A Uniform Framework for Processing Temporal Object Queries

L. Wang, M. Wing, C. Davis, N. Revell
School of Computing Science, Middlesex University
Bound Green Road, London N11 2NQ, United Kingdom
email: {l.c.wang, m.wing, c.davis, n.revell}@mdx.ac.uk

Abstract

This paper explores a uniform framework for processing temporal queries in the context of object-oriented databases. A temporal object data model is developed by extending the unified model of RDBs and OODBs from UniSQL/X with a time-dimension, that forms temporal relational-like cubes but with aggregation and inheritance hierarchies. A query algebra is thereby defined to provide an access of objects through these associations of aggregation, inheritance and time-reference. Due to the hierarchy of the data model and reducibility of the query algebra, an extensible approach to processing temporal object queries within the uniform query processing framework is attained. A set of transformation rules is identified for query rewrite, that includes the known relational and object rules plus those pertaining to the time-reference. To evaluate temporal queries involved in the path, a decomposition strategy is proposed. It has been shown that temporal object queries can be processed and optimized within the uniform query processing framework.

1 Introduction

Whilst the semantic limitations of relational databases (RDBs) are widely recognised, object-oriented database systems (OODBs) are emerging as the most promising database technology to satisfy the needs of advanced database applications such as computer-aided design (CAD), computer-aided manufacturing (CAM), computer-aided software engineering (CASE), document and multimedia preparation, office automation and scientific computing. Many of these advanced database applications require a support for time-varying data. Management of temporal data becomes one of the key challenges that today's OODBs need to address [6, 18]. As reported in [10, 11, 13, 8], the vast majority of research on temporal databases (TDBs) has been focused on the incorporation of time elements into existing data models. Little work that has been done on query processing and optimization [7, 12, 14] is in the context of RDBs. Dayal and Wuu [3] proposed a uniform approach to processing temporal queries in the context of functional object-oriented data model. But their work didn't take account the query optimization and evaluation within a query processing framework. Also their work is based on a functional model and language, that would lead the functional optimization quite different from the algebraic, cost-based optimization techniques employed in RDBs as well as a number of OODBs [9].

In this paper, we investigate processing temporal object-oriented queries within a uniform framework. By the uniform framework, we mean that temporal queries can be processed and optimized within the existing object-oriented framework that is extended from the relational framework, through smoothly extending the techniques developed for existing OODBs and TDBs.

Query optimization techniques are dependent upon the query model/language. The query model, in turn, is based on the data model underlying the data, as the latter defines the access primitives

0-8186-8551-4/98 $10.00 © 1998 IEEE

that are used by the query model [9]. So our work started from the development of temporal object data model. We adopt the unified model of RDB and OODB from UniSQL/X [6, 2] as a snapshot object data model, and then incorporate within it with a time dimension. A temporal object query algebra is thereby defined as a query model/language to provide an access of objects. The features of the hierarchy of data model and the reducibility of query algebra provide a basis to explore or extend the techniques and strategies for processing temporal object queries within the existing query processing framework. An extensible approach to temporal query processing within the uniform framework is therefore attained.

The remainder of this paper is organised as follows. Section 2 describes the adaptation of the unified data model of RDB and OODB with the inclusion of a time dimension. The algebra for the data model is given in Section 3. Section 4 presents an extensible approach to temporal query processing. Processing the temporal query components is discussed in Section 5. Section 6 includes conclusions and future work.

2 The data model

2.1 The unified model of OODB and RDB

The unified data model of RDB and OODB from UniSQL/X [6, 2] extends the relational data model in three important ways, each reflecting a key object-oriented concept: (1) nested predicates; (2) inheritance; (3) methods. Allowing a column of a relation to hold a tuple of another relation directly leads to a nested relation. Allowing the users to attach procedures to a relation and to have the procedures operate on the column values in each tuple achieves the combination of data with program. Allowing the users to organise all relations in the database into a hierarchy, such that between a pair of relations P and C, P is made the parent of C, if C takes (inherits) all columns and procedures defined in P (besides those defined in C), the relational model integrates the object-oriented concept of inheritance. UniSQL/X actually makes one more extension to the relational model: allowing the row/column entry of a relation to have a set of values (i.e., any number of values), rather than just a single value; and further allowing the set of values to be of more than one arbitrary data type. This provides an ability to represent the many-to-many relationship between two collections. The model is an object-oriented model and it is adopted as a snapshot model to incorporate time.

In addition, we preserve the basic object concepts such as "any real-world entity is uniformly modelled as an object", "each object is associated with a unique identifier", *etc.*, so that *heterogeneity* in the time dimension and the *grouped completeness* of algebra can be maintained, as discussed in our previous work [17].

2.2 A temporal object

Let $T = \{..., t_0, t_1, ...\}$ be a set of times, at most countably infinite, over which is defined the linear (total) order $<^T$, where $t_i <^T t_j$ means t_i occurs before (earlier than) t_j. For the sake of simplicity, we can assume that T is isomorphic to the set of natural numbers. Any subset of T is called a temporal set. A temporal set can be represented as a union of disjoint time intervals. The most basic property of temporal sets is that they are closed under finite unions, intersections, and complementation. That is, if T_1 and T_2 are temporal sets, then so are $T_1 \cup T_2$, $T_1 \cap T_2$, $T_1 - T_2$, *and* $\neg T_1$.

If an object o, which is any real world entity, exists for a certain period of time, which is a subset of T (*i.e.*, the temporal set), this period is called the object's lifespan, denoted as $L(o)$ for the object o. If the lifespan $L(o)=[t_{start}, t_{end}]$, the duration of time is called a span: $span(o)= t_{end} - t_{start} + 1$. In

order to support for derived lifespans, we allow the usual set-theoretic operations over lifespans. That is, if L_1 and L_2 are lifespans, then so are $L_1 \cup L_2$, $L_1 \cap L_2$, $L_1 - L_2$, and $\neg L_1$.

A temporal object is defined as a time sequence (TS for short): $\{t, o(t)\}$, $t \in L(o) \subset T$, denoted as $<L(o), o(t)>$, (or simply $o(t)$ or o). A temporal object $<L(o), o(t)>$ asserts that the object $o(t)$ is valid for its lifespan $L(o)$ and its value changes with time. For a constant object o, it may be represented with no timestamp where its time reference is implied as $L(o)$. It can also be represented with the explicit time reference as a temporal object $<L(o), o>$. **Fig. 1** gives three basic types of time-varying entity. For the step-wise constant, $<L(o), o(t)> = \{..., t_{i-1}, ov_{i-1}; t_i, ov_i; t_{i+1}, ov_{i+1}; ...\}$. The term *epoch* from signal processing is used to refer to the time at which the object changes its value, *e.g.*, t_i. The interval during which the value ov_i persists is determined by the epoch t_i and its succeeded epoch t_{i+1}, *i.e.*, $[t_i, t_{i+1})$. If there are n elements in a TS, it is said that there are n epochs. For the discrete time event, if the value of the entity is recorded at every single time point (if it does not, it can be represented by a time series with unequal spacing in time like the step-wise constant, and the entity value at the time point between recorded time points can be assumed or determined depending on the application), we have $<L(o), o(t)> = \{..., t_{i-1}, ov_{i-1}; t_i, ov_i; t_{i+1}, ov_{i+1}; ...\} = \{..., ov_{i-1}, ov_i, ov_{i+1}, ...\} = \{ov_i\}$, *i.e.*, it can be represented by a discrete time series. For the continuous time event, it can be treated as a sampled continuous time signal, and therefore be represented by a discrete time series.

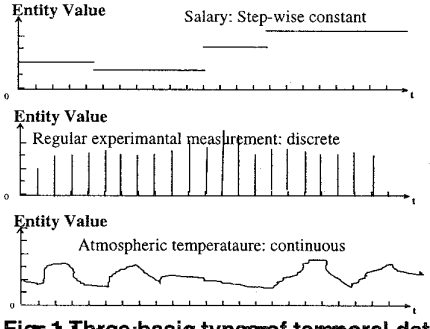

Fig. 1. Three basic types of temporal data

If a TS contains a value for each time point in the lifespan duration, it is called a regular TS [5] (*e.g.*, the above discrete time event). If a TS contains values for only subset of time points within the lifespan, it is called an irregular TS (*e.g.*, the step-wise constant). The complete history of an irregular TS can be reconstructed by a function, *e.g.*, linear interpolation. As a TS is a set, so a temporal object can be represented by its sub-objects. In practice the lifespan may consist of disjoint, noncontiguous segments, as in [5] we prefer to use null values rather than defining multiple segments in the lifespan.

Table 1. Interaction of Tuple Lifespan and Attribute Lifespan

Relation	A_1	A_2	...	A_n
$tuple_1$				
$tuple_2$				
...	...	...	...	
$tuple_m$				$value_{m,n}$

In OODBs, every real world entity is uniformly modelled as an object that is grouped into a class (relation) and interrelated to other objects through associations. Now we isolate a class (relation) C from its association relationship, as shown in **Table 1**. The $value_{m,n}$ is an object with lifespan $l_{m,n}$. The $tuple_m$ is also an object, its lifespan is denoted as $L(t_m)$. We have

$L(t_m) = l_{m,1} \cup l_{m,2} \cup ... \cup l_{m,n}$

The lifespan of attribute A_n is

$L(A_n) = l_{1,n} \cup l_{2,n} \cup ... \cup l_{m,n}$

The lifespan of relation C is

$L(C) = L(A_1) \cup L(A_2) \cup ... \cup L(A_n) = L(t_1) \cup L(t_2) \cup ... \cup L(t_m)$

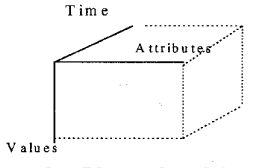

Fig. 2. A 3-Dimensional Class

Thus a 2-dimensional relation (class) "table" becomes a 3-dimensional "cube", as shown in **Fig. 2.**

It is obvious that

$l_{i,j}=L(t_i)\cap L(A_j)$

This implies that there is no value for an attribute in a tuple for any moment in time outside the intersection of the life spans of the tuple and the attribute. Clearly our temporal object model can support a completely heterogeneous temporal dimension.

It is possible to refer to the components of a temporal object. For a temporal object $o=<T,o>$, $o.v$ and $o.T$ refer to its value and temporal set components, respectively (sometimes we omit v, *i.e.*, $o.v=o$, (or $o.v(t)=o(t)$) to refer to the value of the object o). Let A be the name of an attribute that can take a temporal object for its values, then $A.v$ and $A.T$ represent names for the value and temporal set components of the attribute A. Further, the same notation may be applied to class (relation) C. If C is the name of a temporal relation, then $C.v$ and $C.T$ represent names for the value and temporal set components of the class C.

If the domain of attribute A_i of class C is another class C', then implicitly, $L(A_i)=L(C')$. If class C is a subclass of class C', then $L(C)=L(C')$. Moreover, if a database consists of n classes (relations) $C_1, C_2,...C_n$, the lifespan of the database schema is $L= L(C_1)\cup L(C_2)\cup...\cup L(C_n)$.

Table 2 Summary of basic algebraic operators

Operations	Definition	Notes
$O(T_1)$Time-insert $O(T_2)$	$O_3=\{o \mid o\in O\}$ where $L(O_3)=T_1\cup T_2$	$O(T_1)\subseteq O$, $O(T_2)\subseteq O$
$O(T_1)$Time-delete $O(T_2)$	$O_3=\{o \mid o\in O\}$ where $L(O_3)=T_1-T_2$	$O(T_1)\subseteq O$, $O(T_2)\subseteq O$
Difference O_1-O_2	$O_3=O_1-O_2=\{o \mid o\in O_1 \wedge \neg o\in O_2\}$ where $L(O_3)=L(O_1)-L(O_2)$	O_i is a collection. $L(O_i)$ is the life-span of O_i.
Union $O_1\cup O_2$	$O_3=O_1\cup O_2=\{o \mid o\in O_1 \vee o\in O_2\}$ where $L(O_3)=L(O_1)\cup L(O_2)$	
Intersection $O_1\cap O_2$	$O_3=O_1\cap O_2=\{o \mid o\in O_1 \wedge o\in O_2\}$ where $L(O_3)=L(O_1)\cap L(O_2)$	
Select σ_P O	$\sigma_P O=\{o \mid o\in O \wedge P(o,t)\}$	σ_P O selects the elements "o" of set O such as the predicate P(o, t) holds.
Map $g:O_1\rightarrow O_2$	$g:O_1\rightarrow O_2=\{g(o) \mid o\in O_1\}$	For the type of objects in O_1 (i.e., $o\in O_1$), g returns an object of type of O_2 (i.e., $g(o)\in O_2$).
Project $\pi_{<A1,...,Ai>}O$	$\pi_{<A1,...,Ai>}O$ $=\{<A_1:g_1(o),..., A_i:g_i(o)> \mid o\in O\}$	If $g_i=1$ it returns the OID of the domain object of A_i unless A_i is atomic. We retain $g_i=1$ so that *project* on a set of objects (relation) likes the relational *project*.
Join $O1 \bowtie_{p<A_{o1}.A_{o2}>} O2$	$O1\bowtie_{p<A_{o1}.A_{o2}>} O2=\{<A_{o1}:o1, A_{o2}:o2> \mid o1\in O1 \wedge o2\in O2 \wedge P(o1,o2)\}$	Essentially a θ-join as in relational algebra.
Time-slice $\S_{T1}(O)$	$\S_{T1}(O)=\{o \mid \forall t\in (T_1\cap L(o)) [o(t)\in O]\}$	The life-span of $\S_{T1}(O)$ is $T_1\cap L(o)$. Time-slice purely reduces the relation along the temporal dimension. If T_1 equals to time point t_1, i.e., $T_1=t_1$, then $\S_{T1}(O)$ represents an event $o(t_1)$ happened at t_1.
Offset $\gamma(O, l)$	$\gamma(O(t_1), l)= O(t_1+l)$	"Shifts" a snapshot relation at t_1 by the number of positions specified by the offset l.
When $\varpi(O)$	$\varpi(O)=L(O)$	Maps a set of objects O to its temporal set.

3 Query algebra

From the algebra point of view, a temporal OODB (TOODB) can be viewed as a collection of temporal objects, grouped together in classes (relations) and interrelated through three associations: aggregation, generalisation and time-reference. Each temporal relation can be viewed as a 3-dimensional structure (*i.e.,* a cube).

Basically, the standard relational algebra provides a unary operator for each of its two dimensions: *select* for the value dimension and *project* for the attribute dimension. The temporal algebra supports an operation on the third dimension, *i.e.,* the time dimension: *time-slice*. An object algebra allows the predicate of the *select* operation on a contiguous sequence of attributes along a branch of class-aggregation hierarchy. This sort of query is usually represented by a path [1] and is often referred as an implicit join. We have defined the enhanced path that extends the path with time-reference so that the *select* provides an access of data along associations of both aggregation hierarchy and time-reference. A complete set of algebraic operators has been defined for our temporal object data model (The detailed definitions are given [17]). **Table 2** simply lists some basic algebra operators.

4 An extensible approach to processing temporal object-oriented queries

The temporal object data model and an algebra for this model presented in the previous sections provide a basis for query processing. Due to the hierarchical feature of our data model, as shown in **Fig. 3**, our algebra possesses the property of reducibility. That is, when time dimension is not taken into account the temporal object algebra will be reduced to the object algebra and when the object-oriented features of aggregation and inheritance are not taken into consideration, the algebra will be reduced to the relational algebra. We therefore could explore an extensible approach to processing temporal queries within the existing object query processing framework that in turn is extended from relational query processing framework.

4.1 Optimizer hierarchy

Our optimizer is of a hierarchical structure as shown in **Fig. 4**, where the temporal optimizer is built on the top of the object optimizer that, in turn, is on the top of the relational optimizer. The temporal object queries can be processed and optimized within the existing query processing framework through smoothly extending the existing query processing techniques.

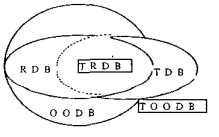

Fig. 3 Data model hierarchy

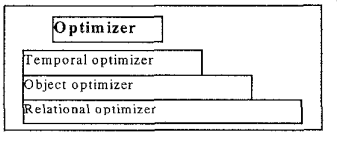

Fig. 4 Optimizer hierarchy

4.2 Query transformations

One important aspect of query optimization is the transformation of one query into an equivalent query that might be more efficient to evaluate. The size of the search space of equivalent query plans for a snapshot query is determined in part by the algebraic equivalence available in the snapshot algebra. As we represent a temporal object as a time series that can

be thought of as the equivalence of a 'blob' object, the transformation rules in object algebra can directly apply. As our object algebra is also consistently extended from relational algebra, the relational algebraic transformation rules can directly apply. Now we identify the following transformation rules.

Relational rules

The following set of equivalencies is derived from well-known algebraic optimization techniques in RDBs. They are called relational rules.

$C_1 \cup C_2 \equiv C_2 \cup C_1$	*commutative law*
$C_1 \cap C_2 \equiv C_2 \cap C_1$	*commutative law*
$C_1 \rhd\lhd C_1 \equiv C_1$	
$C_1 \cup C_1 \equiv C_1$	*idempotent law*
$C_1 \cap C_1 \equiv C_1$	*idempotent law*
$C_1 - C_1 \equiv \varnothing$	
$C_1 \cup \varnothing \equiv C_1$	
$C_1 \cap \varnothing \equiv \varnothing$	
$C_1 \rhd\lhd \varnothing \equiv \varnothing$	
$C_1 - \varnothing \equiv C_1$	
$\varnothing - C_1 \equiv \varnothing$	
$C_1 \rhd\lhd C_2 \equiv C_2 \rhd\lhd C_1$	*commutative law*
$C_1 * C_2 \equiv C_2 * C_1$	*commutative law*
$C_1 \rhd\lhd (C_2 \rhd\lhd C_3) \equiv (C_1 \rhd\lhd C_2) \rhd\lhd C_3$	*associative law*
$C_1 * (C_2 * C_3) \equiv (C_1 * C_2) * C_3$	*associative law*
$\sigma_{P1}(C_1 \cup C_2) \equiv \sigma_{P1} C_2 \cup \sigma_{P1} C_1$	
$\pi_{Ai}(C_1 \cup C_2) \equiv \pi_{Ai} C_2 \cup \pi_{Ai} C_1$	
$\sigma_{P1}(\sigma_{P2}(C_1)) \equiv \sigma_{P2}(\sigma_{P1}(C_1)) \equiv \sigma_{P1 \wedge P2}(C_1)$	

Temporal transformation rules

When time-references are taken into account, the following transformation rules play a role:

$Time\text{-}slice_{T1}(\sigma_{P1}(C_1)) \equiv \sigma_{P1}(Time\text{-}slice_{T1} C_1)$

$Time\text{-}slice_{T1}(\pi_{Ai}(C_1)) \equiv \pi_{Ai}(Time\text{-}slice_{T1} C_1)$

$Offset_1(\sigma_{P1}(C_1)) \equiv \sigma_{P1}(Offset_1 C_1)$

$Offset_1(\pi_{Ai}(C_1)) \equiv \pi_{Ai}(Offset_1 C_1)$

$Agg\text{-}func_{T1}(\pi_{Ai}(C_1)) \equiv \pi_{Ai}(Agg\text{-}func_{T1} C_1)$

It is a good heuristic to push *select, project* and *time-slice* or *offset* as far down the query graph as possible, *esp.*, to perform the time-slice as early as possible.

It has been identified some transformations are incorrect in general [14]:

- A selection can not be pushed through an aggregate operator or offset operator.
- An aggregation operator can not be pushed through an offset operator and vice versa.

Path transformation rules

In OODBs, *select* operation allows its predicate on a contiguous sequence of attributes along a branch of the class-aggregation hierarchy. We use the path to express this sort of predicate and the

enhanced path expression to represent the path that includes the time references. For simplicity we suppose the time-reference occurs at the end of the path (even if it is not, it will only involve the looking up storage and will not provide any difficulty to the query optimization).

Generally speaking, the attribute/domain link between a class C and domain D of one of the attributes A of C creates the join between the class C and D, in which the attribute A of the class C and identifier *OID*, which is defined by the system and which can be considered as an attribute of class D, are join attributes [1]. Therefore, an object query with a path expression involving N classes is equivalent to a relational query, which requires the join in N relations corresponding to N classes [1]. This is why *select* operator is usually called *an implicit join*. According to the definition of our algebra, when the predicate of *select* involving a path expression is as *path op value*=$C_1.A_1.A_2....An^{TM}$ *op v*, the equivalence between an implicit join and an explicit join is as

$$\sigma_{C1.A1.A2....An^{TM}\ op\ v}(C_1) = \pi_{A(C1)}(C_1 \rhd\lhd^P \pi_{A2}(C_2) \rhd\lhd^P \ldots \rhd\lhd^P \sigma_{An\ op\ v}(C_n)) \quad (1)$$

We use $\rhd\lhd^P$ to represent the join that is slightly different from the join defined in our algebra in that the join attributes are the attribute A_{i-1} of the class C_{i-1} and identifier *OID* of C_i, if we join C_{i-1} and C_i. The p*roject* specifies the query target.

If there is a complex predicate involving a single path such as

$P = C_1.A_1\ op\ v_1$ and $C_1.A_1.A_2\ op\ v_2$... and $C_1.A_1.A_2...A_n^{TM}\ op\ v_n = P_1$ and P_2...and P_n

then we have a general form

$$\sigma_{P(C1.A1.A2....An^{TM})}(C_1) = \pi_{A(C1)}(\sigma_{P1}C_1 \rhd\lhd^P \pi_{A2}\sigma_{P2}(C_2) \rhd\lhd^P \ldots \rhd\lhd^P \sigma_{Pn}(C_n)) \quad (2)$$

where P_i is optional that can be omitted if it does not exist, P_n involves both time and value dimensions, and the first project specifies the query target.

The way to visit the path introduces a path traversal operator. The *Linear path traversal operator* is a navigational operator $N\_op[C_1.A_2....A_n^{TM}]$ to execute the implicit join along a path. It is equivalent to a set of joins as in equation (2):

$$N\_op[C_1.A_1.A_2....A_n^{TM}] = \sigma_{P1}C_1 \rhd\lhd^P \pi_{A2}\sigma_{P2}(C_2) \rhd\lhd^P \ldots \rhd\lhd^P \sigma_{Pn}(C_n) \quad (3)$$

According to the associative law in previous subsection, the above linear path traversal operator can be further rewritten into the following form:

$$N\_op[C_1.A_1.A_2....A_n^{TM}] = N\_op[C_1.A_1.....A_{n-1}] \rhd\lhd^P \sigma_{Pn}(C_n) = N\_op[C_1.A_1.A_2....A_{n-1}] \underset{C_{n-1}.A=OID(C_n)\wedge P_n}{\rhd\lhd} C_n \quad (4)$$

Thus

$$\sigma_{P(C1.A1.A2....An^{TM})}(C_1) = \pi_{A(C1)}(N\_op[C_1.A_1.....A_n] \underset{C_{n-1}.A=OID(C_n)\wedge P_n}{\rhd\lhd} C_n) \quad (5)$$

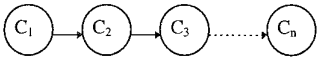

Fig. 5 A sample path

temporal queries

4.3 A decomposition strategy for processing

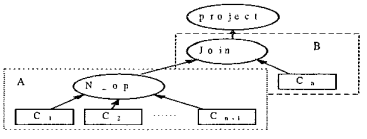

Fig. 6 A operator graph

Fig. 5 is a single path with n classes. **Fig. 6** gives an operator graph (OG) for equation (5) that involves such a single path. An OG is a labelled n-ary tree where the leaf nodes represent collection of objects, the non-leaf nodes represent operators (*e.g.*, join, navigational operator, *etc.*), and the edges represent temporary collections that can be represented by support tables

[4]. A support table can be regarded as a collection of tuples of qualified object identifiers and attributes. Two support tables can be joined together if there exists a common supported collection between them. The execution of an OG follows bottom-up order.

In order to processing temporal queries, we have assumed a temporal class is at the end of a path. Therefore the evaluation of the enhanced path can be initially decomposed by dividing the path into two parts: a sub-path with time-referenced class and an ordinary sub-path (without time-stamped classes) that can be further divided into sub-paths.

The decomposition strategy for processing temporal queries can be further illustrated in **Fig. 7**. A complex user query with path expressions that involves time-references is first translated into a set of single path expressions. A single path is then divided into two sub-paths: a sub-path containing a time-stamped class that can be optimized by making use of the ordering information of temporal data and an ordinary sub-path that can be further decomposed and traversed using different algorithms. The intermediate results of traversed two sub-paths are joined together to create the output query.

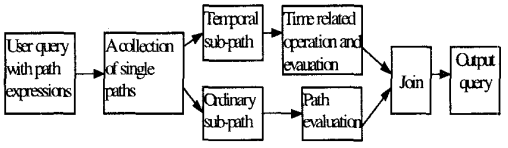

Fig. 7 A decomposition strategy

5 Processing temporal query components

The block *B* in **Fig. 6** consists of temporal predicate evaluation (as well as time-series processing) and a join, which can be further expressed in **Fig. 8**. As temporal data provide more opportunities for optimization [7, 14], temporal optimization comes into play at this stage. When optimizing such a query, the object optimizer takes charge of the outer query block, and the temporal optimizer operates on the nested query block. Each optimizer is responsible for its own query blocks.

5.1 Time-related operations and optimization

Predicate evaluation in **Fig. 8** involves the time-related operation and value evaluation. Temporal operations such as *time-slice, offset, aggregation* can be treated as methods and its output can then participate in the value evaluation. Temporal optimizer must be sure to 'plan' the evocation of function and make use of the ordering information for optimization. Stream processing approach is a strategy for optimization utilising the ordering information of data [7, 14].

5.2 Join

Let *C* and *D* represent the supporting tables or the intermediate results of block *A* in **Fig. 6** and *B'* in **Fig. 8**, as shown in **Fig 9.** There are various join algorithms to join C and D together. The advantage in representing it as an explicit join is that we can use well-known join algorithms to perform optimization. Here a temporal object stands as a 'blob' object that can be treated as an ordinary object in a snapshot OODB. Two basic types of joins are: forward join and reverse join.

Forward-Join (FJ)

The idea of FJ can be informally presented as follows [1]. Given a pair of classes *C* and *D*, such that *D* is the domain of an attribute *A* of *C*, the values of that attribute *A* are identifiers of

the instances of class *D*. If classes *C* and *D* are in this order, in a given permutation, the join between these classes is obtained by the following operations:

For c in C do
begin
retrieve c.A
retrieve d in D such that
OID(d)=c.A if A is single-valued
OID(d)∈c.A if A is multi-valued;
evaluate the predicate on d
end

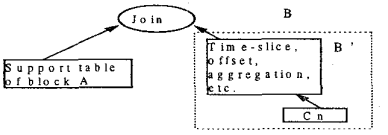

Fig. 8 Further decomposition of temporal sub-path

Implementation of forward join can integrate pointer-chasing with well-known join algorithms for optimization, such as pointer-based nested loops, pointer-based sort-merge and pointer-based hybrid-hash, *etc.*[15].

If, however, the permutation involves retrieving the instances in class *D* before the retrieval of the instances in *C*, then the following reverse join schema will apply.

Reverse-Join (RJ)

The idea of RJ can be informally presented as follows [1].

for d in D do
begin
retrieve u=OID(d);
evaluate the predicate on d
retrieve c in C such that
c.A=u if A is single-valued
u∈c.A if A is multi-valued;
end

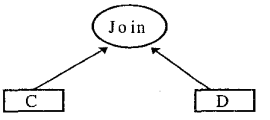

Fig. 9 Join between block A and block B'

As there is no direct link from *D* collection to *C*, a value-based join must be used to check the *OID* membership condition. This algorithm is efficient when the predicate in the last collection is selective. It performs value-based comparisons of *OIDs*, which is generally inefficient on a CPU [4]. Again, the implementation can employ relational algorithms of nest-loop join, sort-merge join, *etc.*

6 Conclusions

In this paper, we present a uniform framework for processing temporal object-oriented queries. The temporal data model presented is of a hierarchical structure: it forms the relational-like cubes but with aggregation and inheritance associations, as it is evolved from the unified model of RDB and OODB that is in turn extended from the relational model. A query algebra, that is thereby defined to provide an access of objects through these associations of aggregation, inheritance and time-reference, possesses the property of reducibility, *i.e.,* when the time-reference is not taken into account, it will be reduced to the object algebra that in turn will be reduced the relational algebra when the hierarchies of aggregation and inheritance do not exist. The hierarchical structure of data model and the reducibility of query algebra provide a basis to extend existing query processing and evaluation techniques for temporal object query processing. Query transformation can then be carried out based on a set of algebraic transformation rules identified. A strategy of decomposition is then proposed for processing temporal queries that involve paths. That is, evaluation of an enhanced path, which is defined to refer to the path with the time-reference, is decomposed by

initially dividing the path into two sub-paths: one containing the time-stamped class that can be optimized by making use of the ordering information of temporal data and another an ordinary sub-path (with no time-stamped class) which can be further decomposed and evaluated using different algorithms. The intermediate results of traversed two sub-paths are then joined together to create the query output. It has been shown that techniques developed for relational join, pointer-based join and sequential processing can be adapted to process the decomposed temporal query components. A temporal object is defined as a time series which can be treated as a function of time, that avoids the complexity of a general temporal relational join. Temporal optimizer is built on the top of object optimizer, that requires less modification of object optimizer since the techniques of object-oriented query processing and evaluation as well as sequence processing can directly apply.

Future work will involve a detailed study of execution algorithms and cost analysis.

References

1. Bertino, E and Martino, L. 1993. *Object-Oriented Database Systems: Concepts and Architectures.* Addison-Wesley Publishers Ltd.
2. D'Andrea, A. and Janus, P. 1996. UNISQL's next-generation object-relational database management system. *SIGMOD RECORD*, 25(3):70-76.
3. Dayal, U. and Wuu, G.T.J. 1992. A uniform approach to processing temporal queries, *Proc. of the 18th Int. Conf. on VLDB*, p407-417, Canada.
4. Gardarin, G., *et al.* 1996. Cost-based selection of path expression processing algorithms in object-oriented databases. *Proc. of the 22th Int. Conf. on VLDB*, p 390-401, Mumbai, India.
5. Ginsburg, S. 1993. A temporal data model based on time sequences. *Temporal Databases: Theory, Design, and Implementation* (edited by A. U. Tansel, *et al.*), p248-270. Benjamin/ Cummings Publishing.
6. Kim, W. 1993. Object-oriented databases systems: promises, reality, and future. *Proc. of the 19th Int. Conf. on VLDB*, p676-687, Dublin, Ireland.
7. Leung, T.Y.C., and Muntz, R.R. 1993. Stream Processing: Temporal Query Processing and Optimization. *Temporal Databases: Theory, Design, and Implementation* (edited by A.U. Tansel, *et al.*), p329-355. Benjamin/ Cummings.
8. Ozsoyoglu, G. and Snodgrass, R. T. 1995. Temporal and real-time databases: a survey. *IEEE Trans. on Knowledge and Data Engineering*, 7(4):513-532.
9. Ozsu, M. T. and Blakeley, J. A. 1995. Query processing in object-oriented database system. *Modern Database Systems: the Object Model, Interoperability, and Beyond,* (edited by W. Kim), p146-174. ACM Press.
10. Pissinou, N., *et al.* 1993. On temporal modelling in the context of object databases. *SIGMOD RECORD,* 22(3): 8-15.
11. Pissinou, N., *et al.* 1994. Towards an infrastructure for temporal databases: report of an invitational ARPA/NSF workshop. *SIGMOD RECORD,* 23(1): 35-51.
12. Segev, A. 1993. Join processing and optimization in temporal relational databases. *Temporal Databases: Theory, Design, and Implementation* (edited by A.U. Tansel, *et al.*), p356-387. Benjamin/ Cummings.
13. Segev, A., *et al.* 1995. Report on the 1995 international workshop on temporal databases. *SIGMOD RECORD*, 24(4): 46-52.
14. Seshadri, P., *et al.* 1996. The design and implementation of a sequence database system, *Proc. of the 22th Int. Conf. on VLDB*, p 99-110, India.
15. Shekita, E. J. and Carey, M.J. 1990. A performance evaluation of pointer-based joins. *Proc. of ACM SIGMOD Conf.*, p300-311, Atlantic, NJ.
16. Snodgrass, R. 1995. Temporal object-oriented databases: a critical comparison. *Modern Database Systems: the Object Model, Interoperability, and Beyond,* (edited by W. Kim), p 386-408. ACM Press.
17. Wang, L., Wing, M., Davis, C. and Revell, N. 1996. An algebra for a temporal object data model, *LNCS 1134, Database and Expert Systems Applications*, p667-677, Proceedings, Zurich, Switzerland.
18. Wang, L., Wing, M., Davis, C. and Revell, N. 1996. Query processing in object-oriented databases. *Proc. of 13th European Meeting on Cybernetics and Systems Research,* p803-808, Vienna, Austria.