OBJECTS IN TIME

by

James Clifford

and

Albert Croker

Information Systems Area
Leonard N. Stern School of Business
New York University
90 Trinity Place
New York, NY 10006

October 1988

Center for Research on Information Systems
Information Systems Area
Leonard N. Stern School of Business
New York University

Working Paper Series

CRIS #190
GBA #88-99

Center for Digital Economy Research
Stern School of Business
Working Paper IS-88-99

Objects in Time

James Clifford and Albert Croker

Information Systems Area
Graduate School of Business Administration
New York University
New York, New York 10006

Abstract

Two recent lines of database research, proceeding independently, have been concerned with providing a richer, more intuitive view of information at the user level. Historical database research has focused on ways to provide users with a view of information anchored and evolving in the temporal dimension. Object-oriented database research focuses on encapsulating both the structure and the behavior of the objects that users intend to model. In this paper we explore how these two lines of research might be brought together, providing to the user the representation and management of *objects in time.*

1. Introduction

Various proposals have been made for incorporating a temporal component into a database system [BADW82,McK86,TAI87]. Usually these proposals have been defined as extensions to the relational data model. In this paper we discuss the modeling of historical data in the context of an object-oriented data model.

Most object-oriented systems (see, for example, [Dit86] and [MSOP86]) owe their origins to the programming language *Smalltalk* [GR83]. Objects, the basic data constructs used in these systems, have proven to be both a powerful and flexible modeling construct. The power of objects arise in part from their ability to encapsulate both structure and behavior. The flexibility with which objects can be used as modeling constructs is due to the sets of data types that can be combined to define objects, the ability to nest the structure of objects, and the ability to encapsulate operations or *methods* on these objects in the manner of abstract data types. Objects are defined as consisting of values that are themselves objects; this nesting terminates with a set of **primitive objects**, such as integers, reals, and characters, that are built into the system. Thus an object that is used to represent an *employee* entity can include a component, say *salary*, whose value is an object representing the salary of that employee.

Object-oriented databases, like other types of databases, are used to model some aspect of the world. Each relevant entity and relationship in the modeled world is represented as an object in the database. Over time the various entities and relationships modeled by

Center for Digital Economy Research
Stern School of Business
Working Paper IS-88-99

the system may change. For example, when modelling employee entities, it is likely that employees may change departments, and that their salaries can be expected to change from time to time.

The traditional view of a database is that its state reflects that of the world at some specific, real or imaginary, instance of time; this instance being determined by the last update to the database. With each update the previous state of the database is lost. In contrast to this view, the state of a historical database models the world as it exists and has existed over some specified period of time [CW83]. If the equating of database objects with entities and relationships is to be retained in the context of an object-oriented database, then it is necessary for these objects to model the evolution of these real world objects over time.

In this paper we show how database objects can be defined in such a way that they can be viewed meaningfully in the context of a historical database. We call objects that are defined in this way **historical objects**. In addition to defining historical objects we also address various issues relating to their use in representing historical data. We do not present a formal model for integrating a treatment of time with a treatment of data as objects; to do so would be premature. Rather we discuss, from an intuitive point of view, those temporal aspects and properties which we believe ought to be captured by any system intended to represent our intuitive notions of "objects" and how they exist in time.

In the remainder of this paper we define what we mean by a historical object, and discuss several issues related both to the structure and to the manipulation of such historical objects.

2. Historical Objects

2.1. Introduction

Like the entities and relationships that they model, an object is characterized by some set of properties. We shall refer to these properties as **attributes.**[1] For example, if an object $\mathcal{O}$ corresponds to some employee, say Karen, then the attributes of $\mathcal{O}$: *NAME*, *SALARY*, *DEPT*, and *MGR* correspond to similarly named properties of the entity that is Karen.

The notion of a key is not inherent to objects. It is possible for two objects of the same type to denote the same values for their corresponding attributes. In lieu of the standard notion of a key, objects are distinguished by an **essence**. (We discuss object essences in the next section.)

Under the traditional view of a database an attribute of an object denotes a single value, that is for consistency also viewed as an object. The attribute *SALARY* in the object representing the employee Karen denotes what we will assume to be Karen's current salary.

[1] The equivalent term used in *Smalltalk* is **instance variable.**

Center for Digital Economy Research
Stern School of Business
Working Paper IS-88-99

The other attributes in this object are interpreted similarly.

However, in the context of a historical database, if, for example, Karen has been employed and therefore relevant to the world being modeled by the database since "*January 1, 1980*", then it becomes reasonable to query the database about her salary on any day during her employment. Thus, unless some specific instance in time is understood or otherwise inferred, the denotation of "Karen's Salary" can be viewed as being not a single value, but all of her salaries during the time she was employed, that is, her salary history.

In order to accommodate this view we define a **historical object** as an object whose attributes denote functional values. These functions, which, again for consistency, are themselves objects, define a correspondence between objects of type *time* to objects of the appropriate type, for example objects of type *salary*, *department*, or *name*.

Often an entity or relationship is relevant to a database for only some restricted period of time. The period of time for which a historical object models an entity or relationship is called the **lifespan** of that object. The domain of the function denoted by an attribute of an object is restricted to exactly those times in the lifespan of that object. (If modifications to a database scheme are to be allowed then it may be desirable to change – for instance, to extend – the definition of an object lifespan. However, this topic is beyond the scope of this paper. [CC87] presents an extended relational model with tuple and schema lifespans.)

2.2. Object Identity

A major issue when dealing with objects is the issue of object identity – how are we to distinguish different objects. Indeed this issue is not a new one. In earlier data models the notion of a *key* was used to so distinguish different records or tuples. In logic the issue of the "essence" of something addresses the same idea – what property of an object is essential to its being itself, so that anything with that property must be that thing, and anything without that property cannot be that thing. (The issues of object identity, existence, cross-world identification, and object counterparts have a long history in the philosophical literature, e.g. [Lew68,Mon74b,Kri80].)

Chen and Warren [CW88] are specifically concerned with this issue; our approach differs considerably from theirs by the introduction of the notion of an *essence*. We believe that each object must have an *essence*, which is a time-invariant identifier shared by no other object. One refers to an object by means of its essence. If two essences are equal then by definition they refer to the same object. Component properties of objects – such as an attribute *SALARY*– have as their value functions from time objects (referred to by their essence) to some other type of object (referred to by its essence). They are, in the terminology of logic, *intensions*. ([Mon74a,Gal75]). The essence of these functions, for example a *SALARY* function, are in general not directly known to the user – instead they would be referred to indirectly as the value of some property of a more essential object, say *Karen*, whose essence the user would know.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-88-99

Now the issue addressed by Chen and Warren is how to tell whether two partially-specified intensions are the same. We would contend that two intensions are the same if and only if they are the value of some nonessential property of the same essential object; otherwise, even though extensionally they may be equal (i.e., have the same value for every time) they are not equal. However if these two intensions are created as different by the user, they would have two different *essences* and thus, though as functions they might be extensionally equal, as objects they are not the same. Thus John's *SALARY* is *never* Mary's *SALARY*, though they might always be earning the same money.

These issues motivate the following definitions:

1. It is essential that a system be able to maintain the integrity of object identity. Since user-defined keys are notoriously *not* time-invariant, for example, even people's social security numbers have had to be changed, our system will need to create and manage time-invariant object identifiers. We call such an identifier an **essence**.

2. Equally relevant to the management of objects over time is the maintenance of *when* that object existed. We call this information the **lifespan** of the object. Since an object may have temporally disjoint periods of existence, a lifespan consists of a set of disjoint intervals of time; such an interval is called an **incarnation**.

3. If E is an essence, we denote the lifespan of E as $E.l$.

4. The essence of each primitive object is simply its name, and the lifespan of each primitive object is $\{[-\infty, +\infty]\}$.

2.3. Object Structure

Various proposals for representing data as objects have incorporated different constructors for defining complex object types (or *classes*) from the primitive types. Common examples of these constructors are *record* construction and *set* (or collection) construction. Without examining any particular such constructor, let us assume that some complex object type O is defined in terms of n simpler object types. I.e.,

$$O = O_1 + O_2 + \ldots + O_n$$

where the symbol "+" is to be interpreted generically as any such constructor. When instantiated, a complex object has its own distinct essence. Any of the operations on objects can of course be applied to complex objects.

Note, however, that when a new object E of type O is created, consisting of the n component objects $E_1, E_2, \ldots, E_n$ of types $O_1, O_2, \ldots, O_n$, respectively, the constraint

$$\forall i[E.l \subseteq E_i.l]$$

must always be satisfied; an object can exist only while its components exist.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-88-99

Relationships, too, are complex objects. However objects can certainly exist in time independently of the relationships they form. Thus the following restriction should be imposed upon relationships:

$$E.l \subseteq E_1.l \cap E_2.l \cap \ldots \cap E_n.l$$

Relationships pose another interesting situation where it is perhaps best to leave the choice of representation up to the user. Consider, for example, the two marriages of Elizabeth Taylor and Richard Burton. Are these two distinct objects (with, therefore, two essences) or are they two incarnations of the same marriage? Either representation should be possible.

Note that as a consequence of this, the following holds:

$$[E_\alpha = E_1 + E_2 + \ldots + E_n] \wedge [E_\beta = E_1 + E_2 + \ldots + E_n] \not\rightarrow [E_\alpha = E_\beta]$$

i.e., not only can there be different relationships defined in terms of the same underlying objects, but there can even be different instances of the same relationship between the same objects.

3. Manipulating Historical Objects

The entity or relationship modeled by an object is assumed to be relevant to the database during certain periods of time. These time periods are reflected in the incarnations of the object's lifespan. Each incarnation begins with a time when the object becomes newly "existing" from the perspective of the application, and terminates with the execution of an operation that "kills" that existence.

An object is brought into being with the operation[2] **CREATE**. The method used to define the **CREATE** operation, if so written, could also define the initial values of the attributes of the instantiated object. **CREATE**(X, B) returns a new essence E which uniquely identifies a new object of type X; the lifespan $E.l$ is $\{[B, \mathbf{now})\}$. Moreover, if X is a compound object type, say $X = X_1 + X_2 + \ldots + X_n$ then n essences $E_1 + E_2 + \ldots + E_n$ are also generated with the same lifespan as $E.l$.

The execution of a **KILL** operation, implemented with the appropriate object method, terminates the most recently opened incarnation in the referenced object's lifespan. **KILL(E,D)** finds the object with essence E and, assuming it has lifespan $\{[B_1, D_1], [B_2, D_2], \ldots, [B_n, \mathbf{now})\}$, updates its lifespan to $\{[B_1, D_1], [B_2, D_2], \ldots, [B_n, D]\}$. If E is a compound object of type X and $X = X_1 + X_2 + \ldots + X_n$ then the lifespans of the n essences $E_1 + E_2 + \ldots + E_n$ are also updated.

After an object has been **KILL**ed (but not removed) from a database it may necessary to **REINCARNATE** it. For example, an employee may subsequently be rehired. The affect of a reincarnation of an object is to extend its lifespan by beginning a new

[2]The term used in Smalltalk for an operation is **message**.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-88-99

incarnation. (This incarnation will be terminated by the next subsequent **KILL** operation that is executed on the object.) Only objects that have been previously **CREATE**d can be **REINCARNATE**d. **REINCARNATE**(E, B) finds the object with essence E and, assuming it has lifespan $\{[B_1, D_1], [B_2, D_2], \ldots, [B_n, D]\}$, updates its lifespan to $\{[B_1, D_1], [B_2, D_2], \ldots, [B_n, D_n], [B, \textbf{now})\}$. If E is a compound object of type X and $X = X_1 + X_2 + \ldots + X_n$ then the lifespans of the n essences $E_1 + E_2 + \ldots + E_n$ are also updated.

It may sometimes be necessary to merge or **IDENTIFY** two descriptions into one because two supposedly distinct objects are now realized to be in fact the same object. **IDENTIFY**(E_1, E_2) finds the objects with essences E_1 and E_2 and, assuming that the object descriptions are "compatible"[3], creates a new object with the merged descriptions of E_1 and E_2 and gives it the essence E_1; the essence E_2 is no longer useable except as an alias for E_1;

It may be necessary, though perhaps forbidden in certain highly sensitive applications, to delete or **DESTROY** permanently any trace of an object from the system. **DESTROY**(E) finds the object with essence E and, removes it from the system. The essence E is thereafter and forever unusable. Once **DESTROY**ed an object cannot be **REINCARNATE**d.

Access to the attributes of historical objects is achieved in the conventional way, through the specification and invocation of the appropriate method. However, because of the structure of historical objects the value of an accessed attribute may have to be manipulated further.

Assume that the expression $\mathcal{O}$ **A** is used to invoke the method that retrieves the value denoted by attribute A of object $\mathcal{O}$. When the method invoked is that of an historical object then the object that is accessed is a function. For example, if **KAREN** is the name of the historical object modeling the employee Karen, then **KAREN salary** returns the function that represents Karen's salary history, and thus associates a salary with each time in the lifespan of **Karen**.

Since the functions denoted by the attributes of historical objects are themselves objects they can, and do, have methods associated with them. In particular, we assume that each such function object $\mathcal{F}$ includes a method that when invoked by the expression $\mathcal{F}$ **at:** *time* returns the value that $\mathcal{F}$ associates with the time denoted by *time*. The expression **KAREN salary at:** "March 1, 1981" returns the value of Karen's salary on the specified date.

Similarly attribute updates are accomplished using an expression of the form $\mathcal{O}$ **A at:** *time* **put:** *value*. This expression updates object $\mathcal{O}$ by extending the function denoted by attribute **A** so that it associates with *time* the value specified by *value*.

[3]What this means is an issue in its own right; see [CC87] for further details on this issue.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-88-99

4. Conclusion

We believe that it is inconceivable to successfully develop an object-oriented model of data without providing for the modelling of the temporal dimension of objects. This paper represents a modest beginning toward amalgamating these two lines of research, object-orientation and historical data modelling, thereby providing users with the ability to model *objects in time.*

References

[BADW82] A. Bolour, T. L. Anderson, L. J. Deketser, and H. K. T Wong. The role of time in information processing: a survey. *ACM SIGMOD Record*, 12(3):28–48, April 1982.

[CC87] J. Clifford and A. Croker. The historical relational data model (hrdm) and algebra based on lifespans. In *Proc. Third International Conference on Data Engineering*, pages 528–537, IEEE, Los Angeles, February 1987.

[CW83] J. Clifford and D.S. Warren. Formal semantics for time in databases. *ACM Trans. on Database Systems*, 6(2):214–254, June 1983.

[CW88] W. Chen and D.S. Warren. *Objects as Intensions.* Technical Report, Dept. of Computer Science, SUNY at Stony Brook, 1988.

[Dit86] *Proc. International Workshop on Object-Oriented Databases*, Pacific Grove, CA, September 1986.

[Gal75] D. Gallin. *Intensional and Higher-Order Modal Logic.* North-Holland, Amsterdam, 1975.

[GR83] A. Goldberg and D. Robson. *Smalltalk-80: The Language and its Implementation.* Addison-Wesley, Reading, MA, 1983.

[Kri80] S. Kripke. *Naming and Necessity.* Harvard University Press, Cambridge, MA, 1980.

[Lew68] D. Lewis. Counterpart theory and quantified modal logic. *The Journal of Philosophy*, 65(5):113–126, March 1968.

[McK86] E. McKenzie. Bibliography: temporal databases. *ACM SIGMOD Record*, 15(4):40–52, December 1986.

[Mon74a] R. Montague. *Formal Philosophy: Selected Papers of Richard Montague.* Yale University Press, New Haven, 1974.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-88-99

[Mon74b] R. Montague. *On the Nature of Certain Philosophical Entities*, pages 148–187. Yale University Press, New Haven, 1974.

[MSOP86] D. Maier, J. Stein, A. Otis, and A. Purdy. Development of an object-oriented dbms. In *Proc. of the Conference on Object-Oriented Programming Systems, Languages and Applications*, pages 472–482, September-October 1986.

[TAI87] AFCET. *Temporal Aspects in Information Systems*, May 1987.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-88-99