D.C., August, 1986.

[MAIE86b] Maier, D., and J. Stein. "Indexing in an Object-Oriented DBMS," in *Proc. Intl. Workshop on Object-Oriented Database Systems, Languages*, Pacific Grove, Calif., Sept. 1986.

[MAIE89a] Maier, D. "Making Database Systems Fast Enough for CAD Applications," *Object-Oriented Concepts, Applications, and Databases*, (ed. W. Kim, and F. Lochovsky), Addison-Wesley, 1989.

[MAIE89b] Maier, D. "Why Isn't There an Object-Oriented Data Model," in *Proc. IFIP 11th World Computer Congress*, San Francisco, Calif., August 1989.

[MICA88] Micallef, J. "Encapsulation, Reusability, and Extensibility in Object-Oriented Programming Languages," *Journal of*

Object-Oriented Programming, Vol. 1, No. 1, April/May 1988, pp. 12–36.

[MINS75] Minsky, M. "A Framework for Representing Knowledge," *The Psychology of Computer Vision*, (ed. P. Winston), McGraw-Hill, New York, 1975.

[MOON89] Moon, D. "The Common LISP Object-Oriented Programming Standard System," *Object-Oriented Concepts, Applications, and Databases*, (ed. W. Kim, and F. Lochovsky), Addison-Wesley, 1989.

[PENN87] Penney, J., and J. Stein. "Class Modification in the GemStone Object-Oriented DBMS," in *Proc. 2nd Intl. Conf. on Object-Oriented Programming Systems, Languages, and Applications*, Orlando, Florida, Oct. 1987.

[PERN90] Pernici, B. "Objects with Roles," in *Proc. ACM Conf. on Office Information Systems, Cambridge*, Mass., April 1990.

[RABI90] Rabitti, F., E. Bertino, W. Kim, and D. Woelk "A Model of Authorization for Next-Generation Database Systems," to appear in *ACM Trans. on Database Systems*.

[RELE89] Release 1.0, Sept. 1989, EDventure Holdings, Inc., 375 Park Ave., New York, NY.

[ROWE87] Rowe, L., and M. Stonebraker. "The POSTGRES Data Model," in *Proc. Intl. Conf. on Very Large Data Bases*, Brighton, England, Sept. 1987, pp. 83–95.

[RUBE87] Rubenstein, W., M. Kubicar, and R. Cattell. "Benchmarking Simple Database Operations," in *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, San Francisco, Calif., May 1987, pp. 387–394.

[SELI79] Selinger, P.G. et. al. "Access Path Selection in a Relational Database Management System," in *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, Boston, Mass. pp 23–34, 1979.

[SHAW89] Shaw, G., and S. Zdonik. "Object-Oriented Queries: Equivalence and Optimization," in *Proc. 1st Intl. Conf. on Deductive and Object-Oriented Databases*, Kyoto, Japan, Dec. 1989.

[SHIP81] Shipman, D. "The Functional Data Model and the Data Language DAPLEX," *ACM Trans. on Database Systems*, vol. 6, no. 1, March 1981.

[SKAR86] Skarra, A., and S. Zdonik. "The Management of Changing Types in an Object-Oriented Database," in *Proc. 1st Intl. Conf. on Object-Oriented Programming Systems, Languages, and Applications*, Portland, Oregon, Oct. 1986.

[SMIT77] Smith, J., and D. Smith. "Database Abstraction: Aggregation and Generalization," ACM Trans. on Database Systems, vol. 2, no. 2, June 1977, pp. 105–133.

[STEF86] Stefik, M. and D.G. Bobrow. "Object-Oriented Programming: Themes and Variations," *The AI Magazine*, January 1986, pp. 40–62.

[STON83] Stonebraker, M., B. Rubenstein, and A. Guttman. "Application of Abstract Data Types and Abstract Indices to CAD Data Bases," *in Proc. Databases for Engineering Applications*, Database Week 1983 (ACM), San Jose, Calif., May 1983,.

[STON86a] Stonebraker, M. "Inclusion of New Types in Relational Database System," in *Proc. 2nd Intl. Conf. on Data Engineering*, Los Angeles, Calif., Feb. 1986, pp. 262–269.

[STON86b] Stonebraker, M., and L. Rowe. "The Design of POSTGRES," in *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, Washington, D.C., May 1986.

[STON87a] Stonebraker, M., E. Hanson, and C.H. Hong. "The Design of the POSTGRES Rule System," in *Proc. Intl. Conf. on Data Engineering*, Feb. 1987, Los Angeles, Calif., pp. 356–374.

[STON87b] Stonebraker, M., J. Anton, and E. Hanson. "Extending a Database System with Procedures," *ACM Trans. on Database Systems*, vol. 12, no. 3, Sept. 1987.

[STON87c] Stonebraker, M. "The Design of the POSTGRES Storage System," in *Proc. Intl. Conf. on Very Large Data Bases*, Brighton, England, Sept. 1987.

[STRO86] Stroustrup, B. *The C++ Programming Language*, Addison-Wesley, Reading, Mass. 1986.

[STRO88] Stroustrup, B. "What is Object-Oriented Programming?," IEEE Software, May 1988, pp. 10–20.

[THUR89] Thuraisingham, M.B. "Mandatory and Discretionary Security Issues in Object-Oriented Database Systems," in *Proc. Object-Oriented Programming Systems, Languages, and Applications*, Oct. 1989, New Orleans, Louisiana.

[TSUR86] Tsur, S., and C. Zaniolo. "LDL: a Logic-Based Data Language," in *Proc. 12th Intl. Conf. on Very Large Data Bases*, August 1986, Kyoto, Japan.

[WOEL87] Woelk, D., and W. Kim. "Multimedia Information Management in an Object-Oriented Database System," in *Proc. Intl. Conf. on Very Large Data Bases*, Brighton, England, Sept. 1987, pp. 319–329.

[ZANI89] Zaniolo, C. "Object Identity and Inheritance in Deductive Databases -- An Evolutionary Approach," in *Proc. 1st Intl. Conf. on Deductive and Object-Oriented Databases*, Kyoto, Japan, Dec. 1989.

[ZICA89] Zicari, R. "Schema Updates in the O2 Object-Oriented Database System," Altair Technical Report no. 89-057, Oct. 1989.