TOOA: A Temporal Object-Oriented Algebra

Ellen Rose[1] and *Arie Segev*[2]
The University of Toledo[1]
Toledo, OH 43606 and
Information and Computing Sciences Division
Lawrence Berkeley Laboratory

Walter A. Haas School of Business[2]
The University of California and
Information and Computing Sciences Division
Lawrence Berkeley Laboratory
Berkeley, California 94720

ABSTRACT

In this paper, we present a temporal, object-oriented algebra which serves as a formal basis for the query language of a temporal, object-oriented data model. Our algebra is a superset of the relational algebra in that it provides support for manipulating temporal objects, temporal types, type hierarchies and class lattices, multiple time-lines, and correction sequences in addition to supporting the five relational algebra operators. Graphs are used as the visual representations of both the schema and the object instances. The algebra provides constructs to modify and manipulate the schema graph and its extension, the object graph. The algebra operates on a collection or collections of objects and returns a collection of objects. This algebra is a first step in providing a formal foundation for query processing and optimizing in a temporal, object-oriented data model.

This work was supported by an NSF Grant Number IRI-9000619 and by the Applied Mathematical Sciences Research Program of the Office of Energy Research, U.S. Department of Energy under Contract DE-AC03-76SF00098.

1. Introduction

The increase in complexity of new applications such as computer aided design, office information systems, scientific databases and multimedia databases has pressed the limits of the relational model and led to research into next-generation data models. These new data models must capture the semantics of complex objects and treat time as a basic component versus as an additional attribute in order to build more accurate models. New primitives to handle temporal data and complex types need to be incorporated into the query and specification languages of these models. An algebra to transform these temporal, object-oriented model queries into a more efficient form for execution is the focus of this paper.

The approach taken herein evolved from the integration of research on extensions of the relational model to include temporal semantics and complex objects and the recent work on object-oriented models. The literature on temporal extensions to the relational model proposes adding temporal semantics through tuple or attribute versioning. Research in these directions includes: [Clifford & Warren 83], [Clifford & Croker 87], [Snodgrass 87], [Segev & Shoshani 87], [Gadia 88b], [Gadia & Yeung 88] and [Navathe & Ahmed 89]. A good summary of temporal extensions to the relational algebra can be found in [McKenzie & Snodgrass 91]. [Tuzhilin & Clifford 90] propose an algebra that is restricted to temporal relational models which support a linear, discrete bounded model of time and propose a new linear recursive operator. Generalized logical temporal models have been proposed by [Ariav 86] and [Segev & Shoshani 87]. A comprehensive bibliography on temporal databases and references to three previous bibliographies can be found in [Soo 91]. Extensions to the Entity-Relationship model incorporate time as either an entity or attribute and include [Klopprogge & Lockemann 83], [Ferg 85] and [Elmasri & Wuu 90].

Recent work on object-oriented algebras include OQL and its algebra in [Alashqur etal 89] and [Guo etal 91] respectively. OQL operates on sets of association patterns which are derived from a graph of object instances. OQL includes a non-association operator and can operate on heterogeneous sets of objects. Our approach is most similar to that found in [Guo etal 91]. [Shaw & Zdonik 90] developed a query language and algebra for ENCORE which supports object identity and the creation of new objects and types. EXTRA [Carey etal 88] and its query language EXCESS were developed using a database generator EXODUS. EXCESS supports querying nested or un-nested sets, arrays, tuples and individual objects similar to O_2 and user-defined functions as in POSTGRES. The above list of OO-Algebras is not meant to be exhaustive but rather to mention those algebras which have features which are more closely related to TOOA.

Research on schema evolution has taken two forms: single-schema modification and schema versioning. Most proposals [Banerjee, etal 88], and [Penny & Stein 87] take the single-schema approach. Disadvantages of this approach include the loss of historical values when an attribute is dropped and the forcing of all users in a multi-user environment to accept all schema changes as *part of their view. Schema versioning in the ENCORE system [Skarra & Zdonik*

86] and in ORION [Kim & Chou 88] [Kim etal 89] doesn't have the aforementioned problems since it allows any user to view any data base through the schema version of their choice. [Skarra & Zdonik 86] deal with versioning class types, not an entire database schema. Any change to a class type creates a new version of the type and its sub-types. Therefore, the schema of the database at any moment will consist of the selected version of each type and the links between them. [Kim & Chou 88] [Kim etal 89] and [Andany etal 91] are the only papers found which deal with database schema versioning. In any version of the database schema, only one version of each distinct class object is allowed. [Kim & Chou 88] develop a model of database versions based on [Chou 88]'s model of object versions and provide implementation techniques to support database schema versioning. The basic premise of their proposal is to allow derived schema versions to inherit all the objects of the schema version from which they are derived. The user is allowed to specify if the objects will be inherited and whether or not they can be updated through the parent. In this approach, any change to the database schema creates a new version of the database schema which can lead to the problem of managing a large number of versions. [Andany etal 91] elaborate on the alternative approach of handling schema modification using view definitions proposed in [Kim & Chou 88] in the Farandole 2 model. In this approach the concept of a context, a connected subgraph of the database schema graph, is used as the level of granularity for versioning. Whereas, the previous approaches look only at structural consistency, [Zicari 91] also looks at consistency in the behaviour of methods after a schema change has occurred. To date, no database systems have incorporated schema versioning [Kim 90]. Our approach is one of schema histories at the type level using the time-sequence concept.

Several papers which focus on temporal, object-oriented models have recently appeared in the literature. [Kafer & Schoning 92] extend a complex object model, adding the following attributes: references to the previous and next tuple, a boolean existence attribute, a transaction time attribute, an object identifier and a valid time point. This model is actually a nested relational model which adopts tuple timestamping and therefore does not allow cyclic queries. [Su & Chen 91] proposed an object-based temporal knowledge representation model called OSAM*/T and query language called OQL/T. Here rules are used to capture temporal semantics other than the valid start and end times of a tuple. [Wuu & Dayal 91] add temporal semantics to OODAPLEX. In this model, a POINT type is the supertype of the TIME and VERSION types. Time varying properties are modeled as functions that map time objects to non-temporal snapshot values. A more detailed comparison with these other works is given in section 6 after the features of TOODM have been explained.

In this paper, a temporal, object-oriented algebra (called TOOA) for the manipulation of the intension (types) and extension (objects) of a temporal, object-oriented data model is proposed. The time-sequence collection type [Rose & Segev 91] is used to model histories of attribute values of objects and histories of the set of properties which define an object's type. Each time-sequence has an associated sequence of corrections which can be merged with the original history.

Multiple time-lines are used in the time-sequence collection type to allow for both historical and rollback queries.

The remainder of this paper is organized as follows. Section 2 presents an example which is used to discuss the basic concepts of the data model (TOODM) for which the algebra is designed. Section 3 gives an overview of the algebra, its notations and assumptions. Section 4 outlines the object level operators and section 5 describes the schema level update operators. It should be noted that the algebra only applies to retrieval operators not to the update operators. Finally, section 6 concludes the paper with a discussion of how our proposal compares with other temporal extensions of object-oriented data models and also delineates future work.

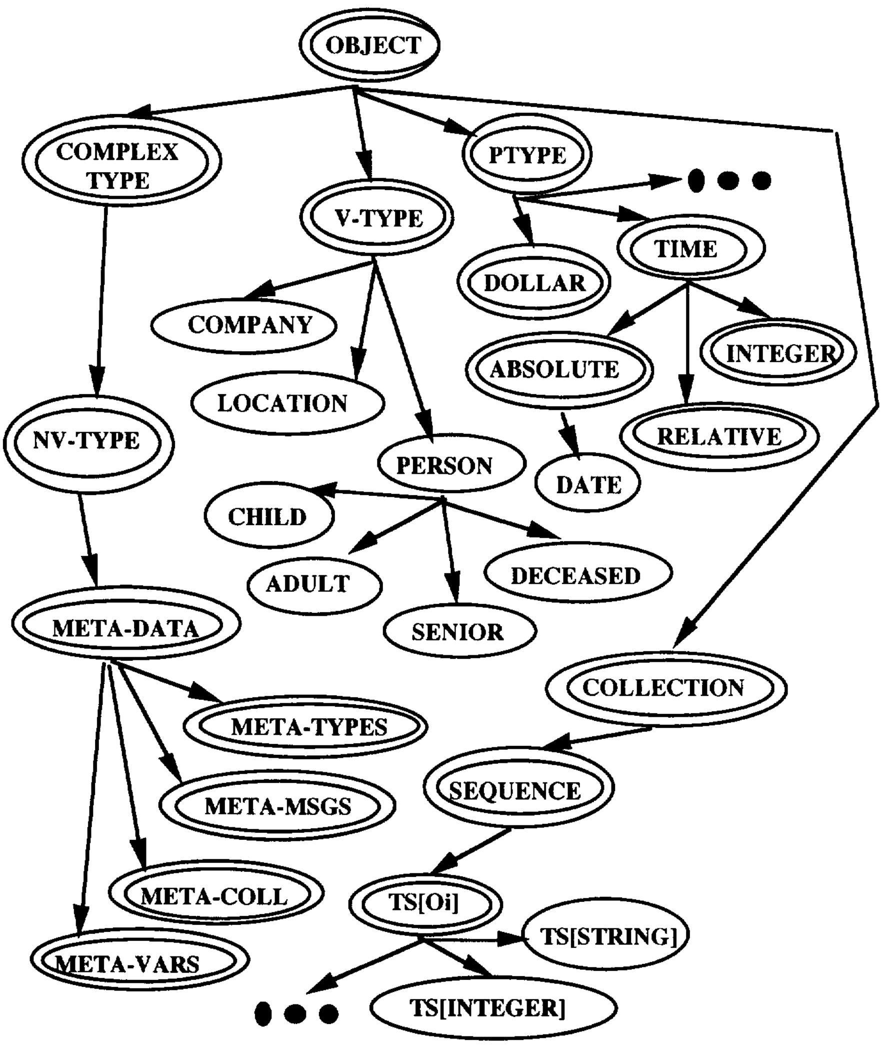
2. Motivating Example and Data Model

The model upon which our algebra operates is the temporal, object-oriented data model (TOODM) in [Rose & Segev 91]. In TOODM, a class is viewed as the extension of the type of an object instance, hereafter referred to as object. A type serves as a template to encapsulate the structure (attributes), behavior (messages/methods) and constraints of an object. Each object has a globally unique system generated identifier that is independent of the object's state or physical location. The state of an object is described by the values held by its attributes. New types must be defined as a subtype of existing types in the type hierarchy. A subtype inherits the properties of its supertypes. Figure 1 shows the types which have been defined in the type hierarchy for our social security database example as well as the system-defined types. This database would track individuals as they progress from childhood to adulthood, old age, retirement and death. At each point in an individual's life, a different set of relevant information may be required. An individual may be described by a different type template at different stages of their life and belong to different classes simultaneously (e.g. every adult is also a person) or at different time points (e.g. a person can't be both a child and an adult simultaneously).

User-defined COMPLEX TYPES such as PERSON are defined as subtypes of V-TYPE which means versionable type. A history is kept of the type definition of all types defined as subtypes of V-TYPE. META-DATA are subtypes of NV-TYPE, meaning non-versionable type, since a history of their type definitions is not maintained. The instances of a subtype of NV-TYPE are versionable. PTYPE, primitive types, (e.g. integers, time, etc.) have atomic values and a set of operators. Collection types, such as SET, TUPLE, SEQUENCE, and LIST are parameterized types used to build the extensions of other types. A special collection type called time-sequence,[1] TS[O_i] is illustrated in Figure 2. TS[O_i] has the type O_i as its parameter. Each time sequence object has the history and correction history (corr-history) properties which each contain a sequence of (A; TL) pairs. "A" represents a tuple of attribute values and "TL" represents a tuple of time-line values, (TL_i), having type time (T_i) or one of its specializations. Valid time (vt)

[1] The time-sequence object and some of its operators are based on concepts from [Segev & Shoshani 87].

Figure 1 - Type Hierarchy for Social Security Database



◯ User-Defined Types

◎ System-Defined Types in TOODM

Note: All subtypes of the time sequence collection type, TS[Oi], are not shown in the diagram; all SET[Oi] and PTYPE objects are also not included to simplify the diagram.

Figure 2 - Time Sequence Collection Type, TS [Oi]

Define TS [Oi] as subclass of Sequence [Oi]

surr: OID

history: {(A;TL)} where A = TUPLE (Ai)
TL = TUPLE (TLi : Ti)

and data-type (Ai) : {SWC, Event, CONT, User}

and gran (TLi) : CalendarSet or Ordinal

and interp-f (Ai) : Functions

default-order: TLi in TL

corr-history: same as history except value(tt) > value(tt)
in the history sequence

lifespan: Union of Temporal Elements in history

●

●

●

attributes

COUNT( ) FIRST( ) LAST( ) Nth ( )

Accumulate(Agg-Op(Target-Ai),Grp(acc-pred))

Aggregate(Agg-Op(Target-Ai),Grp(TLi, gran-value)

Select (LIST[Ai], Conditions(Ai or TLi))

Restrict (TS, condition(another property of Oi))

Merge (History-TS,Corr-TS, Temporal Element)

●

●

●

operators

and transaction time (tt) are mandatory members of this tuple. Each attribute A_i has a data-type which is step-wise constant (SWC), event, continuous (CONT) or user-defined. If the data-type is user-defined, an interpolation function (interp-f) should be provided to determine values for time points which are not recorded as data points. Each time-line, TL_i has a granularity which may be a calendar time (CalenderSet) or an ordinal number. The transaction time value for a correction history element is restricted to be greater than that of the objects in the associated history sequence.

The lifespan attribute is defined as a possibly (but not necessarily) non-contiguous union of temporal elements of the history property. Temporal elements [Gadia 88a] are finite unions of time-lines. Temporal elements are closed under intersection, complement and union. An object's lifespan is the union of it's lifespans in all classes in which it has participated.

The operators in [Segev & Shoshani 87] are defined for time-sequences with one or more attributes but only one time-line, vt. These operators are extended to handle multiple time-lines in TOODM through the representation of TL_i as a tuple of time-lines. Since information can be viewed with and without corrections taken into consideration, a merge operator was developed in TOODM. Count( ), First( ), Last( ) and Nth ( ) return the number of data points in the sequence, the first pair, last pair and nth pair respectively.

Figure 3 shows example type definitions for META-DATA and user-defined types in the example. Only the attributes of the user-defined types are shown to simplify the diagram. Attributes of meta-data types, except the object identifier, take values from time-sequence classes so the database schema can contain different sets of types, constraints, etc. at different time points. The class SET[PERSON] has a set structure and the value structure of one of its instances would be a set of PERSON identifiers. INV indicates that the employer property of Adult and the employees property of Company are inverses representing a relationship.

2.1. Schema Graph

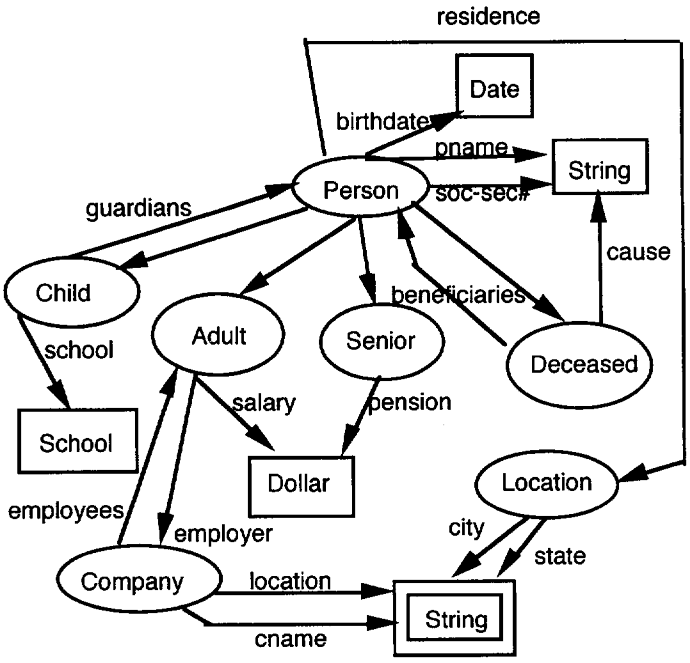
The schema graph, SG, in Figure 4 is a graphical representation of the user-defined properties in Figure 3. Unlabeled arcs represent is-a links directed from a super-type (e.g. PERSON) to its subtypes (e.g. CHILD, ADULT, SENIOR and DECEASED). The schema graph serves as the logical representation of the database. It consists of a set of nodes (types) and a set of links (properties of the types). The set of types is a disjoint union of the set of Primitive Types (PTYPE) and the set of COMPLEX TYPES. Each complex type has an identifier, attributes, a supertype list, operations and constraints. This information is stored in the extensions of the meta-data types shown in Figure 3. If a property is single-valued, its values are stored as a TS[O_i]. If the property is multi-valued its values are stored as TS[SET or LIST[O_i]].

A link in the schema graph represents a property and is directed from its defining object type to its domain. Since more than one link can exist between the same two types, all link types have a unique name. For example, $L^{residence}_{PERSON,LOCATION}$

Figure 3 - Type Definitions of Meta-Data and User-Defined Types

```
Define Meta-Types as subclass of Meta-Data
    Type_Oid: OID
    Type_Name:  TS [String]
    Superclasses: TS [ LIST [Meta_Types] ]
    Subclasses: TS [ SET [Meta_Types] ]
    Attributes: TS [Set [Meta_Vars] ]
    Operations: TS [Set [Meta_Msgs] ]
    Class_Attributes: TS [Set [Meta_Class_Vars] ]
    Relationships: TS [Set [Meta_Rels] ]
    Constraints: TS[ Set [BOOL] ]

Define Meta-Vars as subclass of Meta-Data
    var-oid: OID
    var-signature: TS [<var-name:String,
        var-parent:Meta-Type,
        var-creator : UserID,
        var-type: Meta-Types> ]
    constraints: TS[ Set [BOOL] ]

Define Meta-Msgs as subclass of Meta-Data
    msg-oid: OID
    msg-signature: TS [ <msg-name:String,
        msg-parent: Meta-Types,
        msg-creator: UserID,
        msg-params: Set [Params],
        msg-code:<Code,Userid> > ]
    pre-conditions: TS[Set [BOOL] ]
    post-conditions: TS[ Set [BOOL] ]

Define Meta-Coll as subclass of Meta-Data
    coll-oid: OID
    coll-name:  String
    operators: TS [SET [Operators]
    constraints: TS [Set [BOOL ] ]

Define Person as SubType of V-Type
    soc-sec#: TS [String]
    pname: TS [String]
    birthdate: TS [Date]
    residence: TS [Location ]

Define Child as SubType of Person
    guardians: TS [ SET [Person] ]
    school: TS [School]

Define Adult as SubType of Person
    employer: TS [Company]
        INV is Company.employees
    salary: TS [Dollar]
    dependents: TS [SET [Person] ]

Define Senior as SubType of Person
    pension: TS [Dollar]

Define Deceased as SubType of Person
    cause: TS [String ]
    beneficiaries: TS [SET [Person] ]

Define Company as SubType of V-Type
    cname: TS [String]
    employees: TS [SET [Person] ]
        INV is Adult.employer
    location: TS [Location ]

Define Location as SubType of V-Type
    city: TS [ String ]
    state: TS [ String ]
```

Figure 4 - Schema Graph of Figure 3 User-Defined Types



Note 1: unlabeled arcs are is-a relationships and the String type only appears once but was drawn twice for clarity of the diagram.

Note 2: each arc has a lifespan indicating when it was a property of the type it is directed from; lifespans are temporal elements and may be non-contiguous indicates that each PERSON has a residence whose value is in the class LOCATION and a time-sequence of the location values is kept. Properties with primitive domains can't be represented as inverse links. Recall that relationships are described in the type by attaching the word INV to the property as in Figure 3 where the employer property of ADULT and the employees property of COMPANY are inverses.

2.2. Objects and the Object Graph

Each object has a unique instance identifier (IID) which is a concatenation of the type identifier and the object identifier. The object graph, OG, is the extension of the schema graph. Its nodes are IID's and links represent associations from IID's to time-sequence objects. In Figure 5, person_1 (p1) has subtype objects child_1 (ch1) and adult_1 (a1). Each of these objects is linked to time-sequence objects. For example, the values of the social-security attribute of p1 are found in time-sequence ts10 which indicates the social security number was recorded as "333-33-3333" at tt=71 and is valid from vt=71 into the future (now+).

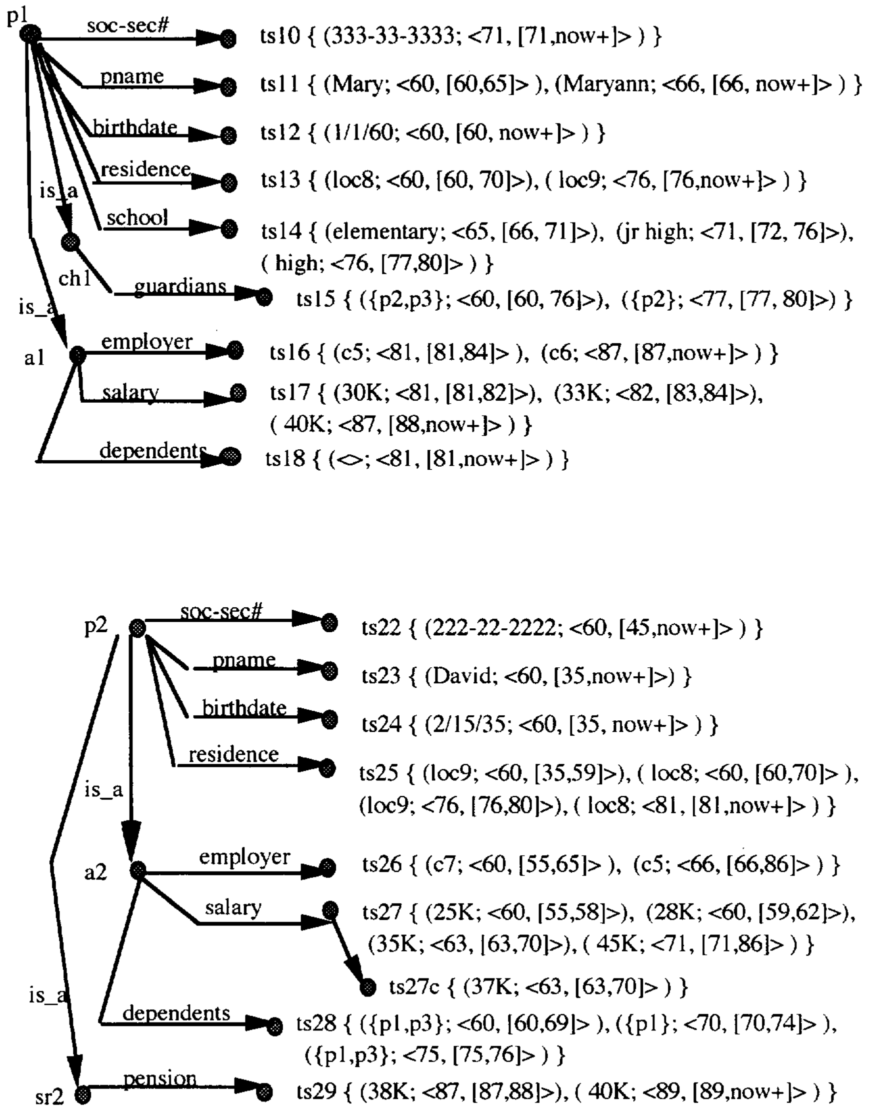
Deletion of information is represented by a gap in the time-sequence values as in ts33 where no residence information was recorded between vt=[71,75]. In Figure 6, ts68, attribute variable (V4=residence) was deleted from Person's type definition for the period vt=[71,75] at tt=71 corresponding to the deleted data in Figure 5. If the information was not available but was defined in the type, it would have appeared in the time-sequence as a null[2] value for vt=[71,75]. The residence link for p3 has a lifespan of <60,[36,70]> $\cup$ <76, [71,76]>. The lifespan of a link in the object graph is the same as the lifespan of its time-sequence object. The transaction time associated with a period where no value was recorded is the previous transaction time where the association existed.

Relationships are constrained in terms of lifespans of the participants. For instance, the values of company c5 in Figure 5 appear in ts41 through the employees link instance which has a lifespan of [66,86]. The first pair in ts41 associates person p2 with the time [66,80], the second pair associates p1 and p2 with time [81,84] and the third pair associates p2 with time [85,86]. Since the employer link is an inverse in the schema graph, c5 must appear in each pair in ts16 (p1's employer link) which has a valid time that intersects with [81,84] and in each pair in ts26 (p2's employer link) that intersects with [66,86].

Non-temporal data is a degenerative case where the lifespan of the link consists of a single element of its time-sequence object. The end point of the valid time interval would be now+ as in the cause of death property.

[2] Different representations for nulls including not-available and unknown can be used.

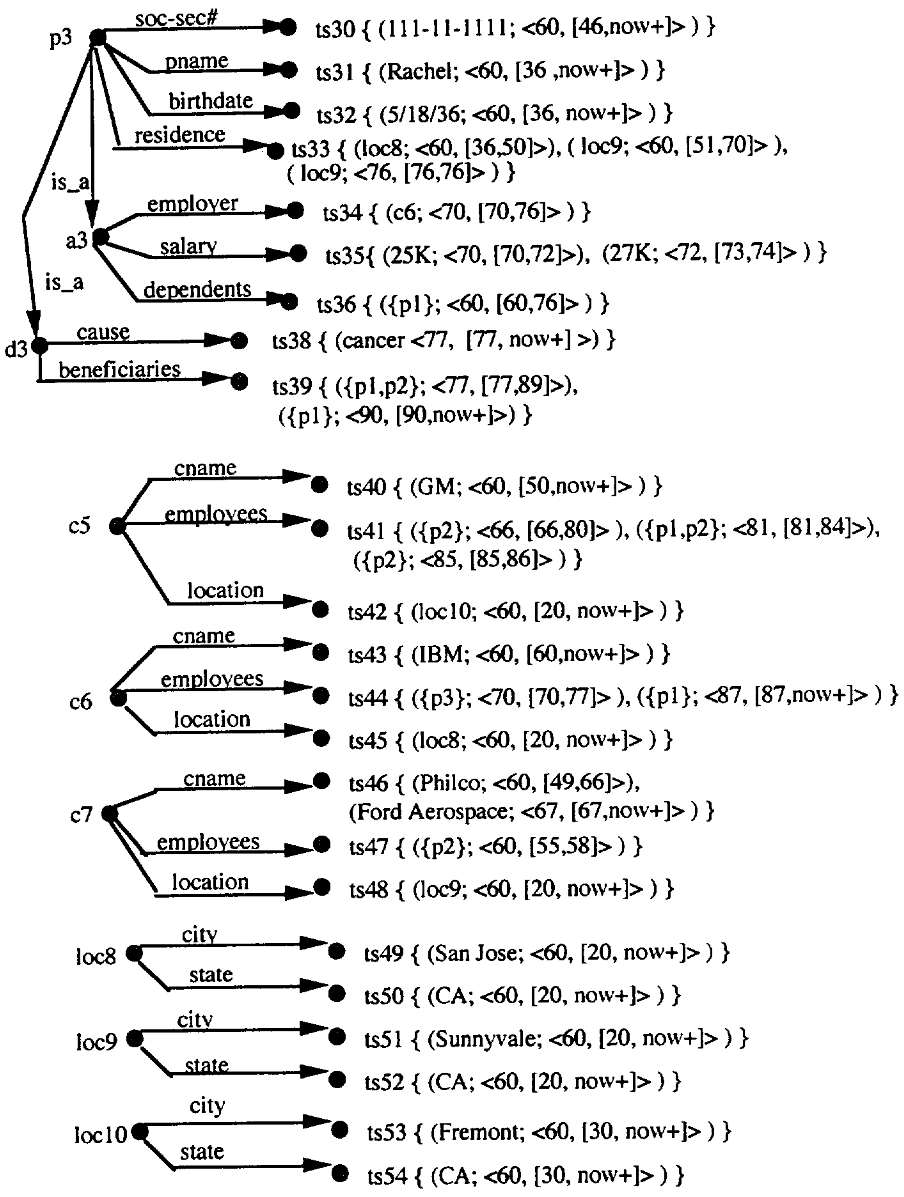
Figure 5 - Object Graph of Some Instances of User-Defined Types


p1
soc-sec#
ts10 { (333-33-3333; <71, [71,now+]>) }
pname
ts11 { (Mary; <60, [60,65]>), (Maryann; <66, [66, now+]>) }
birthdate
ts12 { (1/1/60; <60, [60, now+]>) }
residence
ts13 { (loc8; <60, [60, 70]>), (loc9; <76, [76,now+]>) }
is_a
school
ts14 { (elementary; <65, [66, 71]>), (jr high; <71, [72, 76]>), (high; <76, [77,80]>) }
ch1
guardians
ts15 { ({p2,p3}; <60, [60, 76]>), ({p2}; <77, [77, 80]>) }
is_a
a1
employer
ts16 { (c5; <81, [81,84]>), (c6; <87, [87,now+]>) }
salary
ts17 { (30K; <81, [81,82]>), (33K; <82, [83,84]>), (40K; <87, [88,now+]>) }
dependents
ts18 { (<>; <81, [81,now+]>) }
p2
soc-sec#
ts22 { (222-22-2222; <60, [45,now+]>) }
pname
ts23 { (David; <60, [35,now+]>) }
birthdate
ts24 { (2/15/35; <60, [35, now+]>) }
residence
ts25 { (loc9; <60, [35,59]>), (loc8; <60, [60,70]>), (loc9; <76, [76,80]>), (loc8; <81, [81,now+]>) }
is_a
a2
employer
ts26 { (c7; <60, [55,65]>), (c5; <66, [66,86]>) }
salary
ts27 { (25K; <60, [55,58]>), (28K; <60, [59,62]>), (35K; <63, [63,70]>), (45K; <71, [71,86]>) }
ts27c { (37K; <63, [63,70]>) }
is_a
dependents
ts28 { ({p1,p3}; <60, [60,69]>), ({p1}; <70, [70,74]>), ({p1,p3}; <75, [75,76]>) }
sr2
pension
ts29 { (38K; <87, [87,88]>), (40K; <89, [89,now+]>) }

Note:
The time sequence instances "ts", are sequences of pairs (attribute-tuple; time-line tuple), where the first element of the time-line tuple is record time and the second is valid time.

pi - person i, ai - adult i, chi - child i, sri - senior i, tsi - time sequence i, di - deceased i, ci - company i, loci - location i

Figure 5 - Object Graph of Some Instances of User-Defined Types


p3
soc-sec#
ts30 { (111-11-1111; <60, [46,now+]>) }
pname
ts31 { (Rachel; <60, [36 ,now+]>) }
birthdate
ts32 { (5/18/36; <60, [36, now+]>) }
residence
ts33 { (loc8; <60, [36,50]>), (loc9; <60, [51,70]>),
(loc9; <76, [76,76]>) }
is_a
a3
employer
ts34 { (c6; <70, [70,76]>) }
salary
ts35{ (25K; <70, [70,72]>), (27K; <72, [73,74]>) }
is_a
dependents
ts36 { ({p1}; <60, [60,76]>) }
d3
cause
ts38 { (cancer <77, [77, now+] >) }
beneficiaries
ts39 { ({p1,p2}; <77, [77,89]>),
({p1}; <90, [90,now+]>) }
c5
cname
ts40 { (GM; <60, [50,now+]>) }
employees
ts41 { ({p2}; <66, [66,80]>), ({p1,p2}; <81, [81,84]>),
({p2}; <85, [85,86]>) }
location
ts42 { (loc10; <60, [20, now+]>) }
c6
cname
ts43 { (IBM; <60, [60,now+]>) }
employees
ts44 { ({p3}; <70, [70,77]>), ({p1}; <87, [87,now+]>) }
location
ts45 { (loc8; <60, [20, now+]>) }
c7
cname
ts46 { (Philco; <60, [49,66]>),
(Ford Aerospace; <67, [67,now+]>) }
employees
ts47 { ({p2}; <60, [55,58]>) }
location
ts48 { (loc9; <60, [20, now+]>) }
loc8
city
ts49 { (San Jose; <60, [20, now+]>) }
state
ts50 { (CA; <60, [20, now+]>) }
loc9
city
ts51 { (Sunnyvale; <60, [20, now+]>) }
state
ts52 { (CA; <60, [20, now+]>) }
loc10
city
ts53 { (Fremont; <60, [30, now+]>) }
state
ts54 { (CA; <60, [30, now+]>) }

3. An Overview of TOOA

3.1. Assumptions

Events are viewed as durationless and cause an object to change state. The taxonomy of time given in [Snodgrass & Ahn 85] for valid time (vt) and transaction time (tt) is adopted here. The state of an object persists over some duration of time and is represented by the values held by the attributes of the object over this time duration. The time period over which a state holds is the valid time. The transaction time refers to when an object was recorded. Time is represented as a primitive type which is independent of events. Time can be subtyped into absolute and relative time values as well as specialized time types with different orderings. A partial ordering is useful in representing alternate versions of design objects.

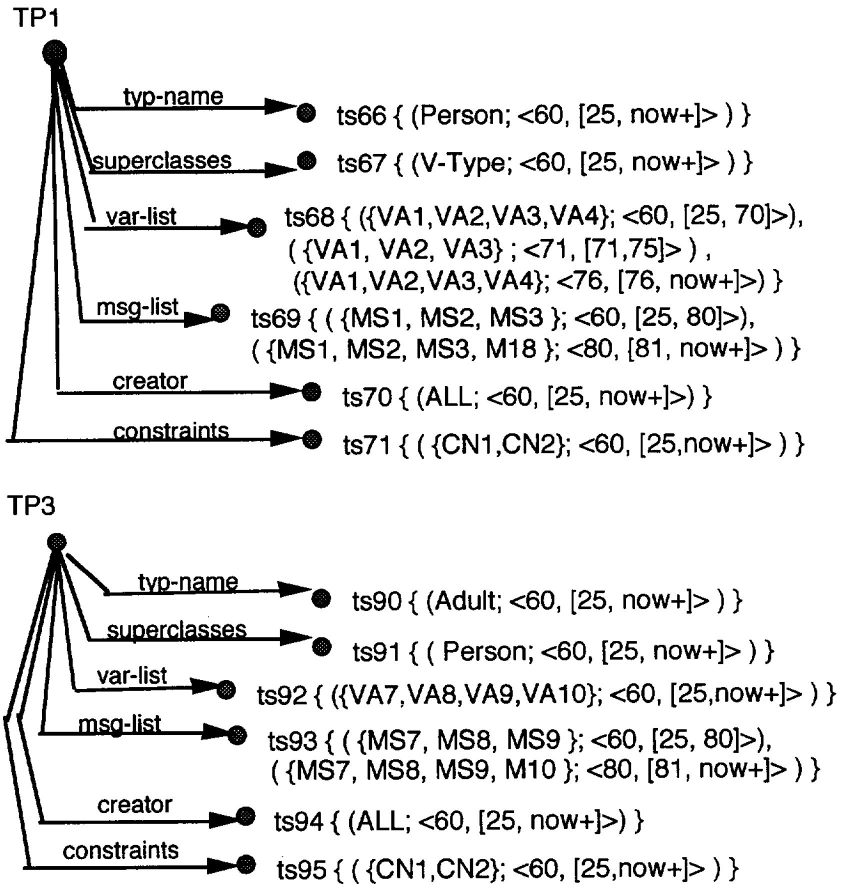
The semantics of existing operations -- retrievals, corrections and updates differ in TOODM. Retrievals can result in a time-sequence of values. Corrections are illustrated in Figure 5 where ts27c is the corr-history for ts27. The query, "Find David's salary at vt=64 as seen from tt=69 accounting for corrections made during tt=[64,69]", can be answered by doing merge(ts27,ts27c,[64,69]). The result with corrections is 37K whereas it would be 35K without. Updates (add, change, delete) also differ from those of a static data model. For instance, changing an existing attribute's value means adding an element to the time-sequence of values representing the attribute. Changes to an existing object implies ending its valid time interval one unit prior to the time of the change, at t-1 and starting the valid time interval of the new value at time t. The end point of the new value's valid time interval could be a specific time point or now+ if it is assumed to hold in the future and present. Adding a new object to a class requires creating time-sequence objects to represent each of its properties. Deletion of an object implies the end of its lifespan without the removal of the information from the database and that the object can't be referenced by any other objects from the time of deletion forward until the time if any that the object re-enters the system.[3] These operators are defined in the type called OBJECT in Figure 1.

3.2. Overview of the Operators

Operations can be applied to a class or the hierarchy rooted at a class, meaning the result may contain heterogeneous objects with different sets of properties. This is different from the relational algebra which only allows operations on homogeneous sets of objects, specifically sets of tuples where all tuples in a particular relation have the same set of attributes. The following two sections discuss the object level and schema level operators in terms of the example instances found in Figures 5 and 6.

[3] Historical information for "deleted" objects can be queried since the history of the object's past states prior to the deletion exists in the database.

Figure 6 - Object Graph of Some Instances of Meta-Types


TP1
typ-name
ts66 { (Person; <60, [25, now+]>) }
superclasses
ts67 { (V-Type; <60, [25, now+]>) }
var-list
ts68 { ({VA1,VA2,VA3,VA4}; <60, [25, 70]>), ({VA1, VA2, VA3} ; <71, [71,75]>) , ({VA1,VA2,VA3,VA4}; <76, [76, now+]>) }
msg-list
ts69 { ({MS1, MS2, MS3 }; <60, [25, 80]>), ({MS1, MS2, MS3, M18 }; <80, [81, now+]>) }
creator
ts70 { (ALL; <60, [25, now+]>) }
constraints
ts71 { ({CN1,CN2}; <60, [25,now+]>) }
TP3
typ-name
ts90 { (Adult; <60, [25, now+]>) }
superclasses
ts91 { (Person; <60, [25, now+]>) }
var-list
ts92 { ({VA7,VA8,VA9,VA10}; <60, [25,now+]>) }
msg-list
ts93 { ({MS7, MS8, MS9 }; <60, [25, 80]>), ({MS7, MS8, MS9, M10 }; <80, [81, now+]>) }
creator
ts94 { (ALL; <60, [25, now+]>) }
constraints
ts95 { ({CN1,CN2}; <60, [25,now+]>) }

Note: The time sequence instances "ts", are sequences of pairs (attribute-tuple; time-line tuple), where the first element of the time-line tuple is record time and the second is valid time.

VA1 = soc-sec#, VA2 = pname, VA3 = birthdate, VA4 = residence, VA5 = school, VA6 = guardians, VA7 = employer, VA8= salary, VA9 = dependents, VA10 = cname, VA11 = employees and VA12 = location, TPi - Type i, MSi - Message i, VAi - Variable i, tsi - time sequence i, CNi - Constraint i

4. Object Level Operators

4.1. Objects with Tuple Values

1) $\bullet_T^k$, Temporal Association

This binary operator returns a collection of graphs of objects. For each graph in the result, an instance of the class or class hierarchy specified by the first operand is associated through link k with an instance or instances of the class or hierarchy rooted at the class in the second operand at time T. If the class represented by the second operand is not primitive, then the association operator can be applied again to retrieve a value of an attribute of that class. We can think of the association operator as a navigational operator that allows us to specify a path through the schema graph that existed at some time T for each link. This means that T must be contained in the lifespan of link k for the operation to be valid since the link is only defined during its lifespan. For example, we can specify a path from the class Adult to Company to String where the link between Adult and Company is employer and the link between Company and String is cname and T is specified as [74,76] for both links. The result of the following is shown in Figure 7a: $Adult \bullet_{[74,76]}^{employer} Company \bullet_{[74,76]}^{cname} String$

2) $|_T$, Temporal Complement

Whereas the previous operator returns a TS[values] that are associated with an object through one of its attributes, this operator returns the members of the second operand that were not associated with the objects belonging to the first operand during time T. For example, we can retrieve the adults who were not employed according to their employment history in the database at time T = [85,87]. We also need to know when this property was defined in the type definition. From Figure 6, we see the employer property (VA7) was defined for type Adult (TP3) at transaction time 60 and was retroactively valid from time 25 to now+. This means that T = [85,87] is valid since it is contained in the lifespan of the property employer for type Adult in the schema. From Figure 5, we see that a_1 was not employed during [85,86], a2 was not employed during [87,now+] and a3 was not employed during [60,69] and only the first two adults were not employed during time T. Note that a3's lifespan ends prior to T and therefore has no employment history after her death. We can express this Temporal Complement operation as follows: $Adult \ |_{[85,87]}^{employer} Company$ where the result appears in Figure 7b.

3) $\sigma\text{–}IF_{TP}$, Temporal Selection Using a Temporal Predicate

Select-IF is a unary operator which returns the set of objects that meet the temporal predicate condition specified by TP. The temporal predicate consists of a non-temporal FOL predicate (NTP), a quantifier (Q) over time (for all or for some) and a temporal element (T) over which the selection is to be taken. The NTP can consist of several predicates combined with the boolean connectives: "and", "or" and "not". Comparisons must be made between objects of the same type: primitive to primitive or IID to IID. The result of a temporal select-IF is shown in Figure 7c where we select those Adults who were employed by IBM some time in the last 2 years. *The time period during which the specified condition must exist is* used to restrict the objects in the result as follows:

Figure 7 - Operator Examples

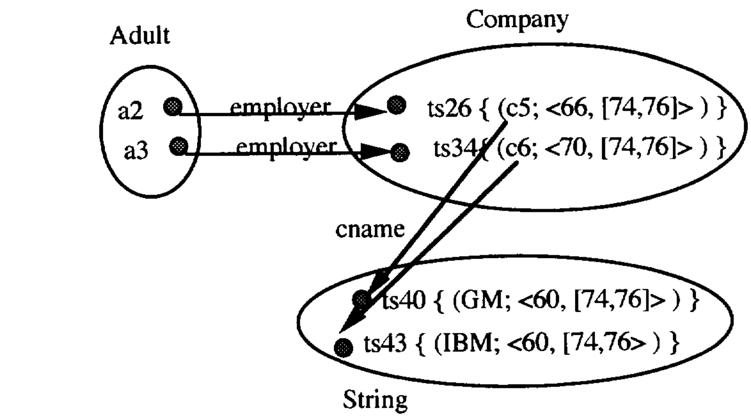

Adult
Company
a2
a3
employer
employer
ts26 { (c5; <66, [74,76]>) }
ts34{ (c6; <70, [74,76]>) }
cname
ts40 { (GM; <60, [74,76]>) }
ts43 { (IBM; <60, [74,76>) }
String

Figure 7a - Temporal Association

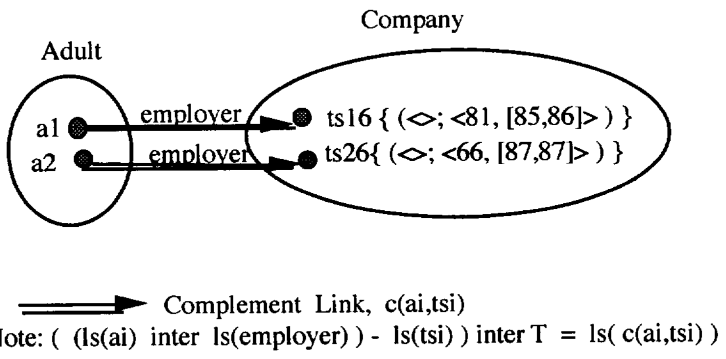

Company
Adult
a1
a2
employer
employer
ts16 { (<>; <81, [85,86]>) }
ts26{ (<>; <66, [87,87]>) }
Complement Link, c(ai,tsi)
Note: ((ls(ai) inter ls(employer)) - ls(tsi)) inter T = ls(c(ai,tsi))

Figure 7b - Temporal Complement

Figure 7 - Operator Examples

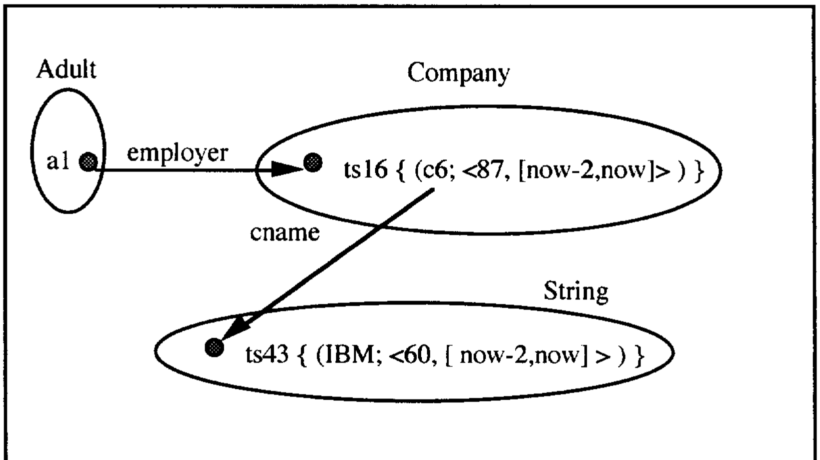

Adult
Company
a1
employer
ts16 { (c6; <87, [now-2,now]>) }
cname
String
ts43 { (IBM; <60, [now-2,now] >) }

Figure 7c- Select-IF

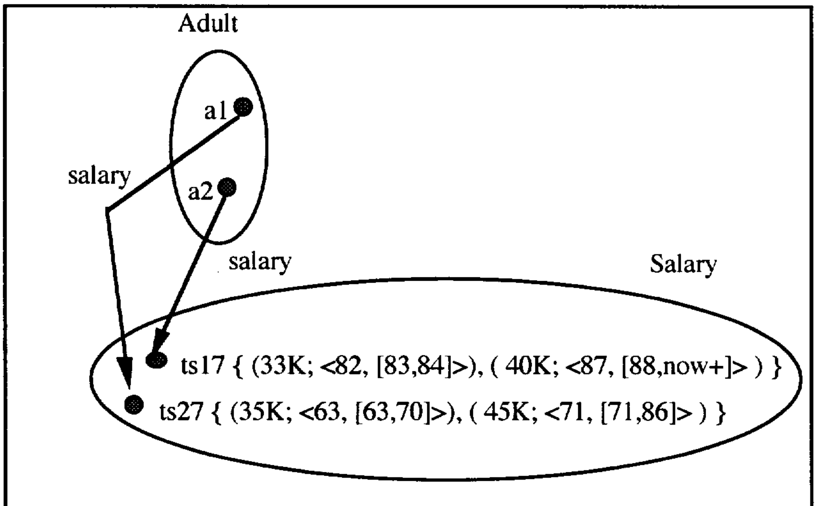

Adult
a1
salary
a2
salary
Salary
ts17 { (33K; <82, [83,84]>), (40K; <87, [88,now+]>) }
ts27 { (35K; <63, [63,70]>), (45K; <71, [71,86]>) }

Figure 7d - Select-When

$\sigma\text{-}IF_{[cname\,=IBM,\ exists,\ [now-2,now]}(\,Adult \bullet Company \bullet String\,)$

4) $\sigma\text{-}WHEN_{NTP}$, Temporal Selection Using a Non-Temporal Predicate

Select-WHEN is a unary operator which returns a set of objects that meet the non-temporal predicate condition specified by NTP. The lifespan of the resultant objects is the set of times when the object meets the NTP condition. The result of a temporal select-WHEN is shown in Figure 7d where we select the Adults who had a salary > 30K at sometime in their history as follows: $\sigma\text{-}WHEN_{[salary>30K]}\,(Adult)$

The lifespan of the resulting objects would be the times when they had salaries greater than 30K.

5) $\cap_T$, Temporal Intersection

$\cap_T$ is a binary operator which is similar to the join operator in a relational algebra. The result only exists if the graphs to be joined have at least one node class in common. The result of a temporal intersection over a set of classes at time T is given in Figure 7e where we get the current names of people who were guardians of children in 1980 and who are also currently classified as seniors. The set of classes in this case just contains the primitive class String.
$(Child \bullet^{guardians}_{1980} Person \bullet^{pname}_{now} String) \cap^{\{String\}}_{now} (Senior \bullet^{pname}_{now} String)$.
Note, we can associate Senior to class String through pname since Senior inherits this property from its supertype, Person.

6) $\cup_T$, Temporal Union

$\cup_T$ is a binary operator which is similar to the union operator in a relational algebra except it can operate over heterogeneous sets of objects. That is, the graphs in the operands need not have the same topology or nodes from the same classes. Only objects which exist in one operand or the other at time T will appear in the resulting set of graphs. In Figure 7f we find the persons who were not classified as seniors or deceased in 1980. This results in p1 and p2 with complement links to both sr1 and sr2 since neither had a link to senior or deceased in 1980. p3 will have a complement link to sr3 only since p3 was classified as deceased in 1980.

7) $-_T$, Temporal Difference

$-_T$ is a binary operator similar to the difference operator in a relational algebra except it can operate over heterogeneous sets of objects. Any objects which exist in the first operand at time T and contain an element of the second operand that exists in it at time T, will not appear in the result. In Figure 7g we find existing adults who were not part of the group of adults who were not linked to a Company in [85,87] by subtracting the result of 7b) from the set of adults. This gives us the adults who did not have a period of unemployment in their employment history. We express this operation as follows:
$Adult \; -^{\{Adult\}}_{now} (Adult \mid^{employer}_{[85,87]} Company)$

Figure 7 - Operator Examples

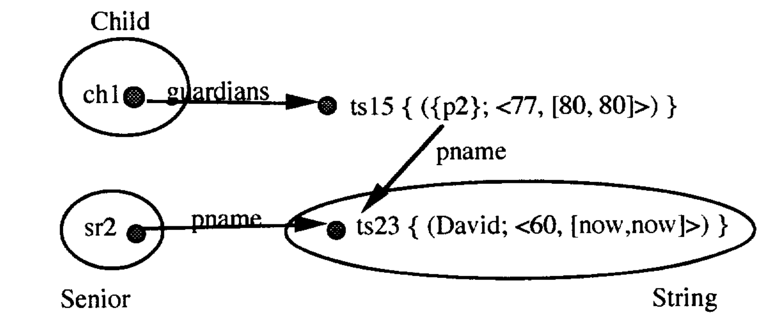

Child
ch1
guardians
ts15 { ({p2}; <77, [80, 80]>) }
pname
sr2
pname
ts23 { (David; <60, [now,now]>) }
Senior
String

Figure 7e - Temporal Intersection

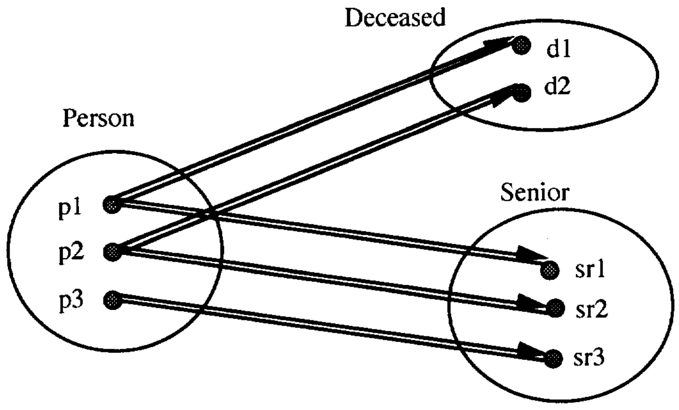


Figure 7f - Temporal Union

Figure 7 - Operator Examples

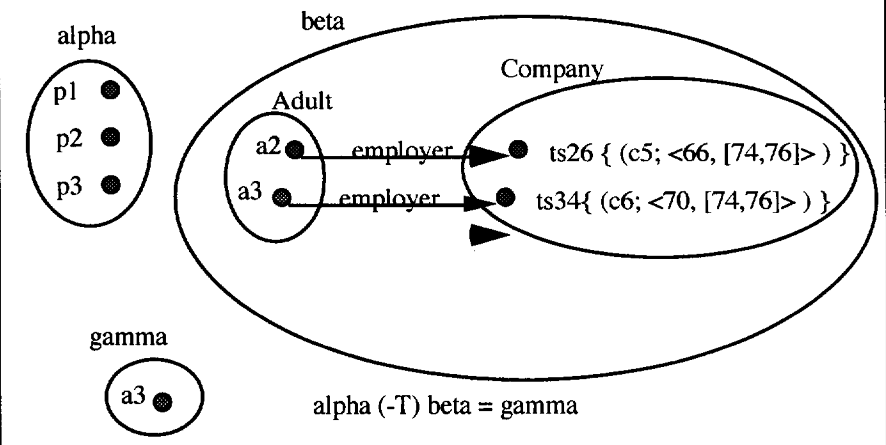


Figure 7g - Temporal Difference

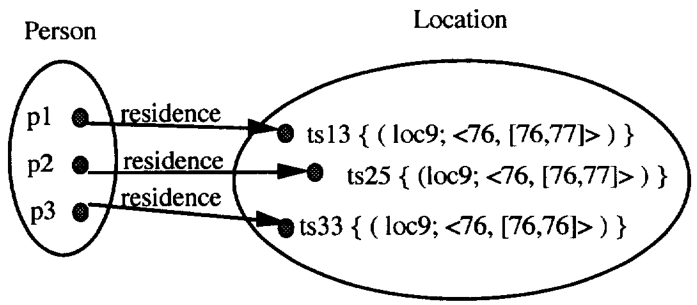

Person
Location
p1
p2
p3
residence
residence
residence
ts13 { (loc9; <76, [76,77]>) }
ts25 { (loc9; <76, [76,77]>) }
ts33 { (loc9; <76, [76,76]>) }

Figure 7h - Time Slice

8) *Time*$-$*Slice*$_T$

This operator reduces the objects in the time dimension just as the relational project reduces relations in the attribute dimension and the relational select reduces relations in the value dimension. It returns a snapshot of the object graph at time T. In Figure 7h we find each person's residence in T = [76,77] as follows: $Time{-}Slice^{\{Location\}}_{[76,77]}\,(Person\ \bullet^{residence}\ Location)$

9) *Rollback*$_T$

This operator produces a snapshot of the database as it existed at some previous transaction time T. Note that the previous operators used T with valid time semantics. In Figure 7i we get the history of dependents for each adult as recorded in the database in 1962. This operation is expressed as follows: $Rollback^{Person}_{1962}\ (Adult\ \bullet^{dependents}\ Person)$

Only the portion of the time-sequence where transaction time is less than or equal to T is retained in the result. No restrictions are placed on the relationship between tt and vt.

10) π_T, **Temporal Projection**

This operator is the counterpart of the relational project in that it returns objects which contain some subset of the object's properties. The difference is it returns a history of the values of each of those properties. For example, each person has four associated attributes in our example database and we may only want to return the history of their residence property if they have ever lived in Sunnyvale. The result of the following expression appears in Figure 7j: $\pi\ (Person\ \bullet^{residence}\ \sigma{-}IF_{[city\ =\ Sunnyvale,\ exists,]}\ (Location\ \bullet^{city}\ String)\ )\ [Location]$ where the temporal element is left out since we are requesting the entire history.

The unary operators have precedence over the binary operators and the ordering for the binary operators is: $\bullet_T$, $|_T$, $\bigcap_T$, $-_T$ and $\bigcup_T$. Parentheses can be used to create other orderings.

4.2. Set and List Structures

Set structures have the operations: $\in$, $\bigcap$, *minus*, $\bigcup$, *count*. List structures have the operators: *ith* $\in$, *sublist*, *concat* and *count*. Other aggregate operators such as max, min and *avg* can be added to collections of objects of type SET[T] or LIST[T] when T is a numeric type.

4.3. Time Sequence Structures

The time-sequence structure is used to model histories of attributes of complex objects. The operators defined in the time-sequence type are only briefly discussed here due to space limitations. These operators include: $\in$, *ith*, *merge*, restriction over a time-line or by values of the attributes in the time-sequence pairs and accumulation over time. These operators are described in [Segev & Shoshani 87]. We have also extended the select, aggregate, accumulate and restrict operators to handle multiple time-lines as discussed in Figure 2 where TL_i can be any time-line. For example, one could request the average salary of adults grouped on valid time with a granularity of year by sending the following message where Result is a keyword in the data manipulation language used to indicate a collection

Figure 7 - Operator Examples

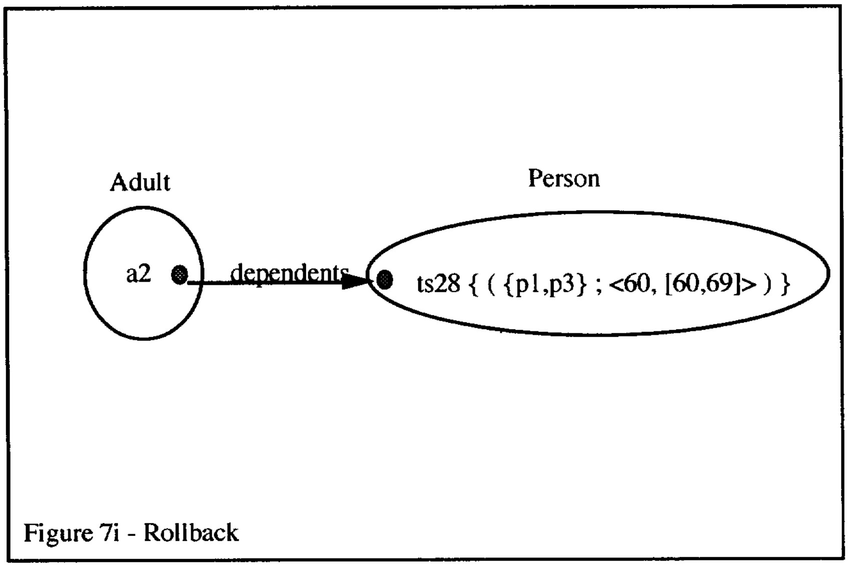


Figure 7i - Rollback

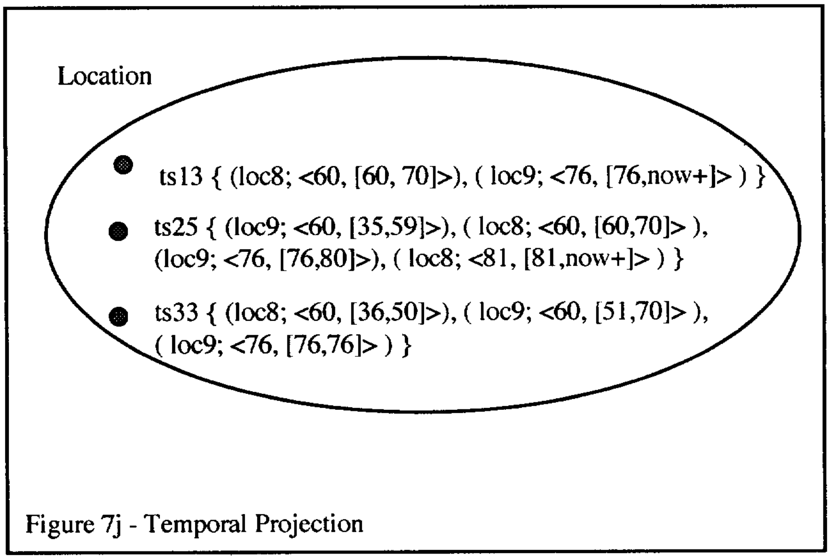

Location
ts13 { (loc8; <60, [60, 70]>), (loc9; <76, [76,now+]>) }
ts25 { (loc9; <60, [35,59]>), (loc8; <60, [60,70]>), (loc9; <76, [76,80]>), (loc8; <81, [81,now+]>) }
ts33 { (loc8; <60, [36,50]>), (loc9; <60, [51,70]>), (loc9; <76, [76,76]>) }

Figure 7j - Temporal Projection

where the result of the operation is stored: Result := Aggregate(AVG(salary),GRP(vt,year)) Similarly, one could request the average salary of adults grouped by transaction time with a granularity of year as follows: Result := Aggregate(AVG(salary),GRP(tt,year))

4.4. Update Operators

The following operators are defined in COMPLEX TYPE and are inherited by all user-defined types except subtypes of primitive types and collection types.

Add_Obj_Instance(o, A)

Adding an object o to the EXT(A) where A is the defining type of object o involves creation of an IID for the object where the class part of the IID is the identifier for type A. The user must supply the valid time interval for each property and transaction time will be the current time. A time-sequence object is created for each property of the new object.

Del_Obj_Instance(o, A)

Deleting an object o from the EXT(A) where A is the object's defining type involves ending the lifespans of all properties of the object. Since relationships are stored as properties their lifespan is also ended. The inverse attribute in the other type participating in the relationship will have the last (A,T) pair[4] in its time sequence which includes the IID of the deleted object adjusted in one of the following ways. If the time-sequence (TS) stores a set or list of values of type A then a new (A,T) pair is added to the TS which does not include the deleted object and the lifespan of the previous element is ended. If the TS object stores a single value the lifespan of the last (A,T) pair is ended.

Mod_Obj_Instance(o, A, [List of property:value])

Modification of an object o in the EXT(A) involves updating the TS associated with each property in the list. The end point of the previous (A,T) object's valid time interval will be set to the current time or a user-specified time and the start time of the new (A,T) object's valid time will be set to that time plus one time unit. The transaction time of the new object is set to the current time. Since relationships between this object and others are by reference and only involve its IID, no other updates are required. Any values in the list can not violate conditions in the set of constraints associated with object type A.

5. Schema Level Operators

The domain for the schema operators is the extension of the meta-data types. These classes contain instances of type definitions, variable definitions, operation (message) definitions and collection definitions. The meta-data classes can be queried using the operators defined at the object level since they are treated in the same way as the data. Therefore, we just define the update operators in this section. Operators for adding/deleting/renaming types, instance variables (attributes),

[4] (A,T) pair is used to refer to a time-sequence pair or element. TS refers to a time-sequence.

messages, and constraints are defined. Operators to add/delete a supertype from a type's list of supertypes, to modify an attribute's domain, parent or creator and to modify a message's parent, creator, parameters and code module are defined as well. We don't include transaction time in the operator's parameters since it is automatically recorded by the system.

1) Add_Type (Name, LIST[O_i], TUPLE[Meta-Vars], SET[Meta-Msgs], UserID, SET[BOOL], vt)

The user supplies a string-value for Name, a list of existing identifiers indicating the new type's supertype list his/her userid and the valid time when the type is allowed to be used. The valid time is restricted to be greater than or equal to the time when the type is recorded (tt) since a type can not have an extension until it exists in the database. Next, the user is asked to define the instance variables that will represent the structure of the object and the messages and constraints that will define its behavior. This operator results in the creation of an instance of Meta-Types, a set of instances of Meta-Vars, a set of instances of Meta-Msgs and a class object called SET[Name] to represent the extension of the new type.

2) Del_Type(OID, Name, UserID, END(vt))

The user supplies the name of the type to delete, his/her userid and the date when the deletion will become effective - END(vt). Names of types must be unique so the name can be used to find the OID of the type to be deleted. The lifespans of all properties associated with the instance of Meta-Types with this OID will be ended with the time END(vt). All the lifespans of the properties of the corresponding Meta-Vars and Meta-Msgs instances will be ended with the same time. These properties will no longer be inherited by the deleted type's subtypes as of END(vt). The supertype lists of the deleted type's subtypes must be updated to exclude the deleted type as of END(vt). This entails terminating the last element of the supertype list time-sequence at END(vt) and adding a new element that doesn't contain the deleted type whose valid start time is END(vt) + 1 where 1 has the same granularity as vt. The endpoint of the new interval is defined as now+. The lifespan of the extension of the deleted type, SET[Name] must also be ended at END(vt).

3) Rename_Type(OID, Old_Name,New_Name, UserID,vt)

Adds a new element to the time-sequence associated with the typ-name property of the Meta-Type instance whose typ-name = Old_Name and creator = UserID. The new element will have New_Name and UserId as its attribute values. It will become valid at time vt.

4) Add_Var(Type-OID, Name, Parent, UserID, Domain, SET[BOOL], vt)

The user supplies the OID of the type to which the variable is to be added, the name of the variable, the parent it is inherited from, his/her userid, the domain type from which the variable will get its values, constraints on the variable and the time when the definition becomes valid. This results in the creation of a new instance of Meta-Vars and adds an element to the time-sequence associated with the instance of Meta-Types that has the given OID.

5) Del_Var(Type-OID, Var-OID, vt)

The lifespan of the last element in the time-sequence associated with the var-list property of the Meta-Type instance with Type-OID as its identifier will be ended at vt. A new element will be started which does not contain the OID of the deleted instance variable. The lifespans of all properties of the instance of Meta-Vars with the identifier Var-OID will also be ended at time vt. If the variable is an inverse, the same procedure must be done for the instance of Meta-Vars representing the inverse variable.

6) Add_Msg(Type-OID, Name, Parent, UserID, SET[Param], Code, vt, SET[BOOL])

This operator creates an instance of Meta-Msgs using the user-supplied values. Pre- and Post-condition constraints for the use of the messages can also be specified as a set of boolean conditions. The last element of the time sequence associated with the msg-list property of the instance of meta-types identified by Type-OID has its lifespan ended at vt and a new element which includes the new message is added to the sequence as in the previous add operations. Parameters in the new message are restricted to represent existing variables during the message's lifespan.

7) Del_Msg(Type-OID, Msg-OID, vt)

The lifespan of the last element in the time-sequence associated with the msg-list property of the Meta-Type instance with Type-OID as its identifier will be ended at vt. A new element will be started which does not contain the OID of the deleted instance variable. The lifespans of all properties of the instance of Meta-Msgs with the identifier Msg-OID will also be ended at time vt.

8) Add_Bool(Type-OID,{Var-OID| Msg-ID| Coll-ID}, BOOL, vt)

This operator creates an instance of type Boolean using the expression, BOOL, supplied by the user. The last element of the time sequence associated with the constraints property of the instance of meta-types identified by Type-OID has its lifespan ended at vt and a new element which includes the new constraint is added to the sequence as in the previous add operations. If a variable, message or collection OID is specified as the second parameter, the previous operation is carried out on the constraint (pre-/post-conditions) property of the appropriate instance.

9) Del_Bool(Type-OID,{Var-OID| Msg-ID| Coll-ID}, BOOL-OID, vt)

The lifespan of the last element in the time-sequence associated with the constraint (pre-/post-condition) property of the specified instance will be ended at vt. A new element will be started which does not contain the OID of the deleted constraint.

10) Add_SuperT($Type_i$, $Type_j$, position, vt)

This operator says to make $Type_j$ a supertype of $Type_i$ and to add $Type_j$ to the supertype list of $Type_i$ at position. This operation entails ending the lifespan of the last element in the time-sequence associated with the superclasses property *of $Type_i$ at time vt and adding a new element which includes the new supertype.*

$Type_j$

and $Type_i$ must both be pre-defined in order to perform this operation.

11) Del_SuperT($Type_i$, $Type_j$, **position, vt)**

This operator says to delete $Type_j$ from the supertype list of $Type_i$ at position. This operation entails ending the lifespan of the last element in the time-sequence associated with the superclasses property of $Type_i$ at time vt and adding a new element which excludes the supertype to be deleted. $Type_j$ and $Type_i$ must both be pre-defined in order to perform this operation.

12) Mod_Var_Signature(Var-OID, Name, Parent, UserID, Domain, vt)

This operator lets the user change the name, parent, userid and/or domain associated with the variable identified by Var-OID. The last element of the time sequence associated with the var-signature property of the instance of meta-vars identified by Var-OID has its lifespan ended at vt and a new element which includes the new signature values is added to the sequence.

13) Mod_Msg_Signature(Msg-OID, Name, Parent, UserID, SET[Param], Code, vt)

This operator lets the user change the name, parent, userid parameters and/or code associated with the message identified by Msg-OID. The last element of the time sequence associated with the msg-signature property of the instance of meta-msgs identified by Msg-OID has its lifespan ended at vt and a new element which includes the new signature values is added to the sequence. Parameters in the new message are restricted to represent existing variables during the message's lifespan.

6. Summary and Future Work

Unlike the three temporal object-oriented models described in the introduction, TOODM supports type definition histories and makes use of the time-sequence construct shown in Figure 2. [Wuu & Dayal 91] and TOODM use attribute time-stamping whereas the other two papers time-stamp at the object instance level. TOODM allows a lifespan to be a finite union of non-contiguous intervals whereas [Wuu & Dayal 91] require contiguous intervals which does not allow for the case where a particular attribute may have been part of a type definition, then removed from it and later added back. [Su & Chen 91] do not directly support tt in their data model, rather they require the user to define rules to handle tt. Neither [Wuu & Dayal 91], [Su & Chen 91] or [Kafer & Schoning 92] include a discussion of a temporal object-oriented algebera with their data model.

In this paper, we developed a temporal, object-oriented algebra (called TOOA) for the manipulation of the intension and extension of a temporal, object-oriented data model. The time-sequence collection type was used to model histories of attribute values of objects and histories of the set of properties defining an object type. The paper makes the following contributions. First, it provides an algebra that operates on collections of object graphs. Second, it provides sets of update operators for both object histories and type definition histories. Third, it provides a merge operator for time-sequence objects to filter in or leave out corrections. The algebra includes a rollback operator to allow reversion back to a previous state of the recorded model as well as operators that retrieve values based on when they were valid. The separation of recorded values from their corrections also allows us to maintain an audit trail which could prove useful in accounting applications and in duplicating the results of previous analyses. Our operators also allow us to specify a different time period for each property of an object through the use of the associate/non-associate operators.

The next step in the development of this model will be to map the algebra to a higher-level, SQL-like syntax. Formal proofs to establish the equivalence properties will be used to determine transformations that will optimize query processing. Constraints are also being developed for the model to provide a means of doing semantic query optimization by transforming the query into a syntactically different but semantically same query which may have a more efficient execution plan. Implementing the optimizer as a rule-based system would make it more extensible, providing a means for adding new information and optimization methods. We will also need to investigate the trade-offs between optimizing the time to execute a query and the time to select the best query execution plan. Determining access patterns and indexes to facilitate those patterns will also be investigated through the development of a graphical front-end to the POSTGRES extended relational system.

REFERENCES

[Alashqur etal 89] A.M. Alashqur, S.Y.W. Su and H. Lam, OQL: A Query Language for Manipulating Object-Oriented Databases, *Proceedings of the 5th International Conference on Very Large Data Bases*, Amsterdam, The Netherlands, 1989, pp. 433-442.

[Andany etal 91] J. Andany, M. Leonard and C. Palisser, Management of Schema Evolution in Databases, *Proceedings of the 17th International Conference on Very Large Data Bases*, Barcelona, Spain, September 1991, pp. 161-170.

[Ariav 86] G. Ariav, A Temporally Oriented Data Model, *ACM Transactions on Database Systems*, V. 11, N. 4, December 1986, pp. 499-527.

[Banerjee etal 88] J. Banerjee, W. Kim, and K.C. Kim, Queries in Object-Oriented Databases, *Proceedings of the 4th International Conference on Data Engineering*, Los Angeles, California, February 1988, pp. 31-38.

[Clifford & Warren 83] J. Clifford and D.S. Warren, Formal Semantics for Time in Databases, *ACM Transactions on Database Systems*, V.8, N. 2, June 1983, pp. 214-254.

[Carey etal 88] M.J. Carey, D.J. DeWitt and S.L. Vandenberg, A Data Model and Query Language for EXODUS, *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, June 1988, pp. 413-423.

[Clifford & Croker 87] J. Clifford and A. Croker, The Historical Data Model (HRDM) and Algebra Based on Lifespans, *Proceedings of the 3rd*

International Conference on Data Engineering, Los Angeles, California, February 1987, pp. 528-537.

[Elmasri & Wuu 90] R. Elmasri and G.T.J. Wuu, A Temporal Model and Query Language for ER Databases, *Proceedings of the 6th International Conference on Data Engineering,* May 1990, pp. 76-83.

[Ferg 85] S. Ferg, Modeling the Time Dimension in an Entity-Relationship Diagram, *Proceedings of the 4th International Conference on the ER Approach, In Entity-Relationship Approach,* Ed. Chen, P.P.S., Elsevier Science Publishers B.V. North-Holland, 1985, pp. 280-286.

[Gadia 88a] S.K. Gadia, The Role of Temporal Elements in Temporal Databases, *Database Engineering,* V 7, 1988, pp. 197-203.

[Gadia 88b] S.K. Gadia, A Homogeneous Relational Model and Query Language for Temporal Databases, *ACM Transactions on Database Systems* V. 13, N. 4, December 1988, pp. 418-448.

[Gadia & Yeung 88] S.K. Gadia and C.S. Yeung, A Generalized Model for a Relational Temporal Database, *Proceedings of ACM SIGMOD International Conference on the Management of Data,* V. 17, N. 3, June 1988, pp. 251-259.

[Guo etal 91] M. Guo, S.Y.W. Su and H. Lam, An Association-Algebra for Processing Object-Oriented Databases, *Proceedings of the 7th International Conference on Data Engineering,* Kobe, Japan, April 1991, pp. 23-32.

[Kafer & Schoning 92] W. Kafer and and H. Schoning, Realizing a Temporal Complex-Object Data Model, to appear in *Proceedings of the ACM SIGMOD International Conference on the Management of Data,* San Diego, California, June 1992.

[Kim & Chou 88] W. Kim and H.T. Chou, Versions of Schema for Object-Oriented Databases, *Proceedings of the 14th International Conference on Very Large Data Bases,* Los Angeles, California, 1988, pp. 148-159.

[Klopprogge & Lockemann 83] M.R. Klopprogge and P.C. Lockemann, Modeling Information Preserving Databases: Consequences of the Concept of Time, *Proceedings of the 9th International Conference on Very Large Data Bases,* Florence, Italy, 1983, pp. 399-416.

[McKenzie & Snodgrass 91] E. McKenzie and R. Snodgrass, Evaluation of Relational Algebras Incorporating the Time Dimension in Databases, *ACM Computing Surveys,* V. 23, N. 4, December 1991, pp. 501-543.

[Navathe & Ahmed 89] S.B. Navathe and R. Ahmed, A Temporal Relational Model and Query Language, *Information Sciences,* V. 49, 1989, pp. 147-175.

[Rose & Segev 91] E. Rose and A. Segev, TOODM - A Temporal, Object-Oriented Data Model with Temporal Constraints, *Proceedings of the 10th International Conference on the Entity-Relationship Approach,* San Mateo, California, 1991, pp. 205-229.

[Segev & Shoshani 87] A. Segev and A. Shoshani, Logical Modeling of Temporal Databases, *Proceedings of ACM SIGMOD International Conference on the Management of Data,* May 1987, pp. 454-466.

[Shaw & Zdonic 90] G.M. Shaw and S.B. Zdonic, A Query Algebra for Object-Oriented Databases, *Proceedings of the 6th International Conference on Data Engineering,* Vol. 12, No. 3, February 1990, pp. 154-162

[Skarra & Zdonik 86] A.H. Skarra and S.B. Zdonik, The Management of Changing Types in an Object-Oriented Database, *Procceedings of the OOPSLA Conference,* Portland, Oregon, September 1986, pp. 483-495.

[Soo 91] M.D. Soo, Bibliography on Temporal Databases, *SIGMOD Record,* V. 20, N. 1, March 1991, pp. 14-23.

[Snodgrass & Ahn 85] R. Snodgrass and I. Ahn, A Taxonomy of Time in Databases, *Proceedings of ACM SIGMOD International Conference on the Management of Data,* May 1985, pp. 236-246.

[Snodgrass 87] R. Snodgrass, The Temporal Query Language TQUEL, *ACM Transactions on Database Systems,* V. 12, N. 2, June 1987, pp. 247-298.

[Su & Chen 91] S.Y.W. Su and H.M. Chen, A Temporal Knowledge Representation Model OSAM*/T and Its Query Language OQL/T, *Proceedings of the 17th International Conference on Very Large Data Bases,* Barcelona, Spain, September 1991, pp. 431-442.

[Tuzhilin & Clifford 90] A. Tuzhilin and J. Clifford, A Temporal Relational Algebra as a Basis for Temporal Relational Completeness, *Proceedings of the 16th International Conference on Very Large Data Bases,* 1990, pp. 13-23.

[Wuu & Dayal 91] G. Wuu and U. Dayal, A Uniform Model for Temporal Object-Oriented Databases, *Proceedings of the 8th International Conference on Data Engineering* February 1991, pp. 584-593.

[Zicari 91] R. Zicari, A Framework for Schema Updates In An Object-Oriented Database System, *Proceedings of the 7th International Conference on Data Engineering,* Kobe, Japan, April 1991, pp. 2-13.