A Temporal Knowledge Representation Model OSAM*/T and Its Query Language OQL/T [1]

Stanley Y. W. Su
Electrical Engineering Dept.
and Computer & Information Science Dept.
Database Systems Research
& Development Center, CSE470
University of Florida, Gainesville, FL 32611

Hsin-Hsing M. Chen
Electrical Engineering Dept.
Database Systems Research
& Development Center, CSE470
University of Florida,
Gainesville, FL 32611

Abstract

The importance of temporal database management and temporal reasoning has long been recognized by the database and AI communities. Many techniques for modeling and managing temporal databases have been introduced. Most of the existing research on temporal databases have been based on the relational data model which has limited capabilities in modeling complex objects, constraints, and behavioral properties. In this paper, we present an object-oriented knowledge-based approach to model and process temporal information. A temporal knowledge representation model OSAM*/T is described. It is an extension of the object-oriented semantic association model OSAM* to capture the concepts of time and history. Object time-stamping is used to record the evolutions of objects, and knowledge rules instead of extra time notions are used to capture special temporal requirements. The temporal model is also featured by its separation of historical data and current data and its simple and maintainable representation of complex objects. In addition to the temporal data model, this paper also introduces a temporal query language OQL/T which is an extension of the object-oriented query language OQL for processing temporal information. Several temporal functions, interval comparison operators, and set operators are described.

1. Introduction

Temporal information is the recording of past activities which are often important to present and future decision-making. Temporal information can be used not only to answer such queries as "What was John's title when Mary was a typist?" and "How many times has John's title been changed during 1976 to 1982?", but also as the basic assertions for a knowledge base management system to reason about past activities and to derive facts which are not explicitly stored in a knowledge base [BOL82, ALL83, McK86, SNO86, DE87].

In recent years, the technology in storage media has made a remarkable progress and has allowed large quantities of temporal data to be stored and processed efficiently [SNO86]. This has motivated a considerable amount of research on temporal databases. They include the extension of data models for modeling temporal data [AND82, CLI83, KLO83, SCH83, LUM84, CLI85, SNO85, ADI86, ARI86, GAD86, TAN86, SEG87, LOR88, NAV89, TAN89, ELM90c], the extension of query languages and relational algebra for accessing temporal data [KLO83, CLI85, GAD86, TAN86, SNO87, LOR88, NAV89, TAN89, ELM90a], and the management of secondary storage to achieve efficient indexing techniques for data access [LUM84, SHO86, AHN86, AHN88, ELM90b, ELM90d]. Most of these efforts have been based on the relational model directly or indirectly because of its well developed theories.

Techniques used to model temporal information using the relational model are attribute time-stamping [KLO83, CLI85, GAD86, TAN86, ELM90a] and tuple time-stamping [LUM84, SNO85, LOR88]. The expansion and contraction of data between the normalized and non-normalized structures required in the attribute time-stamping technique produce a serious time overhead in processing temporal data; whereas, in tuple time-stamping, data-redundancy is the major shortcoming. In order to take advantages

1. This research is partially supported by the National Science Foundation Grant #DMC-8814989 and the Institute of Standards and Technology Grant #60NANB4D0017. The Implementation effort is supported by the Florida High Technology and Industry Council Grant #UPN 85100316.

of these two techniques and to avoid their shortcomings, Navathe and Ahmed [NAV89] used a combined approach which is based on the concept of "temporal normalization". However, their approach only works under a weak assumption called "synchronous attributes", which is a rare case in the real world application. If "synchronous attributes" are not available in an application, the approach will be degenerated to tuple time-stamping over relations which contain only two attributes (i.e., the primary key and a variant attribute). This logical data representation will be worse than using the tuple time-stamping technique over non-binary relations because it generates more redundant data (i.e., replication of keys) and needs more join operations during data processing. In addition to the problems associated with the existing techniques, there are two common problems found in a relational temporal database: the need for more than two time notions and the mixture of historical data with current data. We shall elaborate on these problems in this paper.

Motivated by the above problems of the current time-stamping techniques and the use of the relational model for modeling temporal information, we have taken an object-oriented, knowledge-based approach to model temporal information.• This approach takes advantages of the features of object-oriented databases [BOO86] and the concepts and techniques of integrating artificial intelligence and database techniques [DAY85, SU85, STO86, RAS88].

In this work, we extended the object-oriented semantic association model OSAM* [SU89] and its query language OQL [ALA89] to model and query temporal databases. The resulting model and language are called OSAM*/T (temporal OSAM*) and OQL/T (temporal OQL), respectively. They have the following key features:

1. An object time-stamping technique is used which is a compromise between the attribute time-stamping and the tuple time-stamping techniques. The history of objects, object instances, and object associations are preserved using this technique.

2. Data redundancy is avoided by adopting the "delta file" concept [ROC75, SEV76] for object management.

3. Start-time and End-time are the only two time notions employed in OSAM*/T for recording object histories. Other time notions [LUM84, SNO85] are captured by temporal rules.

4. Historical data and current data are separated both logically and physically. This separation imposes an implicit index on historical and current data, and thus can avoid unnecessary search and sorting during query evaluation.

5. Hierarchically structured object instances are allowed, thus, simplifying the representation and maintenance of the histories of complex objects.

6. In addition to the powerful association operators introduced in OQL, the query language OQL/T provides a set of temporal functions, interval comparison operators, and temporal set operators for manipulating temporal data.

This paper is organized as follows. Section 2 gives an overview of OSAM*/T with an emphasis on temporal concepts such as time sequence, event, object time-stamping, object history, object instance history, and association history. Section 3 describes OQL/T. The semantics of a general query structure is discussed in detail with examples. In addition, interval comparison operators, temporal functions, and temporal set operators are also presented. A summary and a conclusion are given in Section 4.

2. Modeling Temporal Database Using OSAM*/T

2.1 Basic Concept of Time and Event

A time sequence is a series of continuous time points t1, t2, ... tn-1, tn, where t1 < t2 < ... < tn. The distance between any two adjacent time points is identical and is called a time unit. The time unit chosen for a temporal system may be different from application to application depending on the granularity needed. It can be microsecond, minisecond, second, minute, hour, day, month, or ..., etc. A system may use more than one time granularity for its applications' needs by providing functions for the conversion between different time granularities. In the rest of this paper, we shall use "day" as the time granularity in our examples and assume that the history of object instance is recorded discretely even though the interpretation of an object history is continuous.

In this paper, an event is an action which will cause a change of the contents of a database. An update, a delete, or an insert operation is an event. A data retrieval operation is not considered as an event because it does not alter the state of a database and, thus, does not affect the time tags associated with the data.

In a temporal database, an object instance will have a new time tag and new attribute values when an event occurs (note: in case of a delete operation, the new values are nulls). The old instance then becomes a part of the history of the instance. For example, to adjust John's salary to $30,000 beginning on November 10th, 1989 will introduce a new instance to replace the old one in the current database and the old instance is shifted into the historical area:

<11-10-89, - , OID, 448, John, $30,000>...New Instance.
<10-01-81, 11-9-89 , OID, 448, John, $20,000>...Old Instance.

Here, the first two fields represent Start-time and

End-time, "-" stands for the present time, and OID stands for the object identifier.

2.2 Time Notions Used to Model Temporal Information

In OSAM*/T, Start-time and End-time are the only two time notions used to delimit the records of an object history. Start-time is the time when a new object (or an object's instance) becomes active in a database (or in a class). For example, the operations of creating an object, inserting an object instance, and updating an object instance will cause a new Start-time to be attached to the new object instance which has new attribute values. The End-time is the time when an object (or its instance) becomes inactive in a database (or a class). For example, the operations of destroying an object in a database, deleting an object instance from a class, and updating an object instance in a class will cause an End-time to be attached to the inactive object or object instance. The difference between object and object instance will be explained later.

2.3 Using Knowledge Rules to Capture Special Notions of Time

It has been suggested in [LUM84, SNO85, NAV89] that extra time notions in addition to the Start-time and End-time be introduced to capture some application-specific temporal requirements. They include transaction time, effect time, physical time, logical time, or any other user-defined time notions. The disadvantage of adding these extra time notions into a temporal database is the added storage requirements for storing the extra time tags. It is important to keep the number of time notions small because, once a general time notion is added into the database, extra time tags need to be used in every object instance in every object class. We observe that some time notions are applicable only to some specific data. For example, the fact that "the employee Mary's salary, $30K, is retroactively effective from 12-15-87 instead of from 12-15-88" only affects the object instance Mary, and it should not be treated as a general case.

In OSAM*/T, we use only two general time notions: Start-time and End-time. The other time notions are expressed by knowledge rules. We shall use the retroactive update problem as an example to show the advantage of using knowledge rules over extra time notions.

Approach 1: Introduce "Record-time" time notion to capture the fact that Mary's salary of $30K has been retroactively effective on 12-15-87. The original data base and the updated database are given below:

(1) Original database:

Name	Title	Salary	Start-time	End-time	Record-time
<Mary	Secretary	$20K	12-15-85	12-14-88	12-14-88>
<Mary	Secretary	$30K	12-15-88	-	12-15-88>

(2) Updated database:

Name	Title	Salary	Start-time	End-time	Record-time
<Mary	Secretary	$20K	12-15-85	12-14-88	12-14-88>
<Mary	Secretary	$30K	12-15-87	12-14-88	02-14-90>
<Mary	Secretary	$30K	12-15-88	-	12-15-88>

In this approach, the extra time notion "Record-time" is used to record (on 02-14-90) the fact that Mary's salary is retroactively updated. This approach introduces excessive storage requirements since every object instance in the database will need a Record-time tag.

Approach 2: Use the following knowledge rule to capture this specific temporal requirement associated with the object instance Mary.

```
Rule 105
T [02-14-90, - ]
Trigger-cond (Before Retrieve(Employee))
IF WHEN T[12-15-87, 12-14-88]
   CONTEXT Employee [Name = Mary]
THEN Salary = $30K
End
```

In this approach, rule 105 becomes active on 02-14-90 (rules can have history). It is automatically triggered before any retrieval operation is performed on Employee instances. The When and Context expressions in the IF clause specify that if the historical instances of Mary during the period between 12-15-87 and 12-14-88 are accessed, her salary should be $30K instead of the salary recorded in the database. This approach allows many special temporal requirements to be captured by the model and enforced by a DBMS/KBMS without the excessive storage requirement and I/O for data-access of the first approach. Although the run-time execution of rules will increase the query response time, it however can drastically increase the functionalities of the system in terms of managing temporal data and constraints. Rules are only triggered when their associated data are accessed or manipulated. They can be precompiled into some internal form for run-time binding of variables [SIN90, CHU90]. Therefore, they should not significantly degrade the overall performance of a temporal KBMS. Knowledge rule specification and management are the key features of the OSAM* model [SU89, SU91] and the implemented prototype KBMS [LAM89, SIN90, CHU90]. Its extension to temporal rule specification and management is described in another paper [CHE91].

2.4 Object Time-Stamping in OSAM*/T -- A Compromised Approach.

The object time-stamping technique used in this work is a compromise between the tuple time-stamping approach and the attribute time-stamping approach. It aims to reduce data redundancy and, at the same time, avoid excessive overhead in processing time. Different from the combined approach used in [NAV89], time stamps are assigned to a set of semantically related attributes which form instances of an object class instead of the so-called "synchronous attributes". As pointed out in the introduction, the absence of synchronous attributes in a relation will cause the approach in [NAV89] to be degenerated to tuple time-stamping on many binary relations.

In our approach, every object instance is time stamped with a time interval, <Start-time, End-time>. The current database is a snapshot database of the "current" time which contains the most recently updated object instances of all the classes. If an instance is in the current database, it is said to be currently active; otherwise, it is currently inactive. Active object instances have the "-" mark as its End-time. When an event takes place that affects an object instance, its End-time will be set to the time of the event, and the object instance becomes an inactive historical instance which is shifted into a historical area. A new active instance is then created whose Start-time is set one time unit after the time of the event and whose End-time is set to "-".

The proposed time-stamping technique makes both logical and physical separations between current data and historical data which, we believe, is an advantage over the relational approach in which current and historical data are mixed. This is because if a user is interested only in the current information, the DBMS/KBMS only has to search the current database without having to separate the current data from the historical data in query processing.

2.5 Using the "Delta File" Concept in Object Management

In order to avoid data redundancy in the object time-stamping approach, we adopt the "delta file" concept [ROC75, SEV76] for object management and introduce the symbol "#" to mark those unchanged attribute values during object evolution. When a new object instance is created due to an event, all the attributes which are not affected by this event will be marked by "#" instead of storing the unchanged values in the historical object instance. The current object instance will always keep a complete copy of the current values of these attributes. For example, the adjustment of John's salary discussed in Section 2.1 will result in the new and old object instances shown below:

<11-10-89, - , OID, 448-44-6050, John, $30,000> .. New
<10-01-81, 11-9-89 , # , # , # , $20,000> .. Old

Using the "#" mark to represent unchanged attribute values can avoid data redundancy during object evolution. This is particularly important when we are dealing with the history of complex objects. This approach of relating current and historical data will not complicate the search for historical data since, when a historical object instance is needed, the system will search from the current object instance, which carries a complete copy of the current attribute values, to the historical data area. These values will be kept during the search until (1) some values other than "#" marks have been found in the historical data associated with the attributes and (2) the needed historical object instance is found. In the first case, the corresponding attribute values kept by the system are replaced by the historical data and, in the second case, the system will stop searching and use the values kept by the system as the answer.

The above example shows the effect to time tags when data are updated. Other events such as insertion and deletion will also cause the change of time tags and the creation of new instances. In the case of deleting an object instance, the new instance will contain null values.

2.6 Object instance history

OSAM*/T uses a distributed storage model (DSM) instead of the more common static storage model (SSM) for representing the structural properties of objects. In DSM, an object can appear in many classes. The descriptive data about an object are distributed in these classes. An instance is the representation of an object in a specified class and contains the attribute values that characterize the object. An instance is uniquely identified by an instance identification (IID) which is the concatenation of a class ID and an object ID. When any of these attribute values is modified, a history of the object instance is created. If an object participates in more than one class, each object instance in each class will have its own history. This storage model is different from that used in C++, Smalltalk, ONTOS, etc. in which an object can be a member of only one class and all inherited attributes appear in its class at the bottom of the class hierarchy or lattice. A discussion on the relative advantages of these two approaches is out of the scope of this paper. Here, we simply note that time stamps are associated with object instances instead of objects. Data in an instance can be independently accessed and manipulated.

There are three temporal events: insertion, update,

and deletion. When an object initially participates in a class, an instance of that object will be inserted into the class and an object instance history in that class will be started. Subsequent events will add data to its history. Figures 1(a), (b), (c), and (d) give an illustration of object instances in the Employee class as Mary's instance is created, updated and deleted.

2.7 Object History

Object history can be viewed as the collection of the histories of its instances in the classes it participates. Deletion of its instance from a class only means the withdrawal of its participation as an active instance in that class and does not affect its other instances. When an object is created, it must participate in at least one class in the schema. This participation starts the object history as well as the object instance history in that class. If the object participates in only one class, then the object instance history is the same as the object history. When an object is destroyed (or deleted permanently from a database), all the new instances of this object will have nulls as their attribute values.

2.8 Association Histories in OSAM*/T

In OSAM*/T, object classes are defined in terms of their associations with other classes. Five system predefined association types (Aggregation, Generalization, Interaction, Composition, and Crossproduct) are provided for the convenience of database designers and users to specify different semantic relationships or associations among classes and their instances. The semantic distinctions among the association types are captured by knowledge rules which control the manipulations of object instances having the association types. In our KBMS implementation, association types are defined as classes which are subclasses of the Association class. Thus, new association types or subtypes of the existing association types can be introduced by defining subclasses under the Association class or its subclasses. It has also been recognized in [ELM90a] that, beside object history and object instance history, the history of object associations needs to be maintained. One of the advantages of object time-stamping is that association histories among object instances can be derived from object instance histories. Whenever an event occurs to an object instance, the corresponding association history can be inferred from that instance history. We shall use the Interaction association as an example to illustrate this point.

Interaction is an association type used to model some relationships between objects in two or more classes and the relationships themselves are treated as instances of an object class. An object in a defining class which has an interaction association with other classes (called constituent classes) represents a fact that relates the objects of its constituent classes. For example, the fact that specific employee works on a specific project forms an object of a defining class Work-on as modeled in Figure 2(a). Since it is an object, the fact is assigned with an IID. Since associations are themselves objects, the tracking of the history of an interaction association is the same as that of other objects. For example, if employee Mary has been assigned to project P1, an interaction object instance of Works_on will be created and be assigned with an IID, W1. This object instance will consist of Mary and P1's IIDs in addition to its own IID W1. Any change on either Mary's attributes or P1's attributes will not affect this interaction association. However, a deletion of either Mary's instance or P1's instance from Employee or Project class or Mary's withdrawal from P1 will cause the interaction instance to be deleted (referential constraint). In either case, nulls will replace Mary and P1's IIDs in W1's new version as shown in Figure 2(b). In W1's history, the nulls in the historical record with the time interval between t2 and t3-1 indicate that the interaction association between Mary and P1 does not exist. From t3 on, Mary has been working on project P1 again. This is represented by storing Mary and P1's IIDs in the object instance W1. The management of the histories of other association types can be similarly handled.

2.9 Operations for a DBA to Correct Errors

In a temporal database, errors in object history due to careless data entry and update are unavoidable and should be corrected. Corrections of these errors should be allowed without treating these operations as temporal events. For example, mistakes can be made in entering employee's salary $27K for $37K and in making a $10K salary increase for no increase. For the first case, the record should be corrected; whereas, for the second case, the historical record should be deleted. CorObj and DelObj are the two operators provided in OSAM*/T for a DBA to deal with the above cases.

3. The Query Language OQL/T

OQL/T is the query language for processing temporal data/knowledge bases modeled in OSAM*/T. It is an extension of the query language OQL [ALA89] to contain temporal constructs and functions for specifying temporal conditions. The general structure of a query in OQL/T is shown below:

```
WHEN Time_Interval
 WHERE Interval Comparison Expression

CONTEXT Association Pattern Expression
 WHERE conditions
 SELECT object classes and/or attributes
OPERATION(s) object class(es)
```

Query structure in OQL/T

The main difference between the query structures of OQL/T and OQL is that the query structure of OQL/T contains an additional temporal condition (the WHEN clause) which specifies the time interval of a snapshot temporal database. The snapshot contains all the recorded data that fall in that time interval. If an interval comparison expression is specified in the optional WHERE subclause, the interval has to satisfy the expression. Otherwise, the query will not be processed. The CONTEXT clause specifies the pattern of object associations that objects in some referenced classes of the snapshot database should have (specified by the association pattern expression), the inter-class attribute conditions or aggregation conditions of these objects (specified by the optional WHERE subclause), and the selected objects and attributes (specified by the optional Select subclause). The selected objects and attributes are then processed by the system-predefined and/or user-defined operations specified by the OPERATION clause. In the following subsections, we shall explain these language constructs further by examples. The emphasis of the following sections will be on the WHEN clause. More complex examples on the use of the CONTEXT and OPERATION clauses have been presented in [ALA89].

3.1 The WHEN Clause

The WHEN clause is used to specify the snapshot temporal database of interest. It is an optional clause. If it is not specified in a query, the current database is assumed in the processing of the query. A time interval can be specified in this clause either by a specific time reference or by a data reference. In the former case, the keyword WHEN is followed by an explicit time interval specification of the form "T[A, B]", where A and B are two time points and A is less than or equal to B. If A is equal to B, the time reference is called time-point reference and "WHEN T[A, B]" can be replaced by "AT A"; if A is less than B, the time reference is called a time-interval reference. In both cases, the query will be evaluated against all the data in the snapshot database between time A and time B. Examples 1 and 2 illustrate the uses of time-interval and time-point references, respectively. TCOUNT is a temporal function which counts the number of times that title has been changed.

```
Q1: How many times has Mary's title been changed
during the period between 1983 and 1986?

WHEN T[01-01-83, 12-31-86]
CONTEXT EMPLOYEE [Name = Mary]
RETRIEVE TCOUNT(Title)
```

Example 1: Time-interval reference in a OQL/T query.

```
Q2: What is John's title on 11-29-84?

WHEN T[11-29-84, 11-29-84]
CONTEXT EMPLOYEE [Name = John]
RETRIEVE EMPLOYEE.Title
or
AT 11-29-84
CONTEXT EMPLOYEE [Name = John]
RETRIEVE EMPLOYEE.Title
```

Example 2: Time-point reference in a OQL/T query.

If data reference is used for the specification of a time interval, WHEN is followed by an interval expression of the form "INTERVAL (Data Condition)". The Data Condition can specify a simple data condition of an object instance or a complex association pattern expression involving multiple classes. In both cases, the rest of the query will be evaluated against the temporal snapshot database defined by the time interval during which the data condition exists. Examples 3 and 4 illustrate the uses of data conditions in a simple and a more complex case, respectively.

In Q3, Employee in the CONTEXT clause is bound to the Employee specified in the WHEN clause (i.e., the employee Mary). In Q4 the association pattern used in the Interval function specifies that the time interval of interest is the one when employee John worked on project P1. The association operators "*'s" specify that the object instance John should be associated with some instance of Work-on which is also associated with the object instance P1. Employee_1 is an alias of Employee. It represents a different scan of the Employee instance (i.e., a different range variable). Very complex patterns of object associations involving tree and network structures among objects can also be specified as the data condition [see ALA90]. Also, the non-associate operator "!" can be used in the pattern expression to state that those objects in two classes that are not associated with each other would satisfy the search.

Q3: What was Mary's salary when she was a clerk?

```
WHEN INTERVAL(EMPLOYEE [Name = Mary,
Title = Clerk])
CONTEXT EMPLOYEE
RETRIEVE Salary
```

Example 3: Data Condition of a simple object instance.

Q4: What was Mary's salary when John worked on Project P1?

```
WHEN INTERVAL (Employee[Name = John]
* Worked_On * Project[P# = P1])
CONTEXT Employee_1 [Name = Mary]
RETRIEVE Salary
```

Example 4: Data Condition of a complex association pattern expression.

The data condition in a WHEN clause may sometimes return multiple time intervals since multiple snapshot databases defined by different time intervals may satisfy the data condition. In this case, different strategies can be followed. One strategy is to evaluate the rest of the query against all these intervals; the other is to select only one interval for query evaluation. In our implementation, we choose to evaluate a query against all the qualified intervals.

3.2 Temporal Functions

In addition to the interval function discussed above, several other useful functions are described below. Examples for these functions make use of the data shown in Tables 1 and 2.

Start-t	End-t	OID	Name	Title	Salary
<02-18-88,	- ,	125,	Mary,	Manager,	40K>
<10-21-85,	02-19-88,	#,	#,	Supervisor	33k>
<06-10-83,	10-20-85,	#,	#,	Secretary,	27K>
<06-10-81,	06-09-83,	#,	#,	Clerk,	22k>
<11-09-80,	06-09-81,	#,	#,	# ,	15K>

Table 1: Employee Mary's history

Start-t	End-t	OID	Name	Title	Salary
<11-23-88,	- ,	448,	John,	P.Manager,	45K>
<07-10-87,	11-22-88,	# ,	#,	P.Supervisor,	42K>
<02-02-84,	07-09-87,	# ,	#,	Engineer,	38K>
<11-23-82,	02-01-84,	# ,	#,	J. Engineer,	27K>

Table 2: Employee John's history

(1) FORMER: FORMER is a function used to retrieve the former historical instance relative to a reference instance. The general form of FORMER can be expressed as below:

FORMER(parameter) --> historical object instance

where, parameter is a historical event and the output is the historical object instance happened prior to the historical event. An example of this function is to find all the employees who had been "Senior Engineer" before they were promoted to "Project Supervisor" (see Example 5).

Q5: Find the employees who had been "Senior Engineer" before they were "Project Supervisor" ?

```
WHEN INTERVAL(FORMER
(EMPLOYEE[Title=Project Supervisor]))
CONTEXT EMPLOYEE [Title = Senior Engineer]
RETRIEVE Name
```

Example 5: Illustration of the FORMER function.

In this example, the object instances of EMPLOYEE specified in the WHEN clause are used to determine the temporal interval and to serve as an anchor for the object instances of the EMPLOYEE class specified in the CONTEXT clause. Therefore, no alias is needed in this query. When the query is evaluated, the system will first search for all the object instances in the EMPLOYEE class to check if any of their historical instances has a title of Project Supervisor. Then, based on this reference, the system will check if the former title relative to Project Supervisor is Senior Engineer or not. For those qualified object instances, their names will be retrieved. In our database example, there is no object instance which satisfies this temporal condition.

(2) NEXT: NEXT is a function used to retrieve the historical instance that follows a reference instance.

(3) TIME: TIME is a special function used to return a time point. Its parameter is NOW, which is a keyword representing the current time. The "+" and "-" symbols are used together with NOW to indicate the relative time to the current time. For example, "NOW + 3" stands for three time units ahead of the current time; and "NOW - 2" stands for two time units behind the current time. The meaning of TIME(NOW +/- x), therefore, is the projection of the relative time point to current time when this function is evaluated. When x is not given, TIME(NOW) is a projection of the current time. The TIME function is static whereas the construct NOW is dynamic because time is dynamic.

(4) START and END: START and END are used in OQL/T to retrieve the start_time and end_time of a temporal instance. An example for retrieving the time when Mary became a manager is shown below:

Q6: When did Mary become a manager?

```
WHEN INTERVAL(EMPLOYEE[Name
= Mary, Title = Manager])
CONTEXT EMPLOYEE
RETRIEVE START(EMPLOYEE)
```

Example 6: Illustration of START function

In this example, the system will search through Mary's record for the history instances which have the title Manager. Once the instances are found, the start time(s) of the interval(s) during which Mary's title was Manager will be the answer. Based on the data in Table 1, 02-18-88 is the answer to this query.

3.3 Interval Comparison Operators

The interval specified in the WHEN clause may subject to some temporal conditions. In this case, the WHERE subclause is used to specify a Boolean expression of time intervals and interval comparison operators. The expression specifies how the interval following the WHEN clause is related to some other time intervals. If the expression in the WHERE subclause is evaluated to True, the rest of the query will be processed against the snapshot database defined by the interval following WHEN. Otherwise, the query will not be executed.

Given two time intervals A = T[t1,t2] and B = T[t3,t4] which can be defined either by time or data reference discussed above, their start times and end times can have the following temporal relationships:

Case (1): t2 < t3 and t3 - t2 > one time unit

This case says that an interval A is before interval B, and the distance between them is greater than one time unit. The relationship between the two time intervals is said to be either BEFORE or AFTER. That is, interval A is BEFORE interval B, or interval B is AFTER interval A.

Case (2): t2 < t3 and t3 - t2 = time unit
(A PRECEDING B, B FOLLOWING A)

Case (3): t2 = t3 and t2 > t1 and t4 > t3
(A P-ADJACENT B, B F-ADJACENT A, A/B ADJACENT B/A)

Here, P stands for preceding and F for following. In case of a temporal condition in which the relative order of two time intervals are irrelevant, ADJACENT can be used to stand for either P-ADJACENT and/or F-ADJACENT.

Case (4): t1 < t3 < t2 < t4
(A P-CROSS B, B F-CROSS A, A/B CROSS B/A)

Case (5): t1 = t3 and t2 = t4 (A/B EQUAL B/A)

Case (6): t1 ≤ t3 ≤ t4 ≤ t2
(A CONTAIN B or B WITHIN A)

There are four possible combinations in this case and we introduce different names for these cases.

(a) t1 < t3 < t4 < t2 (A O-CONTAIN B or B I-WITHIN A; here O stands for "Outer" and I stands for "Inner")

(b) t1 = t3 < t4 < t2 (A L-CONTAIN B or B L-WITHIN A; here L stands for "Left")

(c) t1 < t3 < t4 = t2 (A R-CONTAIN B or B R-WITHIN A; here R stands for "Right")

(d) t1 = t3 < t4 = t2 (A EQUAL B or B EQUAL A)

Note: EQUAL is a special case of CONTAIN or WITHIN.

The above set of interval comparison operators is a super set of what was presented in [ALL83]. Different keywords are introduced for expressiveness and clarity. They can be used in a Boolean expression with parentheses to specify a complex interval condition in the WHERE subclause of WHEN.

3.4 Temporal Ordering Functions

The main concept of temporal ordering of an object instance's history is to sort the historical records of an object instance in an ascending order basing on their time stamps so that retrieval of historical records in a specified order can be specified [NAV89]. In OSAM*/T, we introduced FIRST, LAST, and NTH as the forward temporal ordering functions and B_FIRST, B_LAST, and B_NTH (where B stands for Backward) as the backward temporal ordering functions. The parameter for functions FIRST, LAST, B_FIRST, and B_LAST is either an object instance or a historical event, and the output is either the first object instance or the last object instance depending on the function used. The parameters for functions NTH and B_NTH are a number and an object instance, and the output is the object's historical instance in the order specified by the number. Query 7 illustrates the use of a forward temporal ordering function.

Q7: Retrieve the names and salaries of those employees whose starting salaries were greater than $25K.

```
WHEN INTERVAL(FIRST(EMPLOYEE))
CONTEXT EMPLOYEE [Salary > $25K]
RETRIEVE Name, Salary
```

Example 7: Illustration of the forward temporal ordering function.

3.5 Moving Window and Processing Functions

Two functions ANY and EVERY are used to implement the "Moving-Window" concept introduced in [NAV89]. A "Moving-Window" is a period of time which moves at a constant pace from the lower bound toward the upper bound of a time interval. In a "Moving-Window" application, conditions in a query will be evaluated as many times as the period shifts from the lower bound toward the upper bound of the time interval at a specified constant pace. That is, each time the period shifts, the conditions in a query will be re-evaluated in the new specified period.

The general structure of a WHEN clause containing ANY/EVERY is shown in Table 3 below:

```
WHEN INTERVAL(parameter) ANY/EVERY C WITHIN
T[A, B] INCREMENT_BY D
```

Table 3: General structure of ANY and EVERY functions.

ANY/EVERY is followed by a period of time C, a valid temporal range T[A, B], and the periodical pace D. Conditions specified in a query is evaluated upon the snapshot databases within period C. The lower bound and upper bound of the period C must be WITHIN T[A, B]. When the evaluation of the conditions is finished with the first period C, the lower bound and upper bound of C advance at the pace D to form the second period C. The operation in the query will then be evaluated upon the snapshot database of the second period C. This process continues until C is no longer WITHIN T[A, B]. If D is not given, the period C will advance at the pace of the time unit specified in C. If T[A, B] is not given, it will be defaulted to the lifespan of the historical object instance specified in the parameter. If an object instance's history starts later than A, then the lower bound of the first period C will be the starting time of the first instance of that object.

Q8: Find the employees whose titles had been changed more than twice within any two-year period during 1976 and 1988.

```
WHEN INTERVAL(EMPLOYEE) ANY 2 YEAR
     WITHIN T[1976, 1988] INCREMENT_BY 1 YEAR
CONTEXT EMPLOYEE
     WHERE TCOUNT(Title) > 2
RETRIEVE Name
```

Example 8: ANY as Moving-Window.

ANY, which is the same as the term "Moving-Window" used in [NAV89], is used to capture "there exist" concept; whereas EVERY is used to capture "for all" concept. Example 8 illustrates the use of ANY.

3.6 Set Operators

In some applications, it is necessary to involve the temporal information of different snapshot databases for the data manipulation of a particular snapshot database. The temporal information in these applications are used as the restricting condition in the time dimension on the interested temporal object instances. In order to achieve this temporal referencing, we introduce the Set Operators INTERSECT, DIFFERENCE, and UNION.

```
WHEN Time Interval 1
CONTEXT association pattern expression 1
        WHERE condition 1
Set Operator (Target Classes)
WHEN Time Interval 2
CONTEXT association pattern expression 2
        WHERE condition 2
```

Table 4: Syntax of a Query with a Set Operator.

The syntax of a query that involves a Set Operator is given in Table 4. The operands of a Set Operator are two contexts which define two separate temporal subdatabases each of which contains object classes and their instances that satisfy the association pattern and condition specified in the Context clause. The result of the set operation is a subdatabase derived from the two subdatabases. One restriction on the two contexts is that there must be at least one intersecting class between them and the operation of the Set Operator is performed on the intersecting class(es). For example, Set Operators can not be applied to the two contexts A * B * C and D * E * F because there is no intersecting class between them; however, they can be applied on the two contexts A * B * C and A * B * D * E because there are two intersecting classes A and B. In the above example, a set operation can be applied on class A, class B, or classes A and B depending on the user's requirement. The "Target Classes" following the Set Operator in Table 4 is used to specify the intersecting classes over which the set operator is performed. "Target Classes" is a member of the power set of all intersecting classes. If it is not provided, the maximal number of intersecting classes will be used as the default.

Based on this syntax, the INTERSECT set operator will return a subdatabase derived from "association pattern expression 1" of the first context. The subdatabase contains only those patterns of object

associations whose object instances of the specified "Target Classes" are also in the subdatabase generated by "association pattern expression 2". An example for the INTERSECT operation is given in Figures 3(a), (b), (c), and (d). In Figure 3(a), two subdatabases are derived basing on the two association pattern expressions "A*B*C" and "A*B*D*E". An INTERSECT set operation is performed between Context A*B*C and Context A*B*D*E over the intersecting class A. Since Context A*B*C produces a subdatabase in which class A contains {a1, a3, a4} and Context A*B*D*E produces a subdatabase in which A contains {a1 a2 a4}, the result of intersection yields {a1, a4} as shown in Figure 3(b). Figure 3(c) and (d) are the results of performing INTERSECT operation under two other conditions. The DIFFERENCE and UNION operators can be similarly defined.

In general, a OQL/T query can contain multiple contexts with each context linked to another through a set operator. A query with multiple contexts is evaluated in a bottom-up fashion. The last set operator in a query is evaluated first. The result of the evaluation is a subdatabase and is used as the second operand for the set operator preceding the last one. This process continues until all the set operators have been evaluated. An example on the use of the set operators is in Example 9. In this example, the employees whose salary were greater than $30K during [03-17-76, 03-17-77] and who had ever been a manager during the period [01-08-79, 12-09-83] are retrieved. In order to properly express this query, three contexts of different time snapshots and two INTERSECT operators are needed. Since the query is interested in current employees, the top-most context specifies a subdatabase containing all "current" employees.

```
Context Employee
 INTERSECT (Employee)
 WHEN T[01-08-79, 12-09-83]
 Context Employee [Title = Manager]
 INTERSECT (Employee)
 WHEN [03-17-76, 03-17-77]
 CONTEXT Employee [Salary >= $30K]
Retrieve Employee.Name
```

Example 9: An OQL/T query which contains multiple contexts linked by two INTERSECT Operators.

4. Conclusion

In this paper, we have presented a temporal object-oriented semantic association model, OSAM*/T. In this model, we use object time-stamping technique and the notions of Start-time and End-time to record the evolution of object instances in different classes. The technique of object time-stamping has led to several advantages such as the logical and physical separation of historical data and current data, the avoidance of normalizing hierarchically structured object instance, and the preservation of object history, object instance history and association history. In order to avoid redundant data, we adopted the "delta file" concept for object management. Additionally, we use knowledge rules to define special time notions for special temporal requirements and to avoid the introduction of many time notions which require an excessive amount of storage space.

In conjunction with the temporal model, we have presented a query language OQL/T for temporal information retrieval and manipulation. In OQL/T, several useful temporal functions, interval comparison operators, forward and backward temporal ordering functions, and set operators have been introduced. They are needed for expressing various temporal conditions of a data reference in an OQL/T query. OQL/T inherits all the features of OQL including its expressive power and non-procedural specification. It is a super-set of OQL. The implementation of the temporal OSAM* and temporal OQL takes advantage of our implemented prototype knowledge base management system OSAM* KBMS by extending its object manager, query processor and OQL translator.

References

[ALA89] Alashqur, A.M., S.Y.W. Su, and H. Lam, "OQL - An Object- Oriented Query Language", Proc. of the Int'l Conf. on VLDB, Amsterdam, The Netherlands, 1989, pp. 433-442.

[ADI86] Adiba, M., and Quang, N.B., "Historical Multi-Media Databases," Proc. of the Int'l Conf. on VLDB, 1986, pp. 63-70.

[AHN86] Ahn, I., "Towards an Implementation of Database Management Systems with Temporal Support," IEEE Proc. of the Int'l Conf. on Data Engineering, 1986, pp. 374-381.

[AHN88] Ahn, I., and Snodgrass, R., "Partitioned Storage for Temporal Databases," Information Systems Vol.13 No.4, 1988, pp.369-391.

[ALL83] Allen, J.F., "Maintaining Knowledge about Temporal Intervals", Comm. ACM, Vol.26, No.11, Nov. 1983, pp. 832-843.

[AND82] Anderson, T.L., "Modelling Time at the Conceptual Level," in Improving Database Usability and Responsiveness, P. Scheuermann (ed.), North Holland, 1982.

[ARI86] Ariav, G., "A Temporally Oriented Data Model," ACM TODS, Vol.11, No.4, 1986, pp. 499-527.

[BOL82] Bolour, A., Anderson, T.L., Deketser, L.J., and Wong, H.K.T. "The Role of Time in Information Processing: A Survey," ACM SIGMOD Rec., 1982, pp. 28-48.

[BOO86] Booch, Grady, "Object-Oriented Development", IEEE Trans. on Software Engineering, Vol.SE-12, No.2, February, 1986.

[CHE91] Chen, H.M., "Classification, Management, and Processing of Temporal Rules in OSAM*/T", paper in preparation, 1991.

[CLI83] Clifford, J., and Warren, D.S., "Formal Semantics for Time in Databases," ACM TODS, Vol.8, No.2, 1983, pp. 214-254.

[CLI85] Clifford, J., and Tansel, A.U., "On an Algebra for Historical Relational Databases: Two Views," ACM SIGMOD Conf., 1985, pp. 247-265.

[CHU90] Chuang, H. S., "Operational Rule Processing in a Prototype OSAM* Knowledge Base Management System, Master's Thesis, Dept. of Computer and Information Science, University of Florida, 1990.

[DAY85] Dayal, U. and J.M. Smith, "PROBE: A Knowledge-Oriented Database Management System", Proc. of the Islamorada Workshop, Feb., 1985.

[DE87] De, S., S. Pan, and A. Whinston, "Temporal Semantics and Natural Language Processing in a Decision Support System", Information Systems, Vol.12, No.1, 1987, pp.29-47.

[ELM90a] Elmasri, R., and Gene T.J. Wuu, "A Temporal Model and Query Language for ER Databases", IEEE proc. of the Int'l Conf. on Data Engineering, 1990, pp. 76-83.

[ELM90b] Elmasri, R., and Gene T.J. Wuu, "The Time Index: An Access Structure for Temporal Data", Proc. of the VLDB, 1990.

[ELM90c] Elmasri, R., I. El-Assal, and V. Kouramajian, "Semantics of Temporal Data in An Extended ER Model", in ER Conf., 1990.

[ELM90d] Elmasri, R., and C. Udomwongsa, "An ER Interface for An Object-Oriented System", Tech. Report #UH-CS-90-30, University of Houston, Aug. 1990, Houston, TX 77204.

[GAD86] Gadia, S.K., "Toward A Multihomogeneous Model For A Temporal Database," IEEE proc. of the Int. Conf. on Data Engineering, 1986, pp. 390-397.

[KLO83] Klopprogge, M.R., and Lockemann, P.C., "Modeling Information Preserving Databases: Consequence of the Concept of Time," Proc. of the Int'l Conf. on VLDB, 1983, pp. 399-416.

[LAM89] Lam, H., et. al., "Prototype Implementation of an Object- oriented Knowledge Base Management System", The Second Conference on Productivity in Computer Integrated Engineering and Manufacturing, Orlando, Florida, Nov. 1989.

[LOR88] Lorenitzos, N. A., and Johnson R. G., "Extending Relational Algebra to Manipulate Temporal Data." Inform. Systems, Vol.13, No.3, 1988, pp. 289-296.

[LUM84] Lum, V., Dadam, P., Erbe, R., Guenauer, J., and Pistor P., "Designing DBMS Support for The Temporal Diemnsion," Proc. ACM SIGMOD Conf., 1984, pp.115-130.

[McK86] McKenzie, E., "Bibliography: Temporal Databases," ACM SIGMOD Rec., Vol.15, No.4, 1986, pp. 40-52.

[NAV89] Navathe, S. B., and Ahmed, R., "A Temporal Relational Model and A Query Language," An international Journal of Information Science Journal, Vol.48, No.2, 1989, pp. 57-73.

[ROC75] Rochkind, Marc J., "The Source Code Control System," IEEE Trans. on Soft. Eng., Vol.SE-1, No.4, 1975, pp. 364-370.

[RAS88] Raschid, L., and S.Y.W. Su, "A Transaction Oriented Mechanism to Control processing in a Knowledge Base Management System", Proc. of the Int'l Conf. on Expert Database Systems, 1988.

[SCH83] Schiel, U., "An Abstract Introduction To the Temporal-Hierarchic Data Model," Proc. of the Int'l Conf. on VLDB, 1983, pp. 322-330.

[SEG87] Segev, A., and Shoshani A., "Logical Modeling of Temporal Data," Proc. ACM SIGMOD Conf., 1987, pp. 454-466.

[SEV76] Severance Dennis G., and Guy M. Lohman, "Differential Files: Their Application to The Maintenance of Large Databases," ACM TODS, Vol.1, No.3, 1976, pp. 256-267.

[SHO86] Shoshani, A., and Kawagoe, K., "Temporal Data Management," Proc. of the Int'l Conf. on VLDB, 1986, pp. 79-88.

[SIN90] Singh, M., The Enforcement of Integrity Constraints in Object-oriented Databases, Master's Thesis, Dept. of Electrical Engineering, University of Florida, 1990.

[SNO85] Snodgrass, R., Ahn, I., "A Taxonomy of Time in Database" Proc. ACM SIGMOD Conf., 1985, pp. 236-246.

[SNO86] Snodgrass, R., "Research Concerning Time in Databases: Project Summaries," ACM SIGMOD Rec., Vol.15, No.4, 1986, pp. 19-39.

[SNO87] Snodgrass, Richard, "The Temporal Query Language TQuel," ACM TODS, Vol.12, No.2, June 1987.

[STO86] Stonebraker, M. and L.A. Rowe, "The Design of POSTGRES", Proc. of the ACM SIGMOD, 1986, pp. 340-355.

[SU85] Su, S.Y.W., and L. Raschid, "Incorporating Knowledge Rules in a Semantic Data Model: An Approach to Knowledge Management", IEEE Conf. on AI Applications, December 1985.

[SU89] Su, S.Y.W., Lam, H., Krishnamurthy, V., "An Object-Oriented Semantic Association Model (OSAM*)" Chapter 17 in Artificial Intelligence: Manufacturing Theory and Practice, edited by S.T. Kumara, A.L. Soyster, and R.L. Kashyap, Published by the Institute of Industrial Engineers, Industrial Engineering and Management Press, Norcross, GA, 1989.

[SU91] Su, S.Y.W. and A.M. Alashqur, "A Pattern-Based Constraint Specification Language for Object_Oriented Databases", in COMPCON 91 Proc., San Francisco, Cal., Feb.25-Mar.1, 1991.

[TAN86] Tansel, A. U., "Adding Time Dimension to Relational Model and Extending Relational Algebra" Information Systems, vol.11, no.4., 1986, pp.343-355.

[TAN89] Tansel, A.U., Arkun, M.E., and Ozsoyoglu, G., "Time-by- Example Query Language for Historical Databases," IEEE Trans. on Soft. Eng., Vol. 15, No. 4, 1989, pp. 464-478.

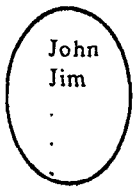


Fig. 1(a) Object instances in Employee class when Mary has not been hired as an employee

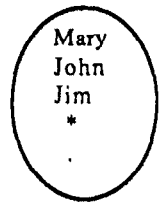


Fig. 1(b) Object instances in Employee class when Mary is hired as an employee (INSERT)

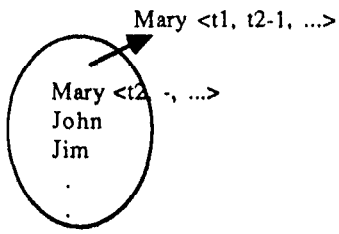


Fig. 1(c) Object instances in Employee class when object instance of Mary has been modified (UPDATE)

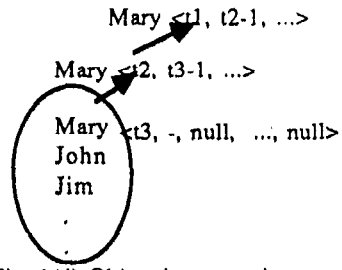


Fig. 1(d) Object instances in Employee class when object Mary has been deleted from Employee class (DELETE)

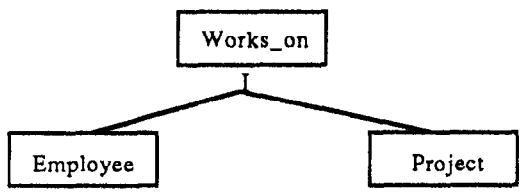


Fig 2(a): Interaction relationship between class Employee and class Project is modeled by Works_On class.

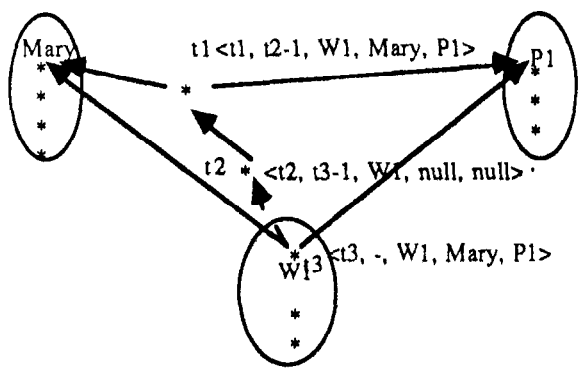
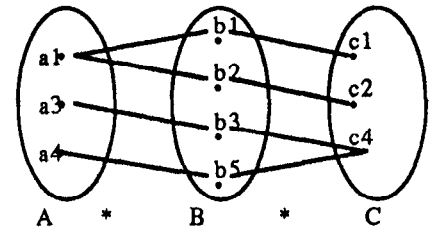


Fig. 2(b): History of object W1 in the Works_On class. Mary worked on project P1 from t1 to t2-1 and then was withdrawn from P1 at time t2. Between t2 and t3-1, Mary was not involved in P1 and this is noted by the absence of links between W1 and Mary and W1 and P1. From t3 on, Mary has been assigned back to project P1 again.



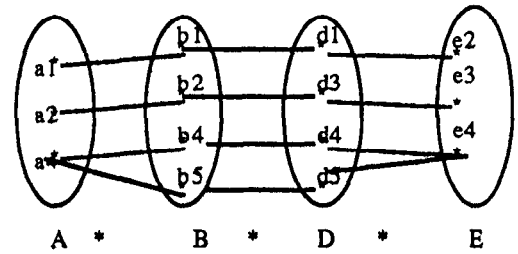
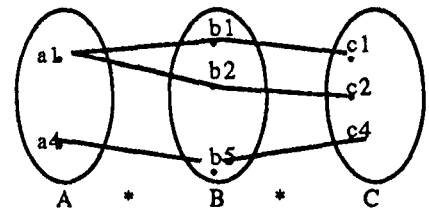
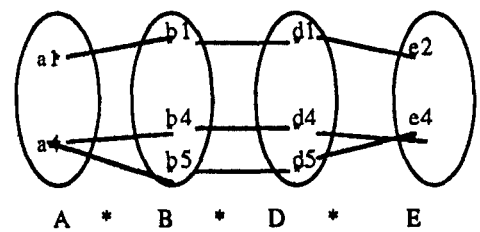


Fig. 3(a) Association patterns of two contexts A*B*C and A*B*D*E



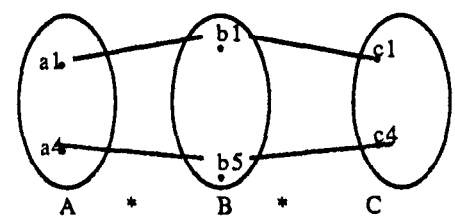
Context A*B*C INTERSECT (A) Context A*B*D*E

Fig 3(b) Result of applying INTERSECT Set Operator over A between the two contexts (A*B*C) and (A*B*D*E)



Context A*B*D*E INTERSECT (A) Context A*B*C

Fig. 3(c) Result of applying INTERSECT Set Operator over A between the two context (A*B*D*E) and (A*B*C)



Context A*B*C) INTERSECT (A, B) Context A*B*D*E

Fig. 3(d) Result of applying INTERSECT Set Operator over A*B between the two context (A*B*C) and (A*B*D*E)