Providing Temporal Support in Data Base Management Systems for Global Change Research*

Ke Qiu
Nabil I. Hachem
Matthew O. Ward
Michael A. Gennert

Data/Knowledge Base Research Group
Computer Science Department
Worcester Polytechnic Institute
Worcester, MA 01609

Abstract

Time is essential in modeling the constantly changing world. However, conventional database systems represent the real world with only a tenseless snapshot that is inadequate for many applications where facts and data need to be interpreted in the context of time. In this work we examine some of the issues applicable to temporal modeling, including examples which illustrate the demands imposed by database management systems for global change research. The main contribution of this paper is the proposal of a framework for managing temporal information in scientific databases. The characteristics of this model are: (1) Time is treated as a basic data type and property. (2) Attribute stamping is adopted whereby the granularity of time can be defined by the user. (3) Time is treated explicitly in the query language. (4) A helical view of time is used for measuring the distance between two time points, and (5) the implications of temporal analysis are addressed.

I. Introduction

Time is essential in modeling the constantly changing world. However, conventional database systems represent the real world with only a tenseless snapshot—the current status. This is inadequate for those applications where time is an indispensable part of the information system, for example, medical information systems, sophisticated MIS tools, and information systems for global change research [27]. In these systems, facts and data need to be interpreted in the context of time.

The need to provide temporal support in data base management systems (DBMS) has been recognized for at least a decade. Four bibliographies appearing in journals give a comprehensive survey of the work that has been done in this area [8, 17, 30, 31]. However, research on temporal databases is far from mature. Up to now, no temporal data models and query languages have gained wide acceptance and no commercial systems are available. Much work remains to be done on both theoretical and implementation-related aspects of temporal databases.

One important application of temporal databases is in Geographic Information Systems for Global Change Research (GCIS*). Temporal support is necessary for a GCIS to answer historical queries,

*This work was supported by the National Science Foundation under grant IRI-9116988 and the Worcester Polytechnic Institute Research Development Council under grant 4-29283.

*We use the term Global Change Information Systems to refer to such databases.

trace and analyze changes, make predications, and perform planning. A general discussion of some requirements and research issues of temporal GISs can be found in [4, 6, 16].

We have outlined a system to provide support for both spatial and temporal analysis of global change in [14]. An important requirement imposed on the model of time was that it be compatible with an object-oriented system design, although the precise model of time to be used was left unspecified. In the present work we examine some of the other issues applicable to temporal modeling, including examples which illustrate the demands imposed by database management systems for global change research.

In Section II we briefly review some problems related to temporal databases. Then, we overview several proposed temporal database systems and evaluate them with respect to their applicability to GCIS. In Section III we present our main contribution, a framework for managing temporal information in scientific databases. We conclude this paper in Section IV with a discussion of some important issues and on-going research.

II. Overview of Temporal Database Issues

A. Data Model

The understandings of time differs between people and different societies. This has a direct influence on the ways temporal support is provided in DBMSs. Some of the questions are: (1) Is time a point or an interval? (2) Is time discrete or dense (complete)? (3) Is time linear or branching? Discrete means that there are only a finite number of points between two time points, i.e., a specific granularity must be assumed. Dense means that between any two points there lies a third one. Complete means that if a series of points is bounded from above by another point, then there is a point that is the least upper bound of that series. (Completeness is the property that distinguishes real from rational numbers; both are dense, but the reals are complete while the rationals are not.)

There are no correct and incorrect answers to the above questions. We can choose the appropriate assumption depending on the application. In most temporal database systems the discrete and linear time assumptions are adopted, and three answers exist to the first question: time as a point, time as an interval, and mixed. Branching time is adopted in version control in CAD and engineering databases.

It is common to think of time as another dimension in the real world, in which we have objects and their attributes as two dimensions and the third dimension is the history of the object. In [27] the real world time (or *valid time*) and system time (or *transaction time*) are distinguished, resulting in a four-dimensional view. Four types of databases are defined based on the type of support for time:

1. Snapshot databases where neither valid time nor transaction time are supported.

2. Historical databases where only valid time is supported.

3. Rollback databases where only transaction time is supported.

4. Temporal databases where both valid time and transaction time are supported.

Usually temporal support in a database system is provided by extending the existing data model. In the relational paradigm, the three- or four- dimensional view is reduced to two dimensions by adding some invisible time attributes on the relations, which is called time stamping. According to the concept of time adopted, a point (instant) or an interval, there are four possible approaches: instant-stamping of tuples [5], interval-stamping of tuples [15, 20, 28], instant-stamping of attributes [9], and interval-stamping of attributes [11]. A similar method is also adopted in extending the E-R model [10].

In [22, 25] a temporal data model which is independent of the existing data models was presented. This model is called TSC (Time Sequence Collection). In TSC, a temporal data value is defined for some object (e.g., a person) at a certain time point (e.g., March 1986) for some attribute of that object (e.g., salary). Thus, a temporal data value is a triplet $< s, t, a >$, where s is the surrogate, t is the time, and a is the attribute value. The representation of TSC in the relational environment is discussed in [23], where the approach of instance-stamping of tuples is adopted.

B. Query Languages

The support of temporal data in database systems requires extending the query language to be able to manipulate temporal data. The most common way is to add temporal ingredients to the SQL or QUEL languages. There are wide variations in the constructs and forms of temporal query languages. The capabilities of these query languages also differ a great deal, ranging from supporting only basic queries to full support of querying, update, and aggregation. This situation arises from the fact that there is no accepted theoretical foundation for temporal databases, although some temporal algebra and temporal calculi have been proposed [5, 7, 9, 11, 33, 35]. Other problems are: What are the primitives for temporal operations? Should temporal reasoning be supported in the query language? If so, how much should be supported?

[28] compares temporal query languages based on some of the selected properties, such as retrieval semantics provided, historical queries, rollback, and implementability demonstrated. The issue of temporal query languages remains one of the great challenges in temporal database research. In the Appendix, we provide a (non-exhaustive) list of primitive temporal operators. This list includes previously proposed operators as well as new operators to extend database query languages with temporal support.

C. Query Optimization and Physical Design

The data in temporal databases are typically append-only. Because we want to capture the whole history of an object, nothing is deleted and updates result in the creation of objects with new time stamps. This scheme has a high overhead in terms of space. As a result, query processing will also suffer. Furthermore, one expects that many queries in a temporal database will need data ordered by time. Temporal inferencing may also be included, resulting in different temporal query patterns from that of conventional database queries.

The performance of temporal databases developed thus far is poor and new storage structures and access methods are needed to obtain adequate performance in those systems. Some of these issues have been addressed by [2, 13, 19].

D. Temporal Databases—Present Work and Evaluation

In the last decade, several temporal database management systems have been developed. Almost all these systems are extensions of earlier relational database systems [18]. These systems can be distinguished by the data model, query language, and underlying system used for implementation. We briefly describe three such systems: TQuel [28], HSQL [20], and TEER-GORDAS [10].

TQuel

TQuel bases the temporal database on the snapshot relational model, with time appearing as additional attributes. In this approach, the logic of the model does not incorporate time at all; instead the query language processor must translate queries and updates involving time into retrieval and modifications on the underlying snapshot relation.

TQuel has adopted the approach of interval-stamping of tuples in incorporating temporal information. Four invisible time attributes are added to capture valid time (FROM, TO) and transaction time (START, STOP). The time granularity is assumed to be constant in TQuel, e.g., one month granularity. Because every tuple has only one time stamp, every change or update to a tuple will introduce a new tuple in the relation. It should be noted that the information for the same object is scattered in several tuples (this may seem to be an unnatural representation). Attributes such as date-of-birth in an employee tuple are identified as user-defined time, and cannot be supported in TQuel.

TQuel extends the QUEL query language with three additional components: WHEN, VALID and AS-OF. The WHEN clause is analogous to QUEL's WHERE clause; it is used to select tuples based on temporal predicates using an implicit time attribute. The VALID clause is used to project time attributes, and the AS-OF clause is used to express rollback.

TQuel has been implemented on an Ingres DBMS by reducing TQuel statements to QUEL statements. This implementation is simple but inefficient. The performance degraded rapidly as information was added to the database.

HSQL

HSQL is also an extension of the relational model. Time in HSQL is assumed discrete and can be either an interval or a point (instance). There are six time granularities: year, month, day, hour, minute and second. Ways to deal with the different time granularities in queries are given. The idea is to convert one time granularity to another when necessary. In HSQL, one can compare time with different granularities. If two time intervals involved in an operation have different granularities, the coarser interval is converted into the finer one. HSQL also includes functions for explicit conversion of granularities. The function UPTO converts a time value to a coarser granularity by truncating, while the function INTERVAL is used to expand a time instant into a time interval of finer granularity.

Two invisible time attributes (FROM, TO) are added to the tables. Therefore, only valid time is supported. The temporal operators in HSQL are much richer than TQuel, including comparisons of time instants and intervals, and operations for conversion between time instants and intervals and between different time granularities. An operation to relate concurrent data from two or more relations is also provided, which is called a concurrent product.

HSQL expands the SQL query language by adding two new ingredients: FROMTIME (TOTIME) clause, which is used to extract a time slice, and EXPAND BY, which is used to convert the time-stamping of the tuples into the required time granularity. The time predicates can also be in a WHERE clause, and time attributes can be in SELECT and GROUP-BY clauses, which means there is no difference between time attributes and ordinary attributes at the conceptual and language levels.

Because HSQL has adopted the same approach as TQuel for time stamping, the drawbacks of TQuel in representing temporal data also appear in HSQL.

HSQL is partially implemented. The run-time system of HSQL can interpret historical operations. The implementation of a practical HSQL processor is underway.

TEER-GORDAS

TEER-GORDAS is an extension of the Enhanced Entity-Relationship data model. Time in TEER-GORDAS is discrete and can be either an interval or a point. The basic construct adopted is the "temporal element"—a finite set of intervals defined in [11]. Only valid time is supported.

In TEER-GORDAS, each entity has associated with it a temporal element, which is called the *life span* of that entity. The temporal value of each attribute is a partial function from the life span of the entity to the value domain of that attribute. The life span of the relationship is defined as the intersection of the life spans of all the participating entities.

GORDAS, a data manipulation language (DML) on the EE-R model, is extended to contain ingredients that specify temporal selection and projection operations. Here, the temporal projection is actually an operation to extract a time slice. Aggregation functions are also extended to the time domain.

Because the basic temporal construct is a temporal element, the temporal operators are those defined on a set. However, the predicates used in the temporal selection can only be equal, contains and their negations. No implementation is mentioned for TEER-GORDAS.

E. Some Comments on the Above Three Systems

The above systems are representative of the endeavors to provide temporal support in a database management system. All of these systems are endowed with the capability to record time in the database and make queries on temporal information.

The time concept they adopted is discrete, with time being an interval or a finite set of intervals. They usually have some fixed time units, but the granularity problems are only treated in HSQL. In a GIS there are often situations where some data is missing in a time series, and interpolation may be used to derive them from the neighboring time. In this case, a type of dense time is needed.

Each of the above systems extends the query language to the time domain by adding ingredients to include temporal operators, more or less, but none has been analyzed to determine the expressiveness of the query language with respect to time. The primitives for a temporal predicate are chosen in an ad hoc manner. There is no justification for the temporal operators set in the query language. None has provided the facility for temporal analysis (described in Section III), which is especially important for change analysis in GIS.

There is a problem with the semantics of valid time (or lifespan) in the above systems. It is up to the user to define what should be the valid time for a particular object, and only one valid time can be defined. When different types of time need to be recorded, it is difficult to choose among them. The worst condition is that time other than valid time cannot be dealt with properly within the systems (e.g; the date-of-birth attribute in a employee relation), because they can only be treated as user-defined time, and no system support is provided. Consider the publishing domain: there are at least three meaningful time-stamps associated with each article published in a journal (submission time, reviewing time and acceptance time). Which should we take as the valid time for that article? Any choice will leave the other two untreated, unless we semantically tie them together.

Another common problem with these systems is poor performance [22]. New access methods and optimization techniques are needed to develop a practical temporal database system.

III. A Model of Time for GCIS

A. Data Model

In this section, we discuss a model of time which we propose and intend to implement for temporal support in a Spatio-Temporal DBMS for Global Change Research [14]. This model, although discussed as a pseudo-extension of SQL, is not meant to be a specific extension of a relational model. We follow a Non-first Normal Form view of relations (N1NF) and believe that it could be mapped into the object-oriented paradigm as well. Our research efforts focus on integrating the spatial and temporal dimensions within a formalism following the object-oriented model. We advocate extensibility and target future implementations to be extensions of systems such as POSTGRES [32].

1. Discrete Time Assumption

The field of global change research is moving from a data-poor to a data-rich environment. Furthermore, it is recognized that data in pure GIS are event-driven, whereby information is accumulated as it is made available. In global change and, more generally, in scientific environments, data can be accumulated following a regular time sampling mechanism. In both cases, some of the data may not be accurate and some may be missing [6].

Different types of time, such as valid time, transaction time, user-defined time, linear time, and branching time are listed in the literature. However, the abstract concept of time can be viewed as a line such as the real number axis.

An origin (point 0) can be defined on this line, so that we can specify other points with respect to it. In our daily life, we usually choose 0 A.D. to be the origin along the time axis, though for geological processes this is not a good choice. Although time is continuous, we usually perceive it as discrete, which means we adopt a unit to measure distance from the origin. There may be different units according to specific applications. The basic units can be year, month, day, hour, minute and second. Other units can be defined at will by specifying the ratio to these basic units, for example, week, decade, or millisecond. With this facility, we can actually simulate dense time, because there is no restriction on defining new units with a smaller or larger scale.

Time can be a point or an interval on the time axis. Concomitantly, every time object should have a corresponding point or interval on that line. Therefore, now, tomorrow, last month, and five years ago are time. On the other hand, quantities such as seven days or fifteen minutes are not locations on the time axis, but are magnitudes of time intervals. We refer to these as *durations* (in the Appendix, we propose operators to handle durations in the query language).

2. Time as a Property and Basic Data Type

To provide temporal support in scientific databases, we propose to treat time as a basic property, which can be related to an entity, an attribute, a relationship or an operator. The system will take time as a basic data type with special operations. It is up to the user to interpret the semantics of time in specific applications.

- A temporal object is defined as a surrogate S with a set of attribute values. These attributes are divided into three groups: temporal attributes, non-temporal attributes and time attributes. For every temporal attribute, a time stamp is associated with it. The value of a temporal attribute is a set of <value, time> pairs. Non-temporal attributes are conventional attributes and time attributes are attributes with time as their data type.
- Every time stamp is associated with a time granularity, which can be system-defined or user-defined. We assume there is no overlap in time for a temporal attribute.
- It should be noted that we do not identify time as valid time or any other kind of time. We think the semantics should be interpreted by the user in accordance with the application.

This method of modeling is similar to attribute stamping. The resultant relations may be N1NF. The existence of time attributes also bears resemblance to tuple stamping. This consideration is derived from our belief that time can be a property of an entity, attribute, or relationship. We should provide support for it at the attribute and entity level.

Tuple stamping and attribute stamping each have their own advantages. For tuple stamping, the history of a single entity is represented with multiple tuples and every change in the value of a temporal attribute will result in a new tuple. *Time normalization* is suggested in [20] to deal with the problem of varying rates of change in temporal attributes. However, this decomposition process is not consistent with the general normalization principle. With the information of an object scattered in multiple tuples, the process of querying about the history of an object becomes more complex in tuple stamping. On the other hand, for queries with a concurrent condition of multiple temporal attributes, the query process will be more complex in attribute sampling. However, it has been shown in [2] that these two modeling methods are equivalent in terms of their information content.

B. The Query Language and Processor

The query language has not taken its final shape yet. We present some of the ingredients of the language and use two different applications to illustrate some of the ideas. The first application is from GIS and is drawn from the cadastral example in [6]. The second application is related to global change and is meant to illustrate some of the inherent differences between the two fields.

Unlike some of the previously proposed systems, in which the time attributes are implicit to the user, we use temporal attributes and time attributes which are explicit to the user. Whenever a temporal attribute or time attribute is involved in a query, the user can specify temporal predicates to choose the specific time slice, or he/she can omit it if the most recent snapshot is desired. Under this condition, temporal predicates are no different than ordinary predicates.

We explicitly distinguish a *temporal retrieval* from *temporal analysis*. A temporal retrieval is a query whose result can be directly retrieved from the database. A temporal analysis is a query in which some processing needs to be done on the result of the retrieval to give the required result. For example, the query "What was the population of Bonn when it became the capital of Germany?" is a temporal retrieval, while the query "Which area had the greatest change in population during the period of 1949 to 1979?" is temporal analysis. Therefore, the query can be divided into two parts: retrieval and analysis. The benefit of partitioning temporal queries into temporal retrieval and temporal analysis is that on the one hand we have a relatively simple and stable retrieval module, thus making the implementation and optimization easier, and on the other hand, we can have a more flexible analysis layer with which the user can define new analysis methods when necessary.

The query processor is divided into two layers in accordance with the above partition, the first layer is the (temporal) information retrieval or query, the second is (temporal) information analysis. Within this structure, each query is decomposed into basic subqueries having three steps:

1. Retrieve (or input) temporal information from the database.
2. Make temporal analysis and inferences based on the result of the retrieved data.
3. Output the result (which can be storing the result back, displaying it, or other means).

For example, the first step of the above query is: retrieve the population in each year of each area during the period of 1949 to 1979. The second step is: calculate the difference of the population in the sequences and find the area with maximum difference. Finally, display the results.

We believe that this approach is important as the inferencing or analysis part may be computationally intensive. In some (possibly most) cases, it needs to be shipped to a high performance engine for analysis. With this scheme, the problem of query processing and optimization in scientific databases becomes one of optimizing the combined query and analysis steps. This is a subject on which we are presently focusing.

1. Temporal Retrieval

A retrieval step fetches a set of data that exists in the database. The process may have two steps: selection and projection. Selection is used to choose the relevant data and projection is used to get the resultant data. (In relational databases, another important operation is join, which we will discuss later).

Corresponding to the above two steps, there are two major parts in a retrieval statement: a condition clause and an output clause. The condition clause is a Boolean expression used to specify the object to be selected and the output clause is an attribute list.

If a temporal attribute is in the output clause, we use the following convention:

1. Attribute name only ⇒ current time.
2. Attribute name followed by ALL ⇒ all history.
3. Attribute name followed by a time point or interval ⇒ specific time period.
4. Attribute name followed by RESTRICT ⇒ specific time period designated by the condition.

If a temporal attribute is in the condition clause, we use the following convention:

1. No temporal predicate ⇒ most recent snapshot.
2. Temporal predicate on the attribute is specified ⇒ specific time period.
3. ALL ⇒ all history.

The complete set of temporal operators has not yet been decided. We use some common ones in our query examples. In the rest of this section, we use some examples from a cadastral system [6], which is a type of GIS system, to illustrate how to construct a temporal retrieval statement. The relations are as follows:

```
OWNER(SSN, name, date-of-birth, profession)
PARCEL(parcel_no, area, class, price, owner)

Where   SSN, name, and parcel_no are non-temporal data
        date-of-birth is a time attribute
        profession, area, class, price, and owner are temporal attributes
```

In practice, every parcel is also associated with some spatial information. (e.g., location, boundary) which we will ignore for simplicity[†].

An object(tuple) in the `OWNER` or `PARCEL` relations might look like:

OWNER

SSN	name	date-of-birth	profession
024-71-1858	Smith	01/06/1949	<manager, (03/90, now)> <sales-rep, (09/74, 02/90)> <salesman, (07/71, 08/74)>

PARCEL

parcel_no	area	class	price	owner
831-5004	3.5	<commercial, (07/89, now)> <residential, (01/60, 06/89)>	<260k, 1991> <257k, 1990> <256k, 1989> <254k, 1988>	<028-51-2323, (02/88, now)> <024-71-1850), (01/60, 01/88)>

In the following examples, we use `V(attribute)` to denote the value of a temporal attribute and `T(attribute)` to denote the time stamp of that attribute. A temporal predicate is in the form `T(attribute) rel-op Time_constant`.

- **Simple temporal query**:

```
EXAMPLE 1. Retrieve the parcel_no and the price history of parcels
which are now worth more than 230k.

SELECT parcel_no, price ALL
FROM PARCEL
WHERE V(price)>230k
```

- **Complex temporal query**:

```
EXAMPLE 2. Retrieve the parcel_no and current price of the parcels
which used to be commercial and whose price was less than 20k at that time.

SELECT parcel_no, price
FROM PARCEL
WHERE V(class)='commercial' AND T(class) overlap T(price)
        AND V(price)<20k

EXAMPLE 3. Retrieve the parcels which were the same class as parcel 831-5144,
whose area was smaller but whose price was higher.

SELECT PARCEL.parcel_no
FROM PARCEL X
WHERE X.parcel_no='831-5144' AND V(PARCEL.class)=V(X.class)
        AND PARCEL.area<X.area AND V(PARCEL.price)<V(X.price)
        AND (T(PARCEL.class) intersect T(X.class)) overlap
            (T(PARCEL.price) intersect T(X.price))
```

[†] We are studying the spatial dimension as well. Our objective is to integrate the spatial and temporal dimensions into a Spatio-Temporal DBMS for Global Change Research.

- **Temporal Join**:

  The complexity of a temporal join is caused by the difference in the semantic constraint that the join result is required to satisfy. There are different definitions of a temporal join. The most general form is a join that involves both non-temporal and temporal predicates. In most cases the temporal predicate is `T1 overlap T2`, i.e the time interval between joined tuples must intersect; while others may have a temporal join based on the union of time interval [13, 18]. Both kinds of definition may be used in practice. Examples 4 and 5 are examples of such semantic differences.

```
EXAMPLE 4. Retrieve the classes of parcels which were owned by professors
sometime in 1960.

SELECT class RESTRICT
FROM OWNER, PARCEL
WHERE OWNER.profession='professor' AND
      OWNER.ssn=PARCEL.owner AND
      (T(OWNER.profession) intersect T(PARCEL.class))
        overlap (01/60, 12/60)

EXAMPLE 5. Retrieve the class history of parcels once owned by professors
which were commercial in 1980.

SELECT class
FROM   OWNER, PARCEL
WHERE  OWNER.profession = 'professor' AND
       V(PARCEL.class) = 'commercial' AND
       T(PARCEL.class) overlap (01/80, 12/80) AND
       OWNER.ssn = PARCEL.owner
```

  Because there is no particular constraint on the join result, different joins can be simulated by specifying appropriate join condition(s) and target list.

2. Temporal Analysis

The above examples, although drawn from GIS applications, are still in the domain of traditional query systems. However, the query patterns are quite different for scientific databases. In scientific databases the queries usually are not about one object, but about groups of objects with specific features, and the query conditions are not so complex as those above. However, analysis may be needed before the results of the queries can be useful.

Most traditional query languages provide aggregate functions such as AVG, SUM, COUNT, MAX and MIN. We can define the corresponding functions in the time domain [10]. Other useful functions might be FIRST and LAST. Some other useful analysis methods based on time are suggested in [21]. However, we feel that a system which can support temporal analysis should be extensible, as one cannot anticipate all possible temporal analysis constructs which might be needed in scientific databases. Furthermore, some of the queries that include temporal analysis might be difficult to express in a single statement [35].

Assume we have collected data on carbon dioxide and temperature in some places in the following relation:

```
CO2-TEMP(location, CO2, temperature, date)

Where   location, CO2, temperature are nontemporal attributes
        date is a time attribute, with granularity being day.
```

Typical queries might be:

```
EXAMPLE 6. Calculate the rate of increase of CO2 concentration from 1986 to
1989 for every location.

SELECT location, change-rate(CO2 RESTRICT)
FROM CO2-TEMP
WHERE date IN (1986, 1989)
```

Here the change-rate is an analysis operator, and the processing might include: average the CO2 of every day to obtain yearly data and calculate the change rate between consecutive years.

```
EXAMPLE 7. Retrieve the locations where the annual average temperature increased
the most in the last decade.
```

This query should be broken down in multiple query/analysis subqueries. A tentative scheme to complete this complex query is:

1. For each location retrieve the temperature data, and compute the average yearly temperature for the last 10 years. Store the results in a temporary relation.
2. For each location retrieve and compute the yearly rate of change of the temperature.
3. Finally, compute the maximum and report the location(s).

This process can be actually a series of pipelined processes each of them consisting of a subquery of the form: input, analyze, output.

An even more complicated scientific query, which requires the extension of the language with new operators, can be illustrated with the following example:

```
EXAMPLE 8. What is the relationship between the change in CO2 and temperature
in Hawaii from 1970 to 1990.
```

In this case, a new correlation operator is needed. This operator will be applied to two groups of data in a time series.

Compared with conventional applications, temporal inferencing and analysis play a more important role in scientific database systems. This is one of the foci of our research.

3. Dealing with Different Time Granularities

Different temporal attributes may have different time granularities. When the temporal predicate involves different time granularities, we can always convert the coarser one into the finer one. This is a simplification of the actual situation. More elaborate work which deals with time granularity is reported in [36].

Functions UPTO (fine to coarse) and DOWNTO (coarse to fine) can be provided for explicit conversion of time granularity. Another useful conversion function is ELAPSE, which gives the time units in a time interval. It can be used in expressing such a query as:

```
EXAMPLE 9. Find the parcels that have been for commercial use for more
than two years.

SELECT parcel_no
FROM PARCEL
WHERE V(class)='commercial' AND T(class) contains NOW
        AND UPTO YEARS (ELAPSE T(class)) >2
```

t

Figure 1: a Helical model of time. Time points that are far apart on the time line can be semantically closer.

4. Interpolation in Time

In scientific databases, the query data might not exist in the database. However, if we have some knowledge about the properties of the data, interpolation methods may be used to obtain the needed data. For example, suppose we did not record the rainfall for every day in a year. We can still deduce the rainfall for a specific day by taking the average of the data recorded before and after that day. Or we could derive the curve for a sequence of days spanning an interval including the missing measurements, and interpolate the missing values. An error term may need to be attached to deduced data if appropriate.

It should be noted that interpolation is not always applicable. For example, if we have records of rainfall for spring and autumn, it is meaningless to obtain the rainfall for summer by taking averages or other interpolation methods, because we may get much rain in spring and autumn, but no rain in summer.

In [21], four types of time sequences are defined, i.e., step-wise constant, continuous, discrete, and user-defined types. For the step-wise constant type, a single value spans a given time interval. Interpolation is not needed. With the continuous type, the value is a continuous function of time, and a curve fitting function can be used for interpolation. For the discrete type, the value at one time point is not related to others and the value cannot be interpolated. For the user-defined type, the value can be computed based on a user-defined interpolation function. For the above rainfall example, we may interpolate the rainfall in summer from the summer rainfall in neighboring years because the rainfall exhibits a cyclic property.

This raises an interesting perspective on time. First, we note that it is only possible to perform temporal interpolation if the time for which data are desired is "close" to times for which data are known. By "close" we mean semantically close, and as the above examples illustrate, closeness may not be equivalent to Euclidian distance along the time line. For the purpose of rainfall interpolation, time points a year apart are closer than points three months apart. This leads to a helical view of time, in which the time line winds upward with a period of one year (Figure 1).

The helical model of time operates at other scales as well. At finer scales, some patterns of human activity suggest a helix with a period of seven days, other activities follow a diurnal rhythm. At coarser scales, the 11-year sunspot cycle may be useful for modeling time for certain climatological analyses. We believe that the helical view of time merits further investigation and plan to do so.

IV. Conclusions and Future Work

In this paper, we gave an overview of our present work and some issues in temporal database systems. A framework for a data model and query language was also proposed to provide temporal support in a spatio-temporal DBMS for scientific databases. The main features of the proposal are:

1. Time is treated as a basic data type and property. We do not distinguish valid time, user-defined time, etc.; the semantics of time is decided by the user. Time can be a property of an entity,

relationship or attribute, which will provide full support for temporal data (in previous systems, user-defined time was not supported.)

2. Attribute stamping is adopted. The time granularity can be defined by the user and incorporated into the system. Time interpolation is also considered.

3. Time is treated explicitly in the query language. There is no difference between a temporal predicate and an ordinary predicate, thus providing a uniform way to express queries on temporal and non-temporal data.

4. Temporal analysis is considered. Dividing temporal queries into temporal retrieval and temporal analysis can simplify the system structure.

5. Propose a helical view of time for the determination of time distance.

However, much work needs to be done before an actual implementation; many problems must be solved. Some of them are:

1. The definition of a temporal query language: most of the current systems have extended SQL or QUEL to include temporal ingredients. However, these extensions have been done in an ad hoc manner, and no investigation has been done on the temporal primitives set and basic temporal constructs.

2. Temporal analysis: it is clear that some degree of temporal analysis should be supported in the temporal query language. But how much support should be provided? Although a temporal relational algebra and calculus (both are expressed in temporal logic) have been proposed as the bases for temporal relational completeness, they are not widely accepted.

   The partition of a temporal query into retrieval and analysis can solve this problem to a certain degree, because new (analysis) methods can be defined by the user and incorporated into the system. However, for an arbitrary temporal query, it is not straightforward or even possible to give a distinction between temporal inferencing and temporal retrieval. Temporal analysis might be included in the selection condition of temporal retrieval. For example, in a company database, consider the query "Find the department in which the employee got the largest increase in salary in the last year." If we treat temporal analysis as operators that can be extended, then the problem of interfacing them with temporal retrieval needs to be addressed.

3. Query optimization and access methods: one of the critical problems in implementing a temporal database system is performance. This may partially account for the absence of commercial temporal database systems at this time. Little work has been done in this area.

   In our research, we are studying query/analysis processing techniques for query optimization of spatio-temporal DBMSs. We expect that our efforts may be used as a starting point for such systems.

Acknowledgements

The authors would like to thank Yuhong Zhang and Yu Zhou for their contributions to this project.

References

1. M.E. Adiba, "Histories and Versions for Multimedia Complex Objects," *Database Engineering*, Vol. 7, pp. 181–188, 1988.

2. I. Ahn, "Towards An Implementation of Database Management Systems with Temporal Support," *Proc. IEEE Conf. Data Engineering*, pp. 374–381, 1986.

3. J.F. Allen "Maintaining Knowledge about Temporal Intervals," *CACM*, Vol. 26, No. 11, pp. 832-843, Nov. 1983.

4. K. Al-Taha, R. Barrera, "Temporal Data and GIS: An Overview," *Proc. Conf. GIS/LIS*, pp. 244–254, Nov. 1990.

5. G. Ariav, "A Temporally Oriented Data Model," *ACM TODS*, Vol. 11, No. 4, pp. 499–527, Dec. 1986.

6. R. Barrera, A. Frank, K. Al-Taha, "Temporal Relations in Geographic Information System: A Workshop at the University of Maine," *SIGMOD Record*, Vol. 20, No. 3, pp. 85–91, Sept. 1991.

7. M.A. Bassiouni, M. Llewellyn, "Handling Time in Query Languages for Statistical and Scientific Database Management," *Proc. 4th Int. Working Conf. SSDBM*, appearing in *Lecture Notes in Computer Science 339*, Springer–Verlag, pp. 105–119, June 1988.

8. A. Bolour, T.L. Anderson, L.J. Dekeyser, H.K.T. Wong, "The Role of Time in Information Processing: A Survey," *SIGMOD Record*, Vol. 12, No. 3, pp. 27–50, 1982.

9. J. Clifford, A.U. Tansel, "On an Algebra for Historical Relational Database: Two Views," *Proc. ACM SIGMOD Conf.*, pp. 247–265, 1985.

10. R. Elmasri, G.T.J. Wuu, "A Temporal Model and Query Language for ER Databases," *Proc. IEEE Conf. Data Engineering*, pp. 76–83, 1990.

11. S.K. Gardia, "A Homogeneous Relational Model and Query Languages for Temporal Databases," *ACM TODS*, Vol. 13, No. 4, pp. 418–448, Dec. 1988.

12. S.K. Gardia, "The Role of Temporal Elements in Temporal Databases," *Database Engineering*, Vol. 7, pp. 197–203, 1988.

13. H. Gunadhi, A. Segev, "A Framework for Query Optimization in Temporal Databases," *Proc. 5th. Int. Working Conf. SSDM*, appearing in *Lecture Notes in Computer Science 420*, Springer–Verlag, pp. 131–147, 1990.

14. N.I. Hachem, M.A. Gennert, M.O. Ward, "A DBMS Architecture for Global Change Research," *Proc. ISY Conf. Earth and Space Science*, Feb. 1992, to appear.

15. W. Kafer, N. Ritter, H. Schoning, "Support for Temporal Data by Complex Objects," *Proc. Conf. Very Large Databases*, pp. 24–35, Aug. 1990.

16. G. Langran, "A review of temporal database research and its use in GIS applications," *Int. J. GIS*, Vol. 3, No. 3, pp. 215–232, 1989.

17. E. McKenzie, "Bibliography: Temporal Databases," *SIGMOD Record*, Vol. 15, No. 4, pp. 40–52, Dec. 1986.

18. L.E. McKenzie, Jr., R. Snodgrass "Evaluation of Relational Algebras Incorporating the Time Dimension in Databases," *ACM Computing Surveys*, Vol. 23, No. 4, pp. 501-543, Dec. 1991.

19. D. Rotem, A. Segev, "Physical Organization of Temporal Data," *Proc. IEEE Conf. Data Engineering*, pp. 547-553, Feb. 1987.

20. N.L. Sarda, "Extensions to SQL for Historical Databases," *IEEE Trans. Knowledge & Data Engineering*, Vol. 2, No. 2, pp. 220–230, June 1990.

21. A. Segev, A. Shoshani, "Logical Modeling of Temporal Data" *Proc. ACM SIGMOD Conf.*, pp. 454–466, 1987.

22. A. Segev, A. Shoshani, "Functionality of Temporal Data Models and Physical Design Implications," *Database Engineering*, Vol. 7, pp. 216–223, 1988.

23. A. Segev, A. Shoshani, "The Representation of Temporal Data Model in the Relational Environment," *Proc. 4th Int. Working Conf. SSDM*, appearing in *Lecture Notes in Computer Science 339*, Springer–Verlag, pp. 39–61, June 1988.

24. Y. Shoham, *Reasoning about Change*, MIT Press, 1988.

25. A. Shoshani, K. Kawagoe, "Temporal Data Management," *Proc. Conf. Very Large Databases*, pp. 205–215, Aug. 1986.

26. A. Silberschatz, M. Stonebraker, J.D. Ullman, "Database Systems: Achievements and Opportunities," *SIGMOD Record*, Vol. 19, No. 4, pp. 6–22, Dec. 1990.

27. R. Snodgrass, I. Ahn, "Temporal Databases," *IEEE Computer*, Vol. 19, No. 9, pp. 35–42, Sept. 1986.

28. R. Snodgrass, "The Temporal Query Language TQuel," *ACM TODS*, Vol. 12, No. 2, pp. 247–298, June 1987.

29. R. Snodgrass, "Temporal Databases: Status and Research Directions," *SIGMOD Record*, Vol. 19, No. 4, pp. 83–89, Dec. 1990.

30. M.D. Soo, "Bibliography on Temporal Databases," *SIGMOD Record*, Vol. 20, No. 1, pp. 14–23, Mar. 1991.

31. R.B. Stam, R. Snodgrass, "A Bibliography on Temporal Databases," *Database Engineering*, Vol. 7, pp. 231–239, 1988.

32. M. Stonebraker, G. Kemnitz, "The POSTGRES Next-Generation Database Management System," *Comm. ACM*, Vol. 34, No. 10, pp. 78–92, Oct. 1991.

33. A.U. Tansel, "Non First Normal Temporal Relational Model," *Database Engineering*, Vol. 7, pp. 224–230, 1988.

34. D. Tasker, "An Entity/Relationship View of Time," *Proc. 6th Int. Conf. on Entity-Relationship Approach*, pp. 234–247, 1988.

35. A. Tuzhilin, J. Clifford, "A Temporal Relational Algebra as a Basis for Temporal Relational Completeness," *Proc. Conf. Very Large Databases*, pp. 13–23, Aug. 1990.

36. G. Wiederhold, S. Jajodia, W. Litwin, "Dealing with Granularity of Time in Temporal Databases," *in Advanced Information Systems Engineering, Proc. 3rd. Int. Conf. CAiSE'91*, pp. 124–140, 1991.

Appendix: Temporal Operator List

The temporal operations depend on the concept of time, whether it be continuous or discrete. They are also determined by the underlying data model adopted to incorporate temporal information. In the following, we list temporal operators based on the discrete time concept. Three basic categories of time are assumed: `POINT`, `INTERVAL` and `DURATION`. A `POINT` is of primitive type `TIME` and is an absolute location along the time line. An `INTERVAL` consists of an ordered pair of points in time and is also of primitive type `TIME`. A `DURATION` is a difference of times, and so, strictly speaking, is not of primitive type `TIME`, but represents a relative displacement along the time line (i.e., the magnitude of an interval). For clarity we partition the operators into four groups: comparison, arithmetic, conversion and aggregate. The convention will be to use variables starting with **p** for a point, **d** for a duration, **i** for an interval, and **l** for a logical or boolean (true or false) value.

A. Comparison

Comparison operators require two operands of the same time category and result in a logical value.

1. Time as a point

By considering time as a point, the usual numeric comparison operators <, <=, =, >=, >, <> can be used (HSQL [20]). If the discrete assumption is adopted, it may be meaningful to specify such comparisons operators as precede, succeed and is-adjacent-to. These operations are of the form

```
l = p1 op p2
```

2. Time as an interval

The relationships between time intervals derived from [3, 4, 20] are:

```
l = i1 before i2       (i1 precedes i2)
l = i1 equal i2        (i1 = i2)
l = i1 meets i2        (i1 meets i2)
l = i1 overlaps i2     (i1 overlaps i2)
l = i1 during i2       (i2 contain i1)
l = i1 starts i2       (begin at the same time point)
l = i1 finishes i2     (end at the same time point)
```

and the reverse relationships as above. The first five relationships are from [20], while the last two are from [3]. In addition, Sarda also lists an `adjacent` relationship `i1 adjacent i2` iff `i1 meets i2 or i2 meets i1`. By comparison, there are fewer temporal predicates in TQuel. They are `precede`, `overlap` and `equal`. These operations are of the form

```
l = i1 op i2
```

3. Durations

Durations, like points, can be compared with numeric operators <, <=, =, >=, >, <>. Operands of different granularities can be converted to a common granularity explicitly or by default (to the finer of the two). These operations are of the form

```
l = d1 op d2
```

B. Arithmetic

Arithmetic operations can be defined as those whose result is also a `POINT`, `INTERVAL`, or `DURATION`.

1. Time as a point

```
p2 = p1 + d    (to shift the time point on the time axis)
d = p1 - p2    (to compute a duration)
```

2. Time as an interval

```
i3 = i1 overlap i2        (intersection)
i3 = i1 extend i2         (union, +)
i3, i4 = i1 difference i2 (subtraction results in one or two intervals)
i2, i3 = negate i1        (all except that interval)
i2 = i1 offset d          (move start and end of interval)
```

The first two are defined in TQuel and in HSQL, while we propose the addition of the last three operators.

C. Conversion

Conversion operations are used between time points and intervals. There are also some conversion operations used to coerce times to different granularities.

t

Table 1: Temporal Operators

Mapping	Example
$P \rightarrow P$	change point granularity
$P \times P \rightarrow L$	first point before second point
$P \times P \rightarrow I$	make an interval
$P \times P \rightarrow D$	compute a duration
$P \times P \rightarrow P$	compute the midpoint in time
$P \times D \rightarrow P$	offset a point
$P \times D \rightarrow I$	define an interval
$P \times I \rightarrow L$	point is within interval
$P \times I \rightarrow I$	extend interval to include point
$I \rightarrow P$	get first point in interval
$I \rightarrow I$	change interval granularity
$I \rightarrow D$	compute magnitude of interval
$I \rightarrow L$	check if interval is empty
$I \times I \rightarrow L$	first interval contains second interval
$I \times I \rightarrow I$	intersection between intervals
$I \times I \rightarrow D$	duration of union
$I \times I \rightarrow P$	first common point
$D \rightarrow D$	change duration granularity
$D \times D \rightarrow L$	durations are equal

1. Point to interval

```
i = make_interval(p1, p2) (combine two points to form an interval t1..t2)
```

2. Interval to point

```
p = begin_of(i)
p = end_of(i)
```

Variants of these are used in TQuel to obtain the start and end points of an interval.

3. Conversion between different time units

```
p1 = coarse_to_fine(p2)
p1 = fine_to_coarse(p2, truncation/round)
```

Similar operations are available for `INTERVALS` and `DURATIONS`.

D. Summary of Basic Operators

Table 1 summarizes the basic temporal operators in terms of mappings between `POINTS` P, `DURATIONS` D, `INTERVALS` I, and `LOGICALS` L. Samples of each type are given.

E. Aggregate Operations

There are some operations we classify as aggregate, either because they include further processing on the result of the query, or they correlate multiple objects with some temporal relationships. Some of these are:

```
get first event later than point p
compute average time between events over interval i
correlate events separated by duration d over interval i
```