Managing Time in Object-Oriented Databases

Lincoln M. Oliveira
Claudia Bauzer Medeiros

Relatório Técnico DCC–14/93

Julho de 1993

Managing Time in Object-Oriented Databases

Lincoln M. Oliveira Claudia Bauzer Medeiros*

Abstract

This paper presents a new approach for modelling and querying temporal object oriented databases. The model presented in this paper extends previous work in the area, by supporting the evolution of all object properties through time (inheritance, composition and behavior), and allowing temporal schema evolution. A prototype of this model is being implemented as a time-managing layer on top of the O2 object-oriented database system. In order to manipulate our temporal objects, we have extended the O2 query language with temporal constructs, which we also discuss in the paper.

1 Introduction

The recording of temporal evolution of data allows users to better model the dynamics of the real world. For this reason, there has been an extensive research effort in the area of temporal databases. Many different models have been proposed, mainly for relational systems (see, for instance, the bibliography in [Soo91]). There still remain several problems to be solved (as witness the comments about open issues in [Sno90, JCG+92]).

Research in temporal object-oriented systems is recent, and has, for the most part, been conducted in the last two years. Existing models do not fully consider all issues involved in introducing time into the object

*Research partially financed by grants FAPESP 91/2117-1 and CNPq 500869/91-0

world – such as schema or behavior evolution. Furthermore, most models consider only one time dimension

The model presented in this paper – TOD – is an attempt to solve these issues. Like most models, TOD supports the temporal dimension by extending a non-temporal model (in this case, object-oriented) with new attributes and functions. Unlike previous proposals, it allows several degrees of freedom in the temporal evolution of objects, by supporting data, behavior and schema modifications in time, for two independent time dimensions. As will be seen, though this enables the representation of a broad spectrum of real life situations, it introduces many problems for the correct maintenance and querying of temporal data.

The main contributions presented are:

- Support of the object-oriented framework

  TOD is fully object-oriented, which means that it takes composition, inheritance and behavior into account when modelling temporal evolution. Other models consider only object composition, and inheritance treatment is limited to special cases.

  Temporal composition involves the process of building temporal objects from other (temporal) objects with varying lifespans and temporal characteristics. It considers issues such as determining the validity of a composite object whose components are valid in different time periods (e.g., when modelling office documents).

  Inheritance issues concern defining appropriate rules for determining the temporal behavior of objects of a class when this behavior is inherited from other classes.

  Temporal modelling of behavior implies supporting the evolution of methods along time. Thus, an object may change in time not because its data values change, but because its behavior is modified.

- Temporal querying

  Temporal queries in TOD can return either time values or database states, and can navigate through and combine database states in

time. Query constructs also extend previous work in the sense that they may return sets of database states over time (instead of one state at a given point in time). Thus, TOD queries can be used as a means for creating new temporal databases.

- Temporal schema evolution

  Unlike previous research, we consider not only the evolution of data (extension) but also of the schema. Thus, the maintenance of previous database states involves keeping track of the associated schema. Schema evolution is subject to *temporalization* rules, which are integrity constraints that determine valid schema modifications given temporal inheritance and composition constraints.

TOD is being implemented as a layer on top of the O2 object-oriented system. This includes providing new builtin methods and functions for defining and querying temporal objects. The implementation is still under way, and is not discussed in this paper. We restrict ourselves to describing the model and the query language constructs.

The rest of this paper is organized as follows: section 2 presents the terminology adopted in this paper, and discusses recent results in the area; section 3 presents TOD , comparing it to other object-oriented temporal data models; section 4 discusses the query language; finally, section 5 presents the conclusions and comments on the present state of the implementation effort.

2 Temporal databases - an overview

This section presents an overview of recent research on temporal databases. We chose to analyze the aspects of temporal modelling which we consider important to the understanding of the problem, and to allow presenting our approach.

An attempt to establish a common terminology basis for temporal research in relational systems appeared in [JCG+92]. This paper adopts

the same terminology, adapting it, when necessary, to the object framework.

2.1 Data Model

Most proposals on modelling temporal databases are based on the relational model. Relational temporal models are usually obtained by adding special (time-related) attributes to a relation. These attributes are subject to a specific semantic interpretation. There are few non-relational proposals: two that are based on the entity-relationship model [KL83, AQ86], one that does not depend on a particular data model [SS87, SS88], and some recent proposals that extend the object-oriented data model [SC91, KS92, WD92, DW92, PM92].

[SC91]'s proposal does not consider all the implications of introducing temporal issues into the OO paradigm, like inheritance. [WD92]'s model, though more comprehensive as far as object-orientation is concerned, is based on a functional paradigm, and leaves to the database designer the definition of great part of the temporal data model, without providing specific temporal primitives. [KS92] base their proposal on the molecule object model (MAD), and do not consider method or schema evolution. [PM92] treat time as an added attribute in an object model that was designed for information systems support, and which is therefore not general.

2.2 Temporal Dimensions

Most authors agree on the need to support at least one of two types of time – *valid time* and *transaction time*. *Transaction time* is system-generated, and records when a fact was actually stored in the database, by means of timestamp generators. *Valid time* is user-provided, and represents the actual time when a fact occurs in the real world. It allows recording information not only about the past, but also about the future. Thus, whereas transaction time increases monotonically, valid time can vary arbitrarily. Another type of time, *user-defined time*, corresponds to a data type whose domain is defined on time values, but does not

have any special meaning to the DBMS. Valid time and transaction time can be seen as two orthogonal dimensions of time, over which the data evolves independently.

Most proposals consider only one kind of time (usually valid time). Only [SA85, MS90], [ABN87], [KRS90] and [Sar90b] provide support to two temporal dimensions (valid and transaction time). [Ari86] and [Gad88] furthermore present proposals to generalize the concept to an arbitrary number of temporal dimensions. Our model supports transaction and valid time. The only object-based model that considers more than one dimension is [WD92].

Some authors discuss the need for more than one temporal dimension, but do not explain how they provide support to them or describe how to take advantage of them. Examples are [LDE+84], [MNA87], [SC91] and [KS92].

2.3 Value Evolution over Time

Database entities (attributes, tuples, components, objects) can vary over time in several ways. The classification presented by [SS88] summarizes the proposals in the literature. Entities that vary following a *step-wise constant* keep their values unchanged during the period between two updates. In case of *discrete* value modification, the value assigned to an entity is valid for only one instant and there is no connection among the entity values in distinct instants. *Continuous* and *user-defined* evolution is modeled by interpolating stored values using a curve fitting function provided, respectively, by the DBMS or by the user.

The step-wise constant variation of values is supported by default in all models analyzed, except [KL83]. In this model the user must define a set of functions to determine values not stored. [SA86], [Sar90a] and [LJ88] support furthermore discrete value evolution. [LJ88] uses an operator to simulate other types of interpolation. TOD , like [SS87, SS88] and [KRS90], provides support to the four types of value modification.

2.4 Algebraic Operators and Query Language

Most models introduce either an algebra or a query language to manipulate temporal constructs. Only three models ([LDE+84], [KL83] and [ABN87]) propose neither. Algebras are usually extensions to the relational algebra and differ in the set of temporal operators and in the manner in which they treat the traditional relational operators.

Most proposals provide an operator to perform *valid time slicing*, i.e., an operator that returns the state of a database entity at some specific valid time interval. Examples are [CC88]'s σ_{WHEN}, $\tau_{@L}$ and $\tau_{@A}$ operators, [Sar90b]'s FROMTIME and TOTIME, [Gad88]'s DURING, [Ari86]'s AT, WHILE, DURING, BEFORE and AFTER and [SC91]'s WHEN clauses. TOD provides the operator (VSLICE) for this purpose.

The most common operation concerning transaction time is the *rollback* operation, which returns a database state that was current at some point in time. Snodgrass' operator ρ [MS90] and clause AS-OF [Sno87], and [Ari86]'s AS-OF clauses are examples of that operation. In our model we generalize this operation using operator TSLICE to select data over a set of time intervals instead of at a single instant.

A third kind of operation yields a set of time values according to some conditions on the data. It is executed either directly enabling access to the timestamp values or by defining specific operators, like [CC88]'s Ω operator or [Gad88]'s TDOM clause. Our model provides two operators (TWHEN/VWHEN) for this purpose.

In a few models the temporal dimension is manipulated by operators analogous to the nested relational operators. [Tan86]'s PACK/UNPACK, [Sar90b]'s EXPAND/COALESCE and [LJ88] FOLD/UNFOLD operators are examples of this.

Temporal object models provide similar operations, and are influenced by the underlying data model. [WD92]'s language differs from all of the above by allowing temporal operations without extending the query language. Their functional model supports storing and querying of time by adding functions to the database. Thus, there is no difference between temporal and nontemporal query syntax.

2.5 Other Aspects

Besides the issues discussed, there are other relevant aspects in temporal modelling. The level of time versioning, i.e., the choice of associating time to entities or entity components (in the relational model, to attributes or tuples) and the choice of the timestamp characteristics — time points, time intervals or sets of time intervals — are extensively discussed. TOD associates time to object components (which resembles attribute stamping); time can be stored as points, intervals or sets of time intervals.

Maintaining past states of the database brings up the issue of schema evolution. In spite of the importance of this topic, it is totally ignored in most the proposals. Only few models emphasize this topic (e.g. [MS90] and [MNA87]). In our model we support schema evolution by associating the schema to transaction time (treating the problem as that of maintaining schema versions over time).

Implementation is another issue to be considered. Several models are supported by an implementation, usually modifying an existent DBMS. Those implementations only aim at demonstrating that the models are implementable, without worrying about efficiency. Our model is being implemented as a layer on top of the O_2 system.

2.6 Synoptic Tables

In the annex, we summarize the topics analyzed. The topic *Schema Evolution* only denotes if the topic was mentioned, which does not mean the model treats this issue in a satisfactory way.

3 TOD - Temporal Object oriented Data model

There is no standard definition for an object-oriented model. TOD relies on [Bee89]'s class-based framework. An object is an instance of a class. It is characterized by its state (contents) and behavior (methods), and is subject to inheritance and composition properties. Objects can

be composed into more complex objects using constructors. Users may also define functions, which operate on values. The database schema is defined by its composition and inheritance graphs, as well as the applicable methods. TOD extends [Bee89]'s object model with valid and transaction time dimensions. From now on, these times will be denoted **TT** and **VT**. The main characteristics of TOD that make it more general than other object oriented temporal models are:

- Temporal categories – simultaneous maintenance of different types of temporal classes;
- The TIME hierarchy – representation and maintenance of the time dimension by means of objects of a special class hierarchy;
- Schema evolution – definition of a set of integrity rules that define valid schema transitions in time.

3.1 Temporal categories

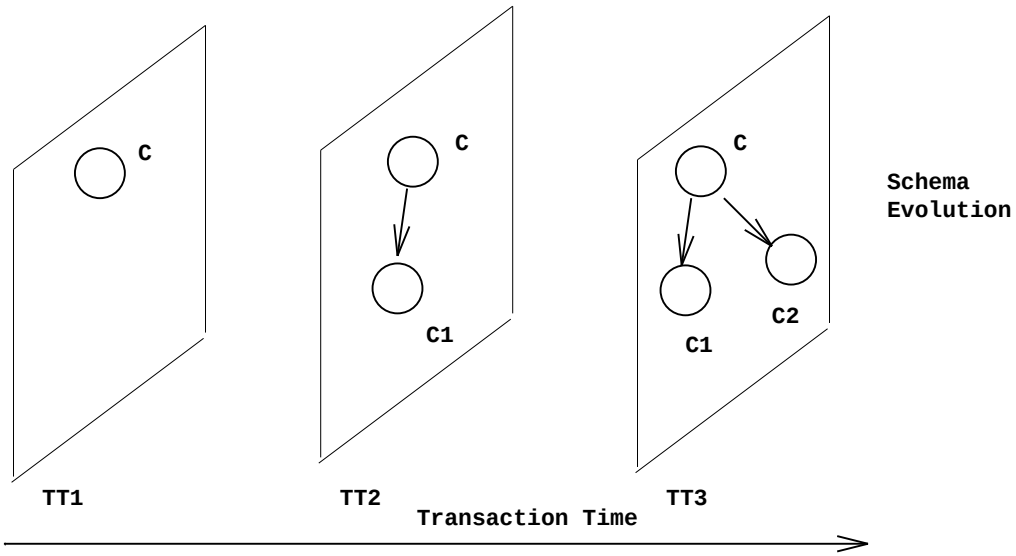
Similar to the classification of temporal relations in [JCG+92], in TOD there are four categories of temporal classes according to the type of time supported:

- **Snapshot classes** – traditional non-temporal classes.
- **Valid time classes** – classes that support only VT . They allow describing the history of object states as known in the present.
- **Transaction time classes** – classes whose objects support only TT . They maintain past (recorded) database states, but not validity information.
- **Bitemporal classes** – classes that support both VT and TT .

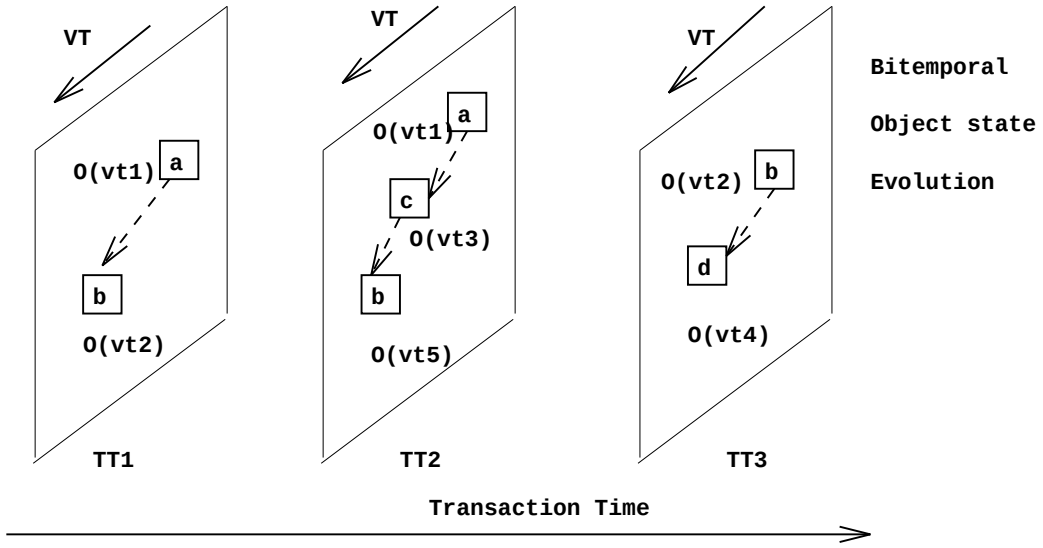
Paraphrasing [JS92]'s definition of a temporal relation, we define a TOD *temporal database* to be a "sequence of historical database states indexed by transaction time". Thus, a database can be visualized as a sequence

of (logical) temporal slices, where each slice is the database's state for a given TT , and where the real world historical evolution of an object is given by VT . The most recent slice in the TT sequence corresponds to the present time, and is indexed by a special TT value – NOW . The past cannot be updated – i.e., updates can only be applied to the NOW slice. With this definition, the schema of a temporal database needs no longer to be fixed, since for each TT slice there may exist a different database schema.

Figures 1 and 2 show two examples of the slice concept. In Figure 1, the schema evolution of a class hierarchy is displayed. Initially, at TT = TT1, there was only class C. At TT = TT2, the $C1$ subclass of C was created. Finally, $C2$ was defined. Figure 2 shows the TT states for a



bitemporal object O. At TT = TT1, the object has two values: a (for VT = $vt1$) and b (for VT = $vt2$). In the second slice, the knowledge about the history of O has changed, and now there are three values. In the third slice, this knowledge has changed again. Section 3.4 contains a discussion about the values of O for this picture. Unlike previous research on temporal object oriented databases, we allow a database to simul-



taneously have the four categories of classes. This imposes restrictions on the valid temporal state transitions (from one slice to its temporal successor). A temporal database without TT support will have only one slice (NOW) with snapshot and valid time classes. A database without VT support will not allow valid time or bitemporal classes.

Suppose that at TT = NOW there are four classes in the database: C_S (snapshot), C_{TV} (valid time), C_{TT} (transaction time) and C_{BT} (bitemporal). Suppose furthermore that all classes have extensions. If the extensions of C_{TT} or C_{BT} are updated, this will imply the creation of a new TT slice[1]. This new slice corresponds to a new NOW state. Temporal state transitions leading to a new slice obey the following rules:

1. all classes and their extensions are (logically) copied to the new NOW slice;

2. updates are applied to this NOW slice (thus maintaining past information unchanged in the previous slice);

[1] In practice, the user may want to define the granularity of TT . Thus, a new TT slice is not necessarily created at each update of a class that supports TT .

3. all information about C_S and C_{TV} is destroyed in the previous slice. It will only preserve the classes that support TT – C_{TT} and C_{BT}. The other two classes exist by definition only in the present. Thus, a rollback operation will not recover all class extensions as stored in a given TT , only those that support this dimension.

   We note that there may exist other interpretations of time semantics – e.g., that snapshot and valid time classes are immune to TT variation, and thus should exist in all slices. We chose however to adopt the time semantics used by all other other temporal models that support both VT and TT .

We stress that this copying of slices is a *logical* operation, and that it does not actually require copying the whole database. Our "slicing" model is based on [CJ90]'s object version management model, where we substitute TT for the version identificator. The use of a version management framework allows TOD to handle versions as well as time. Each slice corresponds to a version of the database.

3.2 Representing the time dimension - the TIME hierarchy

The representation of time is a central issue in all temporal models. Most database researchers represent time as a set of consecutive, equidistant time points. The distance between two such points is called the *minimum time granularity*. A *time interval* (t1..t2) is a set of time points covering the period starting in t1 and ending in t2. We use an approach similar to [WD92]'s to temporally extend an object-oriented model: we maintain a system-supported set of classes, which model time dimensions.

TOD thus distinguishes between temporal objects and the representation of time. This representation relies on an atomic type called **Time_Point**, and a hierarchy of classes whose root is a class called **TIME**, and which is defined as a set of **Time_Point**. Different subclasses of the hierarchy represent different types of time evolution, thus allowing several time representations inside a single database.

Database objects in TOD are ordinary objects, extended with components of the TIME hierarchy. This extension is achieved by means of

the *tuple* constructor, present in all object-oriented models. Database objects may be of three kinds:

- time-invariant objects - do not have a temporal component
- time elements - complex objects of some class of the TIME hierarchy
- time-varying objects - that have at least one component from the TIME hierarchy

Since complex objects can be recursively built from other objects using constructors, a complex time-varying object may have time-varying components as well. This is equivalent to combining attribute and tuple temporal timestamping in the relational model: the object has temporal properties, which describe it as a whole; and each of its components may in turn have specific temporal properties.

The temporal dimension starts at the atomic level, transforming atomic values into time-varying objects. For instance, an atomic attribute NAME of type *string* may be temporally represented by a bitemporal object TNAME:

TNAME: tuple (Value: *string*; VT : *time_h*; TT : *time_h*)

A temporal object containing sets of TNAMEs may be represented, for instance as
O1: tuple (Content: set(TNAME); VT : *time_h*; TT : *time_h*) (bitemporal)
O2: tuple (Content: set(TNAME); VT : *time_h*) (valid time object)
or even O3: set(TNAME) (snapshot object, whose components are bitemporal)

(We use the notation *time_h* to denote that members of the TIME hierarchy. Each one, however, may belong to a different class in this hierarchy.) We point out one of the characteristics of TOD : O2 is a valid time object, but it has bitemporal components (i.e., O2 exists only NOW , but its components may exist in other slices). Queries posed on O2 will ignore the TT dimension of its components; queries posed directly to

these components may consider this dimension. This approach to query processing is one of the points in which TOD differs from other models.

This characterizaton allows a given object to be composed of other objects with different temporal characteristics. Classes of the TIME hierarchy may vary according to time *granularity, type* and *unit*.

The *type* of variation of an object in time is given by four functions (see [SS87]) – step, discrete, continuous and user-defined. The *granularity* can be of three types: points in time, intervals and sets of disjoint intervals. The *unit* indicates how time varies (e.g, second, day, month).

TIME hierarchy classes are subject to methods that perform temporal operations: *set* operations (e.g., intersection, union, difference), *comparison* operations (e.g., equality, containment, inequality) and *order* operations (e.g., before, after, first, last). These methods are activated on queries and on schema updates, to maintain (temporal) schema integrity (see 3.3). There are, moreover, functions that perform temporal conversions to allow operations among members of different classes (e.g., intersection of a TIME object which is an *interval* of *year* units with a TIME object which is a *set of intervals* of *month* units).

Since TT is determined by the DBMS, its properties should remain fixed for a given database system. In general, it is a step-wise function with a point or interval granularity. VT depends on the user; thus, several types of VT can coexist in a single database. This, again, is similar to [WD92]'s model, except that the latter do not consider automatically providing builtin temporal data types.

3.3 Schema updates

The schema of a temporal database can be restructured during the database's lifetime. This issue is ignored by most temporal data models. TOD supports these changes by associating to each schema the TT for which it existed, similar to a version mechanism. We discarded the possibility of associating VT to a schema, since that would imply allowing different schemas inside a database slice, with varying validity spans.

We only discuss updates in the context of empty schema elements

(i.e., classes without extensions). Schema changes of classes with extensions imply managing the oid. A discussion of this problem involves implementation assumptions about a database system. For details, the reader is referred to [Oli93].

3.3.1 Ordering of temporal categories

A database schema contains inheritance and composition links. This means that the creation of new classes cannot be done arbitrarily, since there are implicit constraints that must be obeyed.

The constraints that define a temporally consistent state in TOD are based on a partial order of the temporal categories. This order states that a class from a given category must accept at least all temporal operators applicable to its precedessor classes in the order. Let $\preceq$ denote this partial order. If $C1$ and $C2$ are classes, then $C1 \preceq C2$ means that $C2$ accepts at least all operators accepted by $C1$. In general,

$$\text{snapshot class} \preceq \text{transaction time class} \preceq \text{bitemporal class}$$
$$\text{snapshot class} \preceq \text{valid time class} \preceq \text{bitemporal class}$$

This order shows, for instance, that a bitemporal class allows any temporal operation that is allowed by the other categories. This partial order imposes constraints on inheritance and composition links. For instance, a transaction time class can be a subclass of a snapshot class, but not the opposite. A snapshot class can be composed of bitemporal classes, but not otherwise. Since the temporal dimension corresponds to adding components in a tuple, changing the category just implies adding or eliminating TT /VT from a class type.

3.3.2 Update operations

Temporalization is the process by means of which a database schema is made temporally consistent by forcing changes in existing classes when a schema update is performed. Schema updates can be classified in the following groups

- class creation or removal
- modification of the composition graph (deletion or addition of component, or change of component temporal characteristics)
- modification of the inheritance graph (addition or deletion of edges)
- changes in the temporal characteristics of a class
- behavior changes (methods)

Class creation or removal

Class creation is subject to the partial order constraints (e.g., all subclasses of a bitemporal class must be bitemporal; subclasses of a valid time class may be bitemporal or valid time classes). This extends other time inheritance models, which assume a single temporal category for all members of a hierarchy.

Class removal only eliminates the class description from the NOW slice. If it is a bitemporal or transaction time class, its description remains in the past database slices.

Changes in the composition graph

The only change in the composition graph that may alter temporal characteristics is the addition of a new component (class C_1) to a class C. If this component is an already defined class C_1, this may force alterations in C_1 (and recursively to its component classes) to obey the partial order constraints: the temporal category of the C_1 component must be at least the same as the category of composite class C. For instance, if a snapshot component C_S is added to a transaction time class, then the C_S disappears and a new transaction time class is created, with the same components as C_S, and which will acquire the TT dimension.

Notice that the temporalization process may have to iterate between hierarchy and composition compatibilization until a consistent temporal schema is achieved. For instance, if C_S had any subclasses, they

might have to change their temporal properties after the change in their ancestor.

Changes in the inheritance graph

Two cases must be considered: the addition of an edge and the elimination of an edge. Edge elimination does not require temporal adjustment. The addition of an edge requires compatibilization only if the new inheritance edge links existing classes. In this case, the subclass must be made temporally compatible with the superclass, following the partial order.

Changes in behavior

We consider methods to be part of the database schema. Methods may suffer changes in code. Furthermore, the objects to which they apply may also change in time. Finally, methods may become invalid if the class to which they are attached changes its temporal characteristics.

A method is allowed to change temporally by adding TT and/or VT to its signature. The method can only be applied to objects whose TT and VT values are compatible with its signature values. This compatibility is defined as follows: the object's corresponding time dimensions (VT or TT components) must have an intersection with the method's VT and TT components (i.e., return a positive answer to one of the operators **overlaps, intersects, contains**).

3.4 Value determination

Until now we have not considered the question of evolution of data values in time. The semantics of VT demands definition of this evolution, in order to allow composition of objects with different TIME properties. Again, this is an extension not considered by other models. This only applies to valid time (since we assume that all TT objects in a database belong to the same TIME class).

Consider again object O in figure 2, for slice TT = TT1. Its value is a as of VT $=vt1$, and b as-of VT $=vt2$. Its value between those two times depends on the temporal characteristics of the class to which O belongs. If it varies according to a discrete function, then O's value between $vt1$ and $vt2$ is null. If according to a continuous or user-defined function, the corresponding interpolation function will be used. If the variation is given by a step-wise function, then the value a is valid from vt1 to vt2.

Inheritance and composition links do not affect the temporal properties of a value. In the case of composition, the composite object's temporal properties will determine when its components belong together. For instance, consider the ascending time point order (vt1, vt2, vt3, vt4) and objects O (valid from vt1 to vt3), $O1$ (valid from vt2 to vt4) and $O2$ (valid from vt1 to vt2). Suppose O is composed of $O1$ and $O2$ (i.e., $O =< O1, O2 >$. Then, the values of O during its validity period are: $O =<$ null, $O2 >$ (as-of $vt1$); $O =< O1$, $O2 >$ (as-of $vt2$); $O =< O1$, null $>$ (as-of $vt3$)

4 The query language

Our temporal query language syntax is based on the O2 database query language – O2Query. Our choice of language was influenced by the fact that we already use the O2 system, and are well acquainted with its querying facilities.

4.1 Query constructs in TOD

Temporal queries in TOD may return

- non-temporal elements (e.g., a string),
- time-invariant values (e.g., results of functions that do not vary in time),
- temporal elements (time values, and time-varying objects, classes or databases).

We will only discuss the third type of query, since the others do not present any novelty. The most general form of answer to a temporal query is a (new) temporal database, which restricts the original database to some set of VT and/or TT temporal intervals. This extends other temporal query languages, which do not allow the creation of databases.

Queries must take schema modification into account. Thus, when a query navigates through different database slices, it must consider the schema for every slice[2].

As in several other proposals, the statement of a query depends on the temporal nature of the database elements being accessed. If the database contains only snapshot classes it is a standard nontemporal database, and queries on snapshot classes are standard database queries. We thus ensure one of the properties desired of temporal systems - that temporal queries be reduced to their non-temporal equivalents if the database is not temporal.

By the same token, it makes no sense to look for VT in a transaction-time class (or, analogously, to apply a query involving TT to a valid-time class). Finally, bitemporal classes support queries in VT and/or TT .

Queries on temporal elements use special operators, which can be either *comparators* (return a boolean value) or *constructors* (return a new temporal element). These operators are methods applied to the TIME components of objects of temporal elements. The complete set of operators is described in [Oli93].

Temporal queries that return temporal elements can be of two kinds:

- queries that return objects of the TIME hierarchy (e.g., **intersection ([1990 .. 1992], [1991 .. 1993])** is a temporal query that returns the interval [1991 .. 1992])
- queries that return time-varying objects (e.g., displaying a temporal class extent)

[2] This can be achieved by a mechanism similar to [CJ90]'s version support.

4.2 Queries returning objects of the TIME hierarchy

These are the queries that answer *when* questions. *VWHEN* determines when a given fact was valid in the real world; *TWHEN* returns the set of transaction times in which the fact was stored.

A "when" query over TT is stated as

```
TWHEN (non-temporal predicate on database objects)
FROM { source classes}
VALID (temporal predicate)
```

This query returns the set of TT for which the predicate was valid during the VT specified in the VALID clause.

Its result is obtained through the following three steps

1. obtain the set of TIME objects that satisfy the VALID clause
2. restrict the database to the set of temporal elements that satisfy the predicate
3. return the transaction times at which the elements found in (2) were valid at the times determined in (1).

Analogously, a "when" VT query is obtained as:

```
VWHEN (non-temporal predicate on database objects)
FROM (source classes)
INDB (temporal predicate)
```

This query returns the set of VT for which the predicate is valid in the real world at all the TT values specified in the INDB clause. The result can be described in three steps, similar to the previous VWHEN query. A temporal predicate may be a condition on TIME objects (e.g., VALID (NOW)) or on time-varying objects. For instance,

```
TWHEN (predicate X)
FROM  source classes
VALID  (during (Y.age = 25) )
```

requires in fact two temporal queries of the database: the first to determine what was the time interval for which the age of Y was 25; the second to determine the TT values in which X was true at the same time periods.

4.3 Queries returning temporal objects

These queries return the state(s) of a database that satisfy certain conditions during a VT or TT period. Queries that return objects are called *timeslice* queries. They can be seen as operations that slice the database at specific TT and/or VT intervals.

In the first case, we say that one is applying a *valid timeslice – VSLICE* operation; in the second case, a *transaction timeslice – TSLICE* operation. Both types of slicing must be applied simultaneously in the more general type of query. Unlike slices described by most researchers, (which contain one state of each object), our temporal slices correspond to a set of temporal states of a database.

TSLICE operators are applicable to any class that contains TT (i.e., transaction and bitemporal classes). For each TT in the argument, they return the state of the objects that satisfy the query and which existed at the time. This is similar to [MS90] rollback operator (in the sense that it returns database TT slices given a transaction time). However, unlike that operator, it may return a set of states whenever the time argument contains a set or interval of TT values. The result of a TSLICE operator can therefore be a temporal database covering a set of TT values.

VSLICE operators are applicable to classes that contain VT (i.e., valid time and bitemporal classes). They operate in a way similar to transaction timeslice operators, with the difference that the time argument is based on VT instead of TT . Again, this operator generalizes [MS90] valid timeslice operator.

When the TSLICE (or VSLICE) argument is restricted to one TT or VT span, our timeslice operators are reduced to the rollback and valid timeslice operators of [MS90].

Time slice operators are stated as

```
Standard database query without temporal operators
             (returns values or objects)
TSLICE (temporal expression)
VSLICE (temporal expression)
```

Their results can be described as obtained in the following way:

1. obtain TIME hierarchy objects that satisfy TT (TSLICE) and/or VT (VSLICE)
2. obtain the database temporal state restricted to the spans determined in (1), for objects satisfying the query

Figure 3 shows an example of a query combining TSLICE, VSLICE.

4.4 General temporal queries

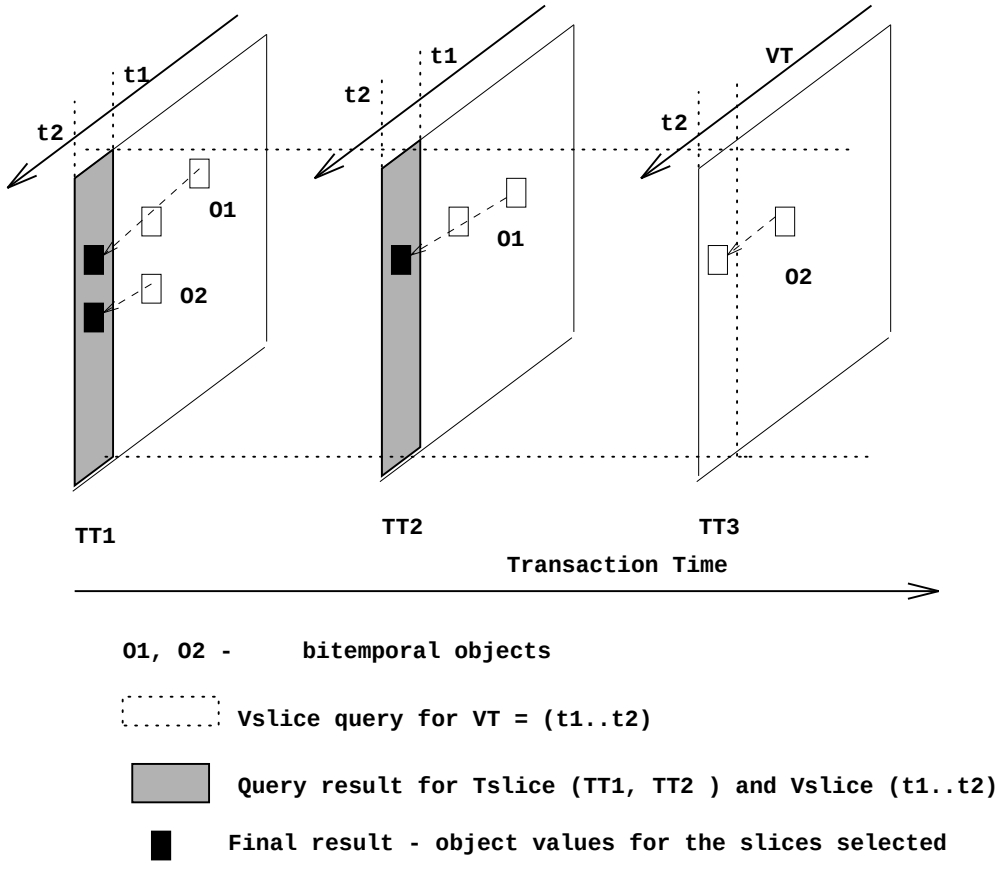
General temporal queries are those where the time expressions apply both to the predicate and to the desired objects. They can be expressed as

```
SELECT (goal) [ { TSLICE   VSLICE }  ]
FROM (target classes)
WHERE (predicate )  [ { VALID |  INDB } clauses ]
(restrict predicate to objects within a specified time period)
```

First, the set of objects satisfying the predicate are determined (being restricted by the temporal expressions associated with the predicate). Next, the answer is given from the state of these objects, given the goal restricted by the VSLICE/TSLICE operators.

5 Conclusions

This paper presented TOD - a temporal object-oriented model, which considers the evolution of objects and behavior – inheritance, composition and schema – over two time dimensions.


t1
t2
O1
O2
t1
t2
O1
VT
t2
O2
TT1
TT2
TT3
Transaction Time
O1, O2 - bitemporal objects
Vslice query for VT = (t1..t2)
Query result for Tslice (TT1, TT2) and Vslice (t1..t2)
Final result - object values for the slices selected

Though many temporal models exist in the literature, there are few proposals for object-oriented systems. TOD extends these proposals by allowing the simultaneous existence of different class categories, evolution of behavior and of schema, and providing constraints for database evolution.

We stress that most of the problems we have treated have not been considered before, since they are due to our having combined two factors which are usually ignored in the literature:

- allowing different class categories in the same database; and
- allowing schema evolution.

These factors have ensured TOD a greater freedom in modelling the real world. It is, therefore, more general than the other temporal object-oriented models.

This model is now being implemented as a layer on top of the O2 database system. The TIME hierarchy has been created as a standard O2 class, and we are implementing the different temporal methods to manage its objects.

The semantics of O2Query is helping the implementation work. The result of a query can be either objects already existing in the database, or a complex value whose type is defined by the query itself. The result of a query can be manipulated by a program and can be used to build new complex objects. This characteristic is very important to our concept of temporal query. It allows us to use the result of a query to build a new temporal database, or to incorporate it as a new component of a given object.

	Data Model	Temporal Dimensions	Value Evolution	Algebra or Query Lang.
[CW83, CC88]	Relational	VT	Step-wise	Algebra
[KL83]	ER	TT	User-defined	None
[LDE+84]	Relational	TT	Step-wise	None
[SA85, SA86, Sno87, MS90]	Relational	VT/TT	Step/Discrete	Both
[Tan86]	Relational	VT	Step-wise	Algebra
[Gad88, GV85, GY88]	Relational	Arbitrary	Step-wise	Both
[AQ86]	ER	TT	Step-wise	Language
[Ari86]	Relational	Arbitrary	Step-wise	Both
[SS87, SS88]	Independent	VT	4 types	Algebra
[MNA87]	Relational	VT	Step- wise	Language
[ABN87]	Relational	VT/TT	Step-wise	None
[LJ88]	Relational	VT	Step/Discrete	Algebra
[Sar90a, Sar90b]	Relational	VT/TT	Step/Discrete	Both
[KRS90]	Relational	VT/TT	4 types	Algebra
[GM91]	Relational	VT	Step-wise	Both
[SC91]	OO	VT	Step-wise	Algebra
[WD92]	OO	Arbitrary	Step-wise	Language
[KS92]	Complex-Obj	VT	Step-wise	Language
TOD	OO	VT/TT	4 types	Language

	Timestamp Level	Timestamp Kind	Schema Evolution	Implementation
[CW83, CC88]	Attribute	Points	$\surd$	$\neg$
[KL83]	Attribute	Points	$\neg$	$\surd$
[LDE+84]	Tuple	Intervals	$\surd$	$\approx$
[SA85, SA86, Sno87, MS90]	Tuple	Points/Intervals	$\surd$	$\surd$
[Tan86]	Attribute	Intervals	$\neg$	$\neg$
[Gad88, GV85, GY88]	Attribute	Set of Intervals	$\neg$	$\neg$
[AQ86]	Variable	Points	$\surd$	$\approx$
[Ari86]	Tuple	Points	$\neg$	$\approx$
[SS87, SS88]	Tuple	Points	$\neg$	$\neg$
[MNA87]	Tuple	Intervals	$\surd$	$\neg$
[ABN87]	Tuple	Points	$\neg$	$\surd$
[LJ88]	Tuple	Points/Intervals	$\neg$	$\surd$
[Sar90a, Sar90b]	Tuple	Intervals	$\neg$	$\surd$
[KRS90]	Tuple	Points	$\neg$	$\approx$
[GM91]	Tuple	Intervals	$\surd$	$\neg$
[SC91]	Object	Intervals	$\surd$	$\neg$
[WD92]	Variable	Intervals	$\neg$	$\surd$
[KS92]	Object	Intervals	$\neg$	$\surd$
TOD	Object	Points/Set Intervals	$\surd$	$\approx$

$\surd$	Mentioned
$\neg$	Not mentioned
$\approx$	Implementation in progress or mapping described

References

[ABN87] T. Abbod, K. Brown, and H. Noble. Providing time-related constraints for conventional database systems. In *Proc 13th VLDB*, pages 167–175, 1987.

[AQ86] M. Adiba and N. Bui Quang. Historical multi-media databases. In *Proc 12th VLDB*, pages 63–70, 1986.

[Ari86] G. Ariav. A temporally oriented data model. *ACM Transactions on Database Systems*, 11(4):499–527, 1986.

[Bee89] C. Beeri. Formal Models for Object-oriented Databases. In *Proc. 1st International Conference on Deductive and Object-oriented Databases*, pages 370–395, 1989.

[CC88] James Clifford and Albert Croker. Objects in time. *IEEE Data Engineering*, 11(4):11–18, 1988.

[CJ90] W. Cellary and G. Jomier. Consistency of Versions in Object-Oriented Databases. In *Proc. 16th VLDB*, pages 432–441, 1990.

[CW83] J. Clifford and D. S. Warren. Formal semantics for time in databases. *ACM Transactions on Database Systems*, 8(2):214–254, June 1983.

[DW92] U. Dayal and G. Wuu. A Uniform Approach to Processing Temporal Queries . In *Proc 18th VLDB*, pages 407–418, october 1992.

[Gad88] S. K. Gadia. A homogeneous relational model and query languages for temporal databases. *ACM Transactions on Database Systems*, 13(4):418–448, December 1988.

[GM91] D. Gabbay and P. McBrien. Temporal Logic & Historical Databases. In *Proc. 17th VLDB*, pages 423–430, 1991.

[GV85] S. K. Gadia and J. H. Vaishnav. A query language for a homogeneous temporal database. In *Proceedings of the 4th ACM SIGACT-SGMOD Symposium on Principles of Database Systems*, pages 51–56, 1985.

[GY88] S. K. Gadia and C. Yeung. A generalized model for a relational temporal database. In *Proc ACM SIGMOD*, pages 251–259, 1988.

[JCG+92] C. Jensen, J. Clifford, S. Gadia, A. Segev, and R. Snodgrass. A Glossary of Temporal Database Concepts. *ACM Sigmod Record*, 21(3):35–43, 1992.

[JS92] C. Jensen and R. Snodgrass. Temporal Specialization . In *Proc IEEE Data Engineering Conference*, pages 594–601, 1992.

[KL83] M. Klopprogge and P. Lockemann. Modelling Information Preserving Databases: Consequences of the Concept of Time. In *Proc 9th VLDB*, pages 399–416, 1983.

[KRS90] W. Kafer, N. Ritter, and H. Schoning. Support for Temporal Data by Complex Objects. In *Proc 16th VLDB*, pages 24–35, 1990.

[KS92] W. Kafer and H. Schoning. Mapping a Version Model to a Complex-Object Data Model . In *Proc IEEE Data Engineering Conference*, pages 348–357, 1992.

[LDE+84] V. Lum, P. Dadam, R. Erbe, J. Guenauer, P. Pistor, G. Walch, H. Werner, and J. Woodfill. Designing DBMS for the temporal dimension. In *Proc ACM SIGMOD*, pages 115–130, 1984.

[LJ88] N. A. Lorentzos and R. G. Johnson. Extending relational algebra to manipulate temporal data. *Information Systems*, 13(3):289–296, 1988.

[MNA87] N. G. Martin, S. B. Navathe, and Rafi Ahmed. Dealing with temporal schema anomalies in history databases. In *Proc 13th VLDB*, pages 177–184, 1987.

[MS90] Edwin McKenzie and R. Snodgrass. Schema evolution and the relational algebra. *Information Systems*, 15(2):207–232, 1990.

[Oli93] L. M. Oliveira. A model for temporal object-oriented databases . Master's thesis, DCC-UNICAMP, april 1993.

[PM92] N. Pissinou and K. Makki. The T-3DIS: an Approach to Temporal Object Databases. In ISMM, editor, *Proc. 1st International Conference on Information and Knowledge Management- CIKM*, pages 185–192, 1992.

[SA85] R. Snodgrass and I. Ahn. A taxonomy of time in databases. In *Proc ACM SIGMOD*, pages 236–246, 1985.

[SA86] R. Snodgrass and I. Ahn. Temporal databases. *IEEE Computer*, 19(9):35–42, 1986.

[Sar90a] N. L. Sarda. Algebra and query languages for a historical data model. *The Computer Journal*, 33(1):11–18, 1990.

[Sar90b] N. L. Sarda. Extensions to SQL for historical databases. *IEEE Transactions on Knowledge and Data Engineering*, 2(2):220–230, 1990.

[SC91] S. Y. W. Su and H. Chen. A Temporal Knowledge Representation Model OSAM*/T and Its Query Language QQL/T. In *Proc. 17th VLDB*, pages 431–442, 1991.

[Sno87] R. Snodgrass. The temporal query language TQUEL. *ACM Transactions on Database Systems*, 12(2):247–298, June 87.

[Sno90] Richard Snodgrass. Temporal Databases Status and Research Directions. *SIGMOD Record*, 19(4):83–89, December 90.

[Soo91] M. Soo. Bibliography on Temporal Databases. *ACM SIGMOD RECORD*, 20(1):14–23, 1991.

[SS87] A. Segev and A. Shoshani. Logical modelling of temporal data. In *Proc ACM SIGMOD*, pages 454–466, 1987.

[SS88] A. Segev and A. Shoshani. The representation of a temporal data model in the relational environment. *Lecture Notes in Computer Science*, 339:39–61, 1988.

[Tan86] A. U. Tansel. Adding time dimension to relational model and extending relational algebra. *Information Systems*, 11(4):343–355, 1986.

[WD92] G. T. Wuu and U. Dayal. A Uniform Model for Temporal Object-Oriented Databases. In *Proc. IEEE Data Engineering Conference*, pages 584–593, 1992.

Relatórios Técnicos – 1992

01/92 **Applications of Finite Automata Representing Large Vocabularies,** *C. L. Lucchesi, T. Kowaltowski*

02/92 **Point Set Pattern Matching in d-Dimensions,** *P. J. de Rezende, D. T. Lee*

03/92 **On the Irrelevance of Edge Orientations on the Acyclic Directed Two Disjoint Paths Problem,** *C. L. Lucchesi, M. C. M. T. Giglio*

04/92 **A Note on Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams,** *W. Jacometti*

05/92 **An (l, u)-Transversal Theorem for Bipartite Graphs,** *C. L. Lucchesi, D. H. Younger*

06/92 **Implementing Integrity Control in Active Databases,** *C. B. Medeiros, M. J. Andrade*

07/92 **New Experimental Results For Bipartite Matching,** *J. C. Setubal*

08/92 **Maintaining Integrity Constraints across Versions in a Database,** *C. B. Medeiros, G. Jomier, W. Cellary*

09/92 **On Clique-Complete Graphs,** *C. L. Lucchesi, C. P. Mello, J. L. Szwarcfiter*

10/92 **Examples of Informal but Rigorous Correctness Proofs for Tree Traversing Algorithms,** *T. Kowaltowski*

11/92 **Debugging Aids for Statechart-Based Systems,** *V. G. S. Elias, H. Liesenberg*

12/92 **Browsing and Querying in Object-Oriented Databases,** *J. L. de Oliveira, R. de O. Anido*

Relatórios Técnicos – 1993

01/93 **Transforming Statecharts into Reactive Systems,** *Antonio G. Figueiredo Filho, Hans K. E. Liesenberg*

02/93 **The Hierarchical Ring Protocol: An Efficient Scheme for Reading Replicated Data,** *Nabor das C. Mendonça, Ricardo de O. Anido*

03/93 **Matching Algorithms for Bipartite Graphs,** *Herbert A. Baier Saip, Cláudio L. Lucchesi*

04/93 **A lexBFS Algorithm for Proper Interval Graph Recognition,** *Celina M. H. de Figueiredo, João Meidanis, Célia P. de Mello*

05/93 **Sistema Gerenciador de Processamento Cooperativo,** *Ivonne. M. Carrazana, Nelson. C. Machado, Célio. C. Guimarães*

06/93 **Implementação de um Banco de Dados Relacional Dotado de uma Interface Cooperativa,** *Nascif A. Abousalh Neto, Ariadne M. B. R. Carvalho*

07/93 **Estadogramas no Desenvolvimento de Interfaces,** *Fábio N. de Lucena, Hans K. E. Liesenberg*

08/93 **Introspection and Projection in Reasoning about Other Agents,** *Jacques Wainer*

09/93 **Codificação de Seqüências de Imagens com Quantização Vetorial,** *Carlos Antonio Reinaldo Costa, Paulo Lício de Geus*

10/93 **Minimização do Consumo de Energia em um Sistema para Aquisição de Dados Controlado por Microcomputador,** *Paulo Cesar Centoducatte, Nelson Castro Machado*

11/93 **An Implementation Structure for RM-OSI/ISO Transaction Processing Application Contexts,** *Flávio Morais de Assis Silva, Edmundo Roberto Mauro Madeira*

12/93 **Boole's conditions of possible experience and reasoning under uncertainty,** *Pierre Hansen, Brigitte Jaumard, Marcus Poggi de Aragão*

13/93 **Modelling Geographic Information Systems using an Object Oriented Framework,** *Fatima Pires, Claudia Bauzer Medeiros, Ardemiris Barros Silva*

Departamento de Ciência da Computação — IMECC
Caixa Postal 6065
Universidade Estadual de Campinas
13081-970 – Campinas – SP
BRASIL

`reltec@dcc.unicamp.br`