Temporal Modules: An Approach Toward Federated Temporal Databases

X. Sean Wang*, Sushil Jajodia*, V. S. Subrahmanian†

Abstract

In a federated database environment, different constituents of the federation may use different temporal models or physical representations for the temporal information. This paper introduces a new concept, called a *temporal module*, to resolve these mismatches among the constituents. Intuitively, a temporal module hides the implementation details of a temporal relation by exposing its information only through two *windowing functions*: a function associating each time point with a set of tuples and a function linking each tuple to a set of time points. A calculus-style language is given to form queries on temporal modules.

Temporal modules are then extended to resolve another type of mismatch among the constituents of a federation, namely, mismatch involving different time units (e.g., *month*, *week* and *day*) used in recording temporal information. Our solution to this mismatch relies on "information conversions" provided by each constituent. Specifically, a temporal module is extended to provide several "windows" to its information, each in terms of a different time unit. The first step to process a query addressed to the federation is to select suitable windows for the underlying temporal modules. In order to do so, time units are formally defined and studied. A federated temporal database model and its query language are proposed. The query language is an extension of the calculus-style language above.

1 Introduction

Storing and manipulating temporal information is important for many database applications [Tan93]. Different databases, however, may use different temporal data models to represent temporal information, different storage strategies to physically store temporal data, or different time units on which the stored temporal information is based [WJL93]. When these databases are combined to form a federation, a unifying framework is needed in order to resolve these temporal mismatches to answer queries at the federation level. The purpose of this paper is to provide such a unifying framework.

In recent years, there have been several efforts to extend the relational model to incorporate the time dimension. These extensions generally take one of two approaches. In the first approach (e.g., [MS87, SS87, NA89]) each relation scheme is extended to include some time attributes (see Figure 1a). Values of these time attributes come from some fixed temporal domain. Given a tuple, the values of these time attributes indicate when the information (i.e., the remaining values of the tuple) is valid.[1] Unlike the first approach that stays within first normal form relations, the second approach (e.g., [CT85, Gad88]) deals with non-first normal form relations. Under the second approach, a timestamp is associated with each attribute value of a tuple (see Figure 1b). The intuition is that the actual value of a particular attribute at a particular time point is the value whose attached timestamp contains that time point.

The above two approaches provide two quite different ways of incorporating temporal information into relational databases. The essence, however, is the same: both are modeling the situation where a fact (or a tuple in relational database terminology) is valid only at certain time points. In other words, a temporal relation under either approach gives a set of pairs (f, T), where f is a fact (i.e., a tuple) and T is a set of time points during which the fact is valid. For example, consider the two temporal relations in Figure 1. Intuitively, each tuple in Figure 1a gives an interval of time during which the attached fact (i.e., a tuple) is valid. Clearly, when a time interval is viewed as a set of time points, the relation in Figure 1a contains three *(fact, time points)* pairs. Although the relation in Figure 1b consists of a single tuple and uses time intervals to timestamp each domain value, it gives the following four *(fact, time points)* pairs:

*Depatment of Information and Software Systems Engineering, George Mason University, Fairfax, VA 22030. The work of these two authors is partly supported by a DARPA grant, administered by the Office of Naval Research under grant number N0014-92-J-4038.

†Department of Computer Science, University of Maryland, College Park, MD 29742. The work of this author is partly supported by the following grants: ARO grant DAAL-03092-G-0225, AFOSR grant F49620-93-1-0065 and NSF grant IRI-9109755.

[1] In this paper we deal only with the *valid* time, and not the *transaction* time [SA85].

Name	Rank	From	To
Jane	Assistant	9-71	12-76
Jane	Associate	12-76	11-80
Jane	Full	11-80	∞

a. A temporal relation from [Sno84].

Name	Salary	Dept
[11,61) John	[11,50) 15K [50,55) 20K [55,61) 25K	[11,45) Toys [45,61) Shoes

b. A temporal Relation from [Gad88].

Figure 1: Two approaches to temporal relations.

$(\langle$John, 15K, Toys$\rangle$, [11,45)),
$(\langle$John, 15K, Shoes$\rangle$, [45,50)),
$(\langle$John, 20K, Shoes$\rangle$, [50,55)),
$(\langle$John, 25K, Shoes$\rangle$, [55,61)).

The above observation leads us to propose a new concept of a *temporal module*, which serves as a unifying framework to resolve mismatches among various methods of storing temporal data. A temporal module consists of a relation scheme together with two *windowing* functions: a function that associates each time point to a set of tuples and a function that links each tuple with a set of time points. The windowing functions allow us to manipulate the temporal aspects of the relation scheme. A temporal module calculus (called Tm-calculus) is introduced to form queries on temporal modules. It is shown that the Tm-calculus can express most of the natural temporal queries.

Temporal modules provide a higher level of abstraction than that provided by previous extensions to the relational model. A temporal module hides the details of the physical implementation of a temporal relation (whether it is stored as a first normal form or a non-first normal form relation). All information associated with the temporal relation is available only through windowing functions. Through the temporal modules, different temporal relations can be combined together to answer federation level queries, i.e., queries involving different temporal databases with different underlying temporal data models.

This higher level of abstraction also provides the flexibility of viewing the same temporal relation in terms of different temporal domains (or, different time units). Some examples of temporal domains include *week*, a domain whose values consist of the number of the week together with the year, and *year*, a domain whose values consist of simply years. A popular choice in literature for the temporal domain is the set of the natural numbers, which is interpreted as a set of linearly ordered, equally spaced time points (e.g., [CT85, GV85]). With our temporal module abstraction, we can easily deal with different temporal domains without changing the underlying temporal representation. To illustrate, suppose in an employee database, salaries of individuals are stored on a monthly basis. By simply changing the windowing functions, a temporal module of the employee salaries on a yearly basis or on a weekly basis can be provided. In other words, *information conversion* among different time units can be achieved with ease.

To study the above type of information conversion, the notion of *time units* is formally treated. It is shown that the set of all time units forms a complete lattice with respect to a *finer-than* relationship[2]. This lattice structure gives rise a formal notion of conversion: we argue that if information in terms of time unit μ_1 can be converted to that in terms of time unit μ_2, then the same information conversion procedure should provide us with the conversion from μ_1 to the greatest lower bound (glb), or the least upper bound (lub), of $\{\mu_1, \mu_2\}$ and then from the information in terms of glb or lub to that in terms of μ_2. Thus, even though there may exist infinitely many possible conversion schemes, it suffices to use the conversion through either glb or lub. For example, suppose the above salary temporal module converts salary information in terms of the time unit *month* (based on which the information is stored) to that in terms of the time unit *week*. The conversion is often achieved as follows: the salary is first converted into daily salary and then, from this daily salary to weekly salary. (Here, the time unit *day* is the glb of *month* and *week*.)

The above characteristics of temporal modules, namely that of hiding different temporal schemes and providing different views in terms of different temporal domains (or time units), yield a unifying framework that combines many different temporal databases into a federation.

The remainder of the paper is organized as follows: in Section 2, temporal modules are formally defined. In Section 3, the query language Tm-calculus is introduced and its expressive power discussed. The concept of time units, along with some of its properties, is given in Section 4. In Section 5, temporal modules are extended to represent temporal information on multiple time units and several results on conversion of information are developed. The Tm-calculus query language is then extended in Section 6 to handle temporal modules with several time units. The paper is concluded with a summary in Section 7.

2 Temporal Modules and Temporal Databases

In this section, we introduce the concept of a temporal module. A temporal module forms an abstract interface

[2] The authors wish to thank an anonymous referee for bringing our attention to [CR87], in which time units are arranged in a linear order.

between temporal relations (or other means of storing temporal data) and user queries.

Throughout this paper, $\mathbf{N}$ is used to denote the set of the non-negative integers. Also, an infinite set of *attribute names* is assumed. Attribute names are denoted by A, B, possibly subscripted. Each attribute name A is associated with an infinite *domain*, denoted $\mathbf{Dom}(A)$. Each finite subset of attribute names is called a *relation scheme* and denoted by R, possibly subscripted. A relation scheme R is usually written as a list of all the attributes in R in the form $\langle A_1, \ldots, A_k\rangle$. A *tuple* t of a relation scheme R is a mapping from the attributes of R to $\bigcup_{A\in R}\mathbf{Dom}(A)$ such that $t(A)$ is in $\mathbf{Dom}(A)$ for each A in R. The set of all tuples of R is denoted by $\mathbf{Tup}(R)$. A tuple t of the relation scheme $\langle A_1, \ldots, A_k\rangle$ is usually written as a list of values of the form $\langle v_1, \ldots, v_k\rangle$, where $v_i = t(A_i)$ for each $1 \leq i \leq k$.

We now define temporal modules. A temporal module is an "abstract object" which stores time related information. The outside world retrieves information through two functions provided by this abstract object. Formally, we have

Definition 1 A *temporal module* is a triple (R, ϕ, τ), where R is a relation scheme, ϕ a partial function, called *time-windowing function*, from $\mathbf{N}$ to $2^{\mathbf{Tup}(R)}$ and τ a partial function, called *tuple-windowing function*, from $\mathbf{Tup}(R)$ to 2^N.

Intuitively, ϕ and τ provide temporal "windows" that may be used to retrieve tuples that were true at a given point of time, or to retrieve time points during which a given tuple holds.

Consider the set of facts shown in Figure 2. A row of the form $\langle p_1, p_2, c, p_3, y\rangle$ in the table of Figure 2 says that in year y the cost at plant p_1 to produce product p_2 was c and the profit percentage was p_3. We now build two temporal modules over these facts. The first temporal module gives information about profit (hiding the cost information) of each plant in producing each product. Specifically, let the table in Figure 2 be denoted by r. Then let TM_1 be the temporal module (R_1, ϕ_1, τ_1), where[3] $R_1 = \{\texttt{Plant}, \texttt{Product}, \texttt{Profit}\}$,

$$\phi_1(y) = \pi_{\texttt{Plant},\texttt{Product},\texttt{Profit}}\sigma_{\texttt{Year}=y}(r),$$

and

$$\tau_1(\langle p_1, p_2, p_3\rangle) = \pi_{\texttt{Year}}\sigma_{\texttt{Plant}=p_1\wedge\texttt{Product}=p_2\wedge\texttt{Profit}=p_3}(r).$$

For example, $\phi_1(1990)$ gives the set consisting of the following tuples

Plant	Product	Profit
Baltimore	Digital Amplifier	15%
Denver	Rock Candy	30%

[3] Relational algebra expressions (see [Ull88]) are used in expressing functions ϕ_1 and τ_1. In general, arbitrary methods can be used to express ϕ_1 and τ_1.

and $\tau_1(\langle\texttt{Baltimore}, \texttt{DigitalAmplifier}, 15\%\rangle) = \{1990, 1991\}$.

Another temporal module we build is about the profits of the plants. Specifically, let TM_2 be the temporal module (R_2, ϕ_2, τ_2), where $R_2 = \{\texttt{Plant}, \texttt{Profit}\}$, $\phi_2(y)$ gives the set of tuples $\langle p_1, p_2\rangle$, where p_1 is a plant name and p_2 is the overall profit percentage of the plant[4] of year y, and $\tau_2(\langle p_1, p_2\rangle)$ gives the years during which plant p_1 had profit p_2.

From the above examples we see that the two "windows" of a particular temporal module, i.e., functions ϕ and τ, are two perspectives that the outside world will view the same information inside. We require that the two windows be consistent in the sense that if a tuple and a time point are "associated" through function ϕ, then they are also "associated" through τ. Formally, we have:

Definition 2 A temporal module (R, ϕ, τ) is said to be *well behaved* if

1. For each tuple t of R, t is in $\phi(i)$ for each i in $\tau(t)$, and
2. For each i in $\mathbf{N}$, i is in $\tau(t)$ for each t in $\phi(i)$.

In the remainder of the paper, all temporal modules are assumed to be well behaved.

A temporal module can be viewed as a source unit of temporal information. A temporal database usually contains a number of such source units. This leads us to the formal definition of temporal databases.

Definition 3 A *temporal database* is a set of elements, called *temporal module names*, and each temporal module name is associated with a temporal module.

For example, D is a temporal database if

$$D = \{\ \texttt{Product\_Profit},\ \texttt{Plant\_Profit}\ \},$$

and `Product_Profit` and `Plant_Profit` associate to the two temporal modules given earlier in the section, respectively.

Let D be a temporal database. For each M in D, we use R_M, ϕ_M and τ_M to denote, respectively, the first, second and third components (i.e., the relation scheme R and the functions ϕ and τ) of the temporal module associated with M. For example, we use $R_{\texttt{Product\_profit}}$, $\phi_{\texttt{Product\_Profit}}$ and $\phi_{\texttt{Plant\_Profit}}$ to denote the relation scheme R_1 and the functions ϕ_1 and ϕ_2 given earlier in this section, respectively.

[4] Note that the overall profit percentage depends on the cost of each product. For example, the total profit percentage of Baltimore plant in 1991 is 16.5%. The value of $\phi_2(y)$ can roughly be expressed by the following pseudo-SQL query:

```
SELECT Plant, Profit=SUM(Cost*Profit/100)/SUM(Cost)*100
FROM Fact_about_profits
Where Year = y
Group by Plant;
```

Plant	Product	Cost	Profit	Year
Baltimore	Portable Humidifier	$5M	10%	1991
Baltimore	Cellular CarPhone	$10M	20%	1991
Baltimore	Digital Amplifier	$2M	15%	1991
Baltimore	Digital Amplifier	$1.5M	15%	1990
Los Angeles	Paper Towel	$1M	5%	1991
Los Angeles	Portable Humidifier	$3M	11%	1991
Denver	Rock Candy	$0.5M	30%	1990

Figure 2: Facts about profits

3 Tm-Calculus

In this section, we show how the relational calculus can be adapted and extended to address queries on temporal modules. We call the resulting calculus Tm-calculus.

In this section, we assume D is a temporal database. We say that "M is a temporal module in D" if M is in D. Also as mentioned at the end of last section, we will use R_M, ϕ_M and τ_M to denote the relation scheme and the two windowing functions of the temporal module associated with M.

The calculus is defined in terms of a two-sorted first order logical language. Variable symbols are of two sorts – *time* variables which range over points of time, and *domain* variables which range over domain values. We use i, j (possibly subscripted) as time variables, and lower case roman letters (possibly subscripted) as domain variables. Also, we use $\mathbf{z}$ (possibly subscripted) as either time or domain variables. Each query in the Tm-calculus is of the form

$$\{\langle \mathbf{z}_1, \ldots, \mathbf{z}_k \rangle \mid F(\mathbf{z}_1, \ldots, \mathbf{z}_k)\},$$

where F is a formula built from atoms and a collection of operators (the precise definitions are given below).

The simplest kind of Tm-calculus formula is an *atom* defined as follows:

- For each temporal module M in TD,
  $$\langle A_1 : x_1, \ldots, A_k : x_k \rangle \in \phi_M(\mathsf{i})$$
  and
  $$\mathsf{i} \in \tau_M(\langle A_1 : x_1, \ldots, A_k : x_k \rangle)$$
  are both atoms if (i) $R_M = \langle A_1, \ldots, A_k \rangle$, (ii) each x_i, $1 \leq i \leq k$, is a domain variable or a (constant) domain value (i.e., an element in $DOM(A_i)$), and (iii) i is a time variable or an integer in $\mathbf{N}$.
- For each comparison operator θ in the set $\{<, >, \leq, \geq, =, \neq\}$, $x\theta y$, $x\theta a$, $\mathsf{i}_1\theta\mathsf{i}_2$ and $\mathsf{i}_1\theta i$ are all atoms if x and y are domain variables, a is a domain value (i.e., a in $DOM(A)$, where A appears in R_M for some temporal module M in TD), i_1 and i_2 are time variables and i is an integer in $\mathbf{N}$.

Tm-calculus formulas are defined inductively in terms of atoms and the connectives $(\wedge, \vee, \rightarrow, \neg)$ and the quantifiers $(\exists, \forall)$ in the usual way. The concept of *free variables* and *bounded formulas* of bounded formulas are also defined as usual. The atomic formula

$$\langle A_1 : x_1, \ldots, A_k : x_k \rangle \in \phi_M(\mathsf{i})$$

is assigned "true" if $\langle x_1, \ldots, x_k \rangle$ is in the set $\phi_M(\mathsf{i})$, and false otherwise. The truth value for the atomic formula

$$\mathsf{i} \in \tau_M(\langle A_1 : x_1, \ldots, A_k : x_k \rangle)$$

is given similarly. The truth value of a bounded formula is then defined as usual based on the truth value of the atomic formula.

Now a query in the Tm-calculus is of the form

$$\{\langle \mathbf{z}_1, \ldots, \mathbf{z}_n \rangle \mid F(\mathbf{z}_1, \ldots, \mathbf{z}_n)\},$$

where F is a Tm-calculus formula and $\mathbf{z}_1, \ldots, \mathbf{z}_n$ are the only free variables in F. (When there is only one free variable in F, the symbol $\langle \ldots \rangle$ will sometimes be omitted in the above query.) An *assignment* α of the variables $\mathbf{z}_1, \ldots, \mathbf{z}_n$ is a mapping such that, for each $1 \leq i \leq n$, $\alpha(\mathbf{z}_i)$ is an integer in $\mathbf{N}$ if $\mathbf{z}_i$ is a time variable, and $\alpha(\mathbf{z}_i)$ is a domain value if $\mathbf{z}_i$ is a domain variable. The answer of the above Tm-calculus query is the set of all the assignments α such that the bounded formula $F(\alpha(\mathbf{z}_1), \ldots, \alpha(\mathbf{z}_n))$ (i.e., substituting each free variable $\mathbf{z}_i$ with $\alpha(\mathbf{z}_i)$ in formula F) is true.

Before we give example queries, we define two additional temporal modules. The first temporal module we use is about managers. Specifically, let `Manager` be a temporal module. The relation scheme for this temporal module is $\langle \texttt{Plant}, \texttt{Name} \rangle$. A tuple $\langle p, n \rangle$ of this relation scheme means that n is a manager at plant p. The functions $\phi_{\texttt{Manager}}$ and $\tau_{\texttt{Manager}}$ associate such tuples to times. Another temporal module is called `Salary` and its relation scheme is $\langle \texttt{Name}, \texttt{Amount} \rangle$. A tuple $\langle n, a \rangle$ of this relation scheme means that n earns a. The functions $\phi_{\texttt{Salary}}$ and $\tau_{\texttt{Salary}}$ are the windowing functions of the temporal module. Our examples below will be concerned with the four temporal modules we have established so far, i.e., `Product_Profit`, `Plant_Profit`, `Manager` and `Salary`.

Example 1 "Who, and when, managed a plant producing a profit lower than 1%." This is expressed in the Tm-calculus as follows:

$$\{\langle p,\mathsf{i}\rangle \mid \exists p_1 \exists p_2 (\mathsf{i} \in \tau_{\mathtt{Manager}}(\langle \mathtt{Plant} : p_1, \mathtt{Name} : p\rangle) \wedge \langle \mathtt{Plant} : p_1, \mathtt{Profit} : p_2\rangle \in \phi_{\mathtt{Plant\_Profit}}(\mathsf{i}) \wedge p_2 < 1)\}.$$

Example 2 "Who, during his/her tenure as the manager of a plant, was able to have a profit of more than 19% on Cellular Car Phones after 1990?" This is most naturally expressed as the Tm-calculus query:

$$\{m \mid \exists \mathsf{i} \exists p_1 \exists p_2 (p_2 > 19 \wedge \mathsf{i} > 1990 \wedge \mathsf{i} \in \tau_{\mathtt{Product\_Profit}}(\langle \mathtt{Plant} : p_1, \mathtt{Product} : \mathtt{CellularCarPhone}, \mathtt{Profit} : p_2\rangle) \wedge \langle \mathtt{Plant} : p_1, \mathtt{Name} : m\rangle \in \phi_{\mathtt{Manager}}(\mathsf{i}))\}.$$

Thus, Tm-calculus allows us to express queries about "sometime after." For those familiar with temporal logics, temporal logic formulas of the form $\Diamond\psi$ (the formula is true at time i if ψ is true sometime after i) can be expressed in our framework as follows: assume, inductively, that we can express the statement "ψ is true at time i" in the Tm-calculus via some formula $F_\psi[i]$. Then $\Diamond\psi$ can be expressed as the query: $\{\mathsf{i} \mid \exists \mathsf{i}_1(\mathsf{i}_1 > \mathsf{i} \wedge F_\psi[\mathsf{i}_1])\}$, i.e., find those times such that F_ψ will be true after that time.

Example 3 "After when is the profit on cellular car phones always over 14%?" This is most naturally expressed as the Tm-calculus query:

$$\{\mathsf{i} \mid \forall \mathsf{i}_1(\mathsf{i}_1 > \mathsf{i} \rightarrow \forall p_1 \exists p_2 (p_2 > 14 \wedge \langle \mathtt{Plant} : p_1, \mathtt{Product} : \mathtt{CellularCarPhone}, \mathtt{Profit} : p_2\rangle \in \phi_{\mathtt{Product\_Profit}}(\mathsf{i}_1)))\}.$$

This example shows that formulas concerning "always in the future" (i.e., formulas of the form $\Box\psi$) can be dealt with in the Tm-calculus. This is also a direct consequence that $\Diamond\phi$ can be handled in our framework. Indeed, in temporal logics, we have $\Box\psi$ is equivalent to $\neg\Diamond\neg\psi$.

Example 4 "Find those managers who is now making over 200K (assuming it is now 1992) and who managed a plant some time in the past during which the plant's profit was below 1%." This is most naturally expressed in the Tm-calculus as follows:

$$\{m \mid \exists s (s > 200000 \wedge \langle \mathtt{Name} : m, \mathtt{Salary} : s\rangle \in \phi_{\mathtt{Salary}}(1992) \wedge \exists \mathsf{i}(\mathsf{i} < 1992 \wedge \exists p_1(\langle \mathtt{Plant} : p_1, \mathtt{Name} : m\rangle \in \phi_{\mathtt{Manager}}(\mathsf{i}) \wedge \exists p_2 (p < 1 \wedge \langle \mathtt{Plant} : p_1, \mathtt{Profit} : p_2\rangle \in \phi_{\mathtt{Plant\_Profit}}(\mathsf{i})))))\}.$$

In general, temporal formulas of the form $\mathbf{P}\psi$ ("was ψ true in the past?") can be immediately handled in our framework by replacing the conjunct $\mathsf{i}_1 > \mathsf{i}$ in the $\Diamond$ modality case by $\mathsf{i}_1 < \mathsf{i}$.

Example 5 "In which plant the profits on cellular car phones have always been below 22% in the past (assuming now is 1992)?" The query is expressed in the Tm-calculus as follows:

$$\{p \mid \forall \mathsf{i} \forall p_1 (\mathsf{i} < 1992 \wedge \langle \mathtt{Plant} : p, \mathtt{Product} : \mathtt{CellularCarPhone}, \mathtt{Profit} : p_1\rangle \in \phi_{\mathtt{Product\_Profit}}(\mathsf{i}) \rightarrow p_1 < 22)\}.$$

In general, queries of the form $\mathbf{H}\psi$ ("was ψ true at all times in the past?") can be handled in our framework.

Example 6 "Which plant, since its profit was more than 20%, has always had a profit more than 15%?" To express this query in the Tm-calculus, let $min_{\mathsf{i}}(\psi(\mathsf{i}))$ be the formula

$$\psi(\mathsf{i}) \wedge \neg\exists \mathsf{i}_1(\mathsf{i}_1 < \mathsf{i} \wedge \psi(\mathsf{i}_1)),$$

where ψ is a Tm-calculus formula. Clearly, $min_{\mathsf{i}}(\psi(\mathsf{i}))$ is true if and only if i is the minimum of those i' such that $\psi(\mathsf{i}')$ is true. Now the above query is expressed as follows:

$$\{p \mid \exists \mathsf{i} \exists p_1 (min_{\mathsf{i}}(\langle \mathtt{Plant} : p, \mathtt{Profit} : p_1\rangle \in \phi_{\mathtt{Plant\_Profit}}(\mathsf{i}) \wedge p_1 > 20) \wedge \forall \mathsf{i}_1 \forall p_2 (\mathsf{i}_1 > \mathsf{i} \wedge \langle \mathtt{Plant} : p, \mathtt{Profit} : p_2\rangle \in \phi_{\mathtt{Plant\_Profit}}(\mathsf{i}_1) \rightarrow p_2 > 15))\}.$$

In a similar vein, queries concerning *until* can be handled in our framework as well. For example, queries such as "What plant had been producing Rock Candy until the Los Angeles plant started making Paper Towels?" can also be easily expressed in Tm-Calculus.

The above examples and informal arguments form the basis for the proof of the following theorem.

Theorem 1 Tm-calculus is at least as expressive as queries formed by using formulas in linear-time propositional temporal logic (using modalities $\Box, \Diamond, \mathbf{P}, \mathbf{H}, \mathbf{S}$ as defined by Gabbay [Gab87]).

Thus, Tm-calculus is a powerful language and can express most of the natural temporal queries. Since the domain of time is infinite, however, some queries will have infinite answers. For example, consider the example of Figure 1. Suppose we ask: "In which years did the Baltimore plant *not* make Cellular Car Phones?" By its very nature, the set of time points is infinite, and the answer to this query is the infinite set $\{0, \ldots, 1990, 1992, 1993, \ldots\}$. However, for all practical purposes, most databases were created at some point in time `START` (we will always consider `START` to be 0). Furthermore, in most applications, there is little interest in what is going to be true 5,000 years later. In other words, it may often make sense to establish an *a priori* upper bound on time, i.e., to think of time as stretching from the start time 0 to some fixed time `END`. All temporal modalities discussed previously can be redefined easily to work within this fixed *finite* time-frame. We call temporal logic with a finite, fixed (`START`, `END`) time-frame *bounded temporal logic.*

Though bounded temporal logics cause TM-calculus queries to have finite answers in terms of time, they do not force finite answers in terms of other domain values. For example, the answer to the Tm-calculus query $\{x|\neg\exists \mathrm{i}(\langle A : x\rangle \in \phi(\mathrm{i}))\}$ may be infinite. We can define a notion of *safeness* similar to the one of the relational calculus [Ull88]. (Due to the page limit, we omit the definition of the safe Tm-calculus here.) We obtain the following computability result by using the concept of bounded temporal logic and the notion of safeness.

Theorem 2 All safe Tm-calculus queries under the semantics of bounded time have finite answers and are computable.

4 Time Units

In the previous two sections, we implicitly assumed that all temporal modules in a temporal database use the same time unit. For instance, all the temporal modules in our earlier examples used *year* as their implicit common time unit. However, as mentioned in the Introduction, the mismatch concerning time units used by individual constituents is one of the problems we will have when we combine different temporal databases into a federation. In this section, the concept of time units is formally defined and its properties are studied. Temporal databases will be extended in the next section to federated temporal databases which include temporal modules of varying time units.

There has been a great deal of discussion with respect as to the nature of time (e.g., [And82]). From the database application point of view, we adopt the notion that time is discrete.[5] That is, there is a smallest time unit manageable by all databases, e.g., $1/2^{16}$ of a second, and the time points in this smallest time unit are isomorphic to the non-negative integers, i.e., $\mathbf{N}$.

The temporal information stored in a database, however, is often not in terms of the smallest time unit. It is generally stored in terms of some larger time unit. For example, we store the profit in a *year*, the rainfall in a *season* and the salary of a *month*. Conceptually, each moment of time in these larger time units corresponds to an interval of the smallest time units. This leads to our definition of time units.

Definition 4 A mapping μ from $\mathbf{N}$ to $2^{\mathbf{N}}$ is called a *time unit* if

(1) 0 is in $\mu(0)$;

(2) for integers i and $i_1 < i_2 < i_3$, $\{i_1, i_3\} \subseteq \mu(i)$ implies i_2 is in $\mu(i)$;

(3) for integers $i \neq j$, $\mu(i) \cap \mu(j) = \emptyset$; and

(4) for each integer i, there exists j such that i is in $\mu(j)$.

Intuitively, condition (1) above says that each time unit starts from the beginning. Condition (2) ensures that for each i, $\mu(i)$ includes a "continuous" block, condition (3) is for the purpose that no two such blocks overlap, and condition (4) requires that each time unit covers the whole time line, i.e., $\mathbf{N}$.

Thus, a time point i in time unit μ corresponds to an interval of the smallest time units. By relating an arbitrary time unit to the smallest time unit, the conversion between information in different time units will be manageable. But first, let us discuss some properties of the set of all time units.

Clearly, there is a "finer-than" relation between time units. For example, *day* is finer than *month*, *month* is finer than *year*, and etc. Formally, we have:

Definition 5 Let μ_1 and μ_2 be two time units. Then μ_1 is said to be *finer* than μ_2 if for each i, there exists $j \leq i$ such that $\mu_1(i) \subseteq \mu_2(j)$. μ_1 is said to be *coarser* than μ_2 if μ_2 is finer than μ_1. μ_1 and μ_2 are said to be *incomparable* if neither μ_1 is finer than μ_2, nor μ_2 is finer than μ_1.

It is easily seen that *year* is coarser than *month*, and *week* and *month* are incomparable.

If μ_1 is finer than μ_2, then we will write $\mu_1 \preceq \mu_2$. Clearly, $\mu \preceq \mu$ for each time unit μ. Furthermore, it is obvious that $\mu_1 = \mu_2$ if $\mu_1 \preceq \mu_2$ and $\mu_2 \preceq \mu_1$. Thus, we have the following result:

Theorem 3 $\preceq$ is a partial order.

In fact, the set of time units forms a complete lattice with respect to the order imposed by $\preceq$. That is, we have

Theorem 4 The set of all time units is a complete lattice with respect to the finer-than relation[6].

The fact that the time units form a lattice enables us to *navigate* through the time units in a natural way. Intuitively, two incomparable time units are "linked" by their lub or glb. This observation will be used in Section 5 when temporal modules are extended to have windows of different time units.

The time units as defined above may sometimes be very hard to "compute" and counter-intuitive to "real-life" concepts of time units. Indeed, let $\mu(i) = \{i\text{-th prime}, \ldots, (i+1)\text{st prime} - 1\}$ (assuming 0-th prime is 0) for each $i \geq 0$. Clearly, μ is a time unit by definition. However, it is very difficult to imagine someone would actually use this time unit to measure anything. All time units in real life, such as *day*, *month* and *year*, seem to be "periodic" in nature. This leads to the following definition:

[5] Many of our results can be generalized to the case when time is assumed to be non-discrete (e.g., dense, continuous).

[6] The *top* element of this lattice is the time unit μ_T defined by $\mu_T(0) = \mathbf{N}$ and $\mu_T(i) = \emptyset$ for each $i > 0$, and the *bottom* element is the time unit μ_B defined by $\mu_B(i) = \{i\}$ for each $i \geq 0$.

Definition 6 For each integer $K > 0$ and subset M of $\mathbf{N}$, let $M_{-K} = \{i - K | i \in M\}$. A time unit μ is said to be *periodic* if there exist non-negative integers K_1, K_2 and n such that $\mu(i) = \mu(i + K_1)_{-K_2}$ for each $i > n$.

Thus, *day*, *week* and *month* are all periodic time units[7]. The time unit *year* is also periodic.

The concept of periodic time units given here is to provide the users with some flexibilities. The discussions in the following sections, i.e., Section 5 and 6, hold even for periodic time units.

It is easily seen that the time unit μ defined by $\mu(0) = \mathbf{N}$ and $\mu(i) = \emptyset$ for each $i > 0$ is a periodic time unit. Also, given a finite set of time units, the greatest lower bound and the least upper bound are both periodic. Hence, we have:

Theorem 5 The set of all periodic time units is a (non-complete) lattice with respect to the finer-than relation.

All periodic time units do not form a complete lattice because the greatest lower bound of an infinite set of periodic time units may not be periodic. Indeed, consider the set $\{\mu_i | i \geq 0\}$ of time units, where each μ_i is defined as follows:

$$\mu_i(0) = \{0, \ldots, 2^i\}, \text{ and } \mu_i(j) = \mathbf{N} - \mu_i(0) \text{ for each } j > 0.$$

It is easily seen that there is no greatest lower bound of the set (in the set of periodic time units).

We conclude this section by stating the following result:

Theorem 6 Given two time units (periodic time units, respectively) μ_1 and μ_2 such that (i) $\mu_1 \preceq \mu_2$ and (ii) $\mu_1(i) \neq \emptyset$ and $\mu_2(i) \neq \emptyset$ for each $i \geq 0$. Then there are infinite number of time units (periodic time units, respectively) μ such that $\mu_1 \preceq \mu \preceq \mu_2$.

For example, between *day* and *week*, we can have the following periodic time unit: for every i-th week, combine the last two days in the week into a period called weekend. Thus, in this time unit, every i weeks we will have a weekend. We now have infinite number of periodic time units (i.e., one for each i) which are between *day* and *week*.

5 Extended Temporal Modules and Federated Temporal Databases

In this section, we extend the temporal databases defined in Section 2 to federated temporal databases. In a federated temporal database, temporal modules on various different time units may co-exist.

In a temporal database, all temporal modules have a "predefined" time unit. For example, the temporal database example at the end of Section 2 consists of temporal modules in terms of the time unit 'year." This raises at least two problems if temporal databases on different time units are combined to form a federation.

Firstly, a query may be issued to a temporal module in terms of a different time unit than the one in terms of which the temporal module is defined.[8] For example, one may ask for the profit percentage of the Los Angeles plant in the first month of 1992, or the profit percentage of the Baltimore plant in the decade 1980–1989.

Secondly, several temporal databases may be needed jointly to answer a particular query and the temporal databases involved use different time units. For example, consider a temporal database for employee's monthly salaries and a temporal database about yearly profits of plants. The information about profits and about salaries are both needed to answer such queries as "who made more than 100K during the year when the profit was lower than 0.5%?".

In order to solve the above problems, the concept of temporal modules is extended. Basically, the extended temporal modules will be able to convert information to different time units. More specifically, we have:

Definition 7 An *extended temporal module* is a quadruple $(R, \upsilon, \phi, \tau)$, where R is a relation scheme, υ a set of time units, ϕ a mapping from $\upsilon \times \mathbf{N}$ to $2^{\mathbf{Tup}(R)}$ and τ a mapping from $\upsilon \times \mathbf{Tup}(R)$ to $2^{\mathbf{N}}$. Each time unit in υ is called an *available time unit* of the extended temporal module.

Thus, an extended temporal module is a temporal module which can deal with (possibly) many time units. For example, a temporal module which keeps a person's salaries of particular months can easily be extended to give daily and hourly salaries of these months. Furthermore, this temporal module may also give information about a year's salary of the person. (Notice that the daily and hourly salaries may only involve a simple division. It may be more than just a multiplication, however, to obtain a year's salary of a particular person in a particular year. Indeed, the person might get a raise or switch the job.)

An extended temporal module gives the outside world multiple views of presumably the same information. These different views are derived from the same source. For example, suppose the information source is about monthly salaries. Then daily salary can be calculated from the monthly salaries, and yearly salary can be obtained by accumulating every month's salary of the whole year.

Consider such information conversion within an extended temporal module. Suppose μ_1 and μ_2 are available time units and information in terms of μ_1 is transformed into that in terms of μ_2. Three cases arise:

1. $\mu_1 \preceq \mu_2$. The conversion is usually an "aggregation" procedure. The same procedure should give us the

[7] Notice that a time unit μ is periodic does not mean that the size of $\mu(i)$ is fixed for each i.

[8] This could also happen in a non-federated temporal database. However, this problem would appear more frequently in a federated temporal database system, since a user may not be aware of the time units used in various constituents.

information in terms any time unit between μ_1 and μ_2. For example, if several month's rainfall amounts are collect to give a year's rainfall amount, then the same procedure can be used to give the rainfall amount for half years.

2. $\mu_2 \preceq \mu_1$. The conversion is usually an "interpolation" procedure [Cli82, SS87, WJL91, WJL93]. A similar interpolation procedure should give us the information in terms of any time unit between μ_1 and μ_2. For example, if a yearly salary is averaged to give hourly salary, a similar procedure can be used to give daily and monthly salaries.

3. μ_1 and μ_2 are incomparable. To perform such a translation, there should be a procedure to convert from the information in terms of μ_1 to their common greatest lower bound (or least upper bound) μ_3, and then translate the information in terms of μ_3 to μ_2. Hence, the information should also be available in terms of μ_3, i.e., a lower bound (or the upper bound) of μ_1 and μ_2. As an example, consider the procedure in converting information about monthly salaries to the salary of a particular week. Clearly, we may have to convert monthly salaries into daily salaries and then from these day's salaries to the salary of the week.

The above considerations lead us to the following definition:

Definition 8 Let $ETM = (R, \upsilon, \phi, \tau)$ be an extended temporal module, $\mu_l = glb(\upsilon)$ and $\mu_u = lub(\upsilon)$. Then ETM is said to be *downward closed* (*upward closed*, respectively) if υ satisfies the following condition:

For each time units μ', if $\mu_l \preceq \mu' \preceq \mu''$ ($\mu'' \preceq \mu' \preceq \mu_u$, respectively) for some μ'' in υ, then μ' is in υ.

Furthermore, ETM is said to be *closed* if it is either downward closed or upward closed.

Clearly, each extended temporal module $(R, \upsilon, \phi, \tau)$, where $|\upsilon| = 1$, is both upward closed and downward closed.

For each closed extended temporal module

$$(R, \upsilon, \phi, \tau),$$

if υ contains at least two time units μ_1 and μ_2 such that (i) $\mu_1 \preceq \mu_2$ and (ii) $\mu_1(i) \neq \emptyset$ and $\mu_2(i) \neq \emptyset$ for each $i \geq 0$, then υ contains infinite number of time units by Theorem 6. In practice, υ in most closed extended temporal modules will be an infinite set. This is unreasonable for a database to store all those time units. This leads us to the definition of "time units generator." Intuitively, a time units generator is a finite set of time units plus the direction ("up" or "down") the new time units will be generated. Formally:

Definition 9 A *time unit generator* is a pair (υ, d), where υ is a finite set of time units and d is in $\{\mathbf{down}, \mathbf{up}\}$. For each time unit generator (υ, d), let $G(\upsilon, d)$ be the set of time units as follows:
if $d = \mathbf{down}$, then

$$G(\upsilon, d) = \{\mu | glb(\upsilon) \preceq \mu \preceq \mu_1 \text{ for some } \mu_1 \text{ in } \upsilon\}$$

and if $d = \mathbf{up}$. then

$$G(\upsilon, d) = \{\mu | \mu_1 \preceq \mu \preceq lub(\upsilon) \text{ for some } \mu_1 \text{ in } \upsilon\}.$$

Informally, time unit generators capture the idea that if information is transformed between two units μ_1 and μ_2, then (depending on whether going up or down) all the time units available to the outside world should be $G(\{\mu_1, \mu_2\}, d)$.

In an extended temporal module, we shall use a time unit generator instead of a (usually infinite) set of available time units. By abuse of language, we shall also call $(R, (\upsilon, d), \phi, \tau)$ a closed extended temporal module if $(R, \upsilon, \phi, \tau)$ is an extended temporal module, υ a finite set of time units and d is either 0 or 1,

We are now ready to define federated temporal databases. Specifically, a *federated temporal database* is a set D of elements, called *extended temporal module names*, and each extended temporal module name is associated with a closed extended temporal modules. For example, D is a federated temporal database if

$$D = \{\ \texttt{Product\_Profit},\ \texttt{Plant\_Profit}\ \},$$

and `Product_Profit` and `Plant_Profit` are associated to two extended temporal modules, respectively, that are closed.

Let D be a federated temporal database. For each M in D, we will in the remainder of this paper use ϕ_M and τ_M to denote the third and the fourth components (i.e., the ϕ and τ functions), respectively, of the closed extended temporal module associated with M,

6 Extended Tm-Calculus

In this section, the Tm-calculus is extended to form queries on federated temporal databases. The extended Tm-calculus allows its users to specify the time units for each time variable and constants used in queries.

An *extended Tm-calculus formula* is a Tm-calculus formula with the following changes:

(1) Each integer constant i appearing in an atom is changed to $i : \mu$, where μ is a time unit. For example, $1992 : \mathit{year}$.

(2) Each application of quantification of the form $\exists \mathsf{i}(\psi(\mathsf{i}))$ ($\forall \mathsf{i}(\psi(\mathsf{i}))$, respectively) is changed to $\exists \mathsf{i} : \mu(\psi(\mathsf{i}))$ ($\forall \mathsf{i} : \mu(\psi(\mathsf{i}))$, respectively). For example, $\exists \mathsf{i} : \mathit{month}(\mathsf{t} \in \phi_{\texttt{Plant\_Profit}}(\mathsf{i}))$.

(3) The comparison operations are changed to the following set

$$\{.x < y., .x > y., .x \leq y., .x \geq y., .x = y., .x \neq y. \\ |x, y \text{ in } \{\mathbf{a}, \mathbf{s}\}\}.$$

The character $\mathbf{a}$ is read as "all" and $\mathbf{s}$ as "some." The semantics of $\mathsf{i}_1.\mathbf{a}\theta\mathbf{s}.\mathsf{i}_2$ is interpreted as follows: Suppose i_1 is of time unit μ_1 and i_2 time unit μ_2. Then $\mathsf{i}_1.\mathbf{a}\theta\mathbf{s}.\mathsf{i}_2$ is true iff for *all* i in $\mu_1(\mathsf{i}_1)$ there exists *some* j in $\mu_2(\mathsf{i}_2)$ such that $i\theta j$ is true. The other combinations of $\mathbf{a}$ and $\mathbf{s}$ are interpreted in similar ways.

(4) Each time variable $\mathbf{z}_i = \mathsf{i}$ in the query

$$\{\langle \mathbf{z}_1, \ldots, \mathbf{z}_k\rangle | F\}$$

is changed to $\mathsf{i} : \mu$, where μ is a time unit. For example,

$$\{\mathsf{i} : \mathit{year} | \langle \texttt{Plant} : \texttt{LosAngeles}, \texttt{Profit} : 10\rangle \in \phi_{\texttt{Plant\_Profit}}(\mathsf{i})\}.$$

Notice that the changes in (1), (2) and (4) above are for the purpose of pointing out the time unit of each time variable and constant. Since each time point in a time unit corresponds to an interval in the "smallest" time unit and two time variables (or a time variable and a time constant) may be in different time units. Thus, we need to change the comparison operators. The change in (3) above is for this purpose.

We note here that all 13 interval comparisons in [All83] can be expressed by extended Tm-calculus formulas. For example, $\mathsf{i}_1 \mathit{during} \mathsf{i}_2$ can be expressed as follows[9]:

$$(\mathsf{i}_1.\mathbf{a} \leq \mathbf{s}.\mathsf{i}_2) \wedge (\mathsf{i}_1.\mathbf{a} \geq \mathbf{s}.\mathsf{i}_2).$$

As another example, $\mathsf{i}_1 \mathit{starts} \mathsf{i}_2$ can be expressed by $(\mathsf{i}_1.\mathbf{a} \leq \mathbf{s}.\mathsf{i}_2) \wedge (\mathsf{i}_1.\mathbf{a} \geq \mathbf{s}.\mathsf{i}_2) \wedge \neg(\mathsf{i}_2.\mathbf{s} < \mathbf{a}.\mathsf{i}_1)$.

We now give two examples to exhibit the extended Tm-calculus. The extended temporal modules we use are four temporal modules, namely `Product_Profit`, `Plant_Profit`, `Salary` and `Manager`, extended to the following set of available time units: $G(\upsilon, \mathbf{down})$, where

$$\upsilon = \{\mathit{year}, \mathit{month}, \mathit{week}\}.$$

(Thus, these extended temporal modules are closed.)

Example 7 "During which month in 1992 the profit percentage of Los Angeles plant exceeded 19%?" This is expressed in the extended Tm-calculus as follows:

$$\{\mathsf{i} : \mathit{month} | \mathsf{i}\, \mathit{during}\, 1992 : \mathit{year} \wedge \exists p (p > 19 \wedge \langle \texttt{Plant} : \texttt{Los Angeles}, \texttt{Profit} : p\rangle \in \phi_{\texttt{Plant\_Profit}}(\mathsf{i})\},$$

where *during* comparison is defined earlier.

[9] For those not familiar with this *during* comparison and the next one about *starts*, $\mathsf{i}_1 \mathit{during} \mathsf{i}_2$ means, roughly, that the interval (represented by) i_1 is covered by the interval i_2, and $\mathsf{i}_1 \mathit{starts} \mathsf{i}_2$ means that the interval i_1 is covered by i_2 and there is no time point in i_2 which is before all time points in i_1.

Notice that $\phi_{\texttt{Plant\_Profit}}$ is invoked as

$$\phi_{\texttt{Plant\_Profit}}(\mathsf{i})$$

instead of $\phi_{\texttt{Plant\_Profit}}(\mathit{month}, \mathsf{i})$. This is because the time unit that i uses is clear from the context. Indeed, each time variable i in an extended Tm-calculus query $\{\langle \mathbf{z}_1, \ldots, \mathbf{z}_k\rangle | F\}$ is either bounded or $\mathbf{z}_j$ is of the form $\mathsf{i} : \mu$ for some j, and thus has a "designated" time unit.

Example 8 "What's the yearly profit percentage during the first whole calendar year in which John served as a manager, assuming John could take office any month in a year?" (Notice that if John took office in the middle of a calendar year, then the answer should be the profit of the next calendar year.) First, suppose that i_m is a time variable on time unit *month* and i_y a time variable on *year*. Then the formula below is true iff either (i) i_m is the first month of i_y or (ii) i_y is the first year after i_m:

$$FY(\mathsf{i}_m, \mathsf{i}_y) = (\mathsf{i}_m \mathit{starts} \mathsf{i}_y) \vee (\mathsf{i}_m.\mathbf{a} \leq \mathbf{a}.\mathsf{i}_y \wedge \neg\exists \mathsf{i} : \mathit{year}(\mathsf{i}_m.\mathbf{a} \leq \mathbf{a}.\mathsf{i} \wedge \mathsf{i}.\mathbf{a} \leq \mathbf{a}.\mathsf{i}_y)).$$

Now the query can be expressed as follows:

$$\{p | \exists \mathsf{i}_m : \mathit{month} \exists \mathsf{i}_y : \mathit{year} (FY(\mathsf{i}_m, \mathsf{i}_y) \wedge \mathit{min}_{\mathsf{i}_m}(\exists p_1 \langle \texttt{Plant} : p_1, \texttt{Name} : \texttt{John}\rangle \in \phi_{\texttt{Manager}}(\mathsf{i}_m)) \wedge \langle \texttt{Plant} : p_1, \texttt{Profit} : p\rangle \in \phi_{\texttt{Plant\_Profit}}(\mathsf{i}_y))\},$$

where $\mathit{min}_{\mathsf{i}_m}$ is the formula used in Example 6.

We conclude this section by a brief discussion of how to process extended Tm-calculus queries. As seen from the above, in Tm-calculus queries, the time units are explicitly stated for each application of the windowing functions. However, each windowing function of an extended temporal module has a fixed set of associated time units. Thus, the system has to determine if the time unit required by the query is available. This step may involve the use of time unit generators and/or information conversions, both of which are discussed in the previous section.

7 Conclusions

In this paper, we have introduced the concept of a temporal module for resolving certain mismatches among different temporal databases within a federated database system. Temporal modules provide us with a high level of abstraction through two windowing functions ϕ and τ. We now can query the federated temporal database, each probably based on a different data model, in the federation through a unifying framework.

Another contribution of the paper is the formal treatment of time units. Here a time point in a "higher" time unit is seen as a (not necessary fixed-length) interval of the "lowest" time unit. (The lowest time unit in practice is usually much finer than the ones actually used by applications.) This enables us to seek relationships among time points in different time units in a natural way. Indeed, the information conversion over different time units can now be seen as through the lub or the glb of the involved time units.

The basic ideas behind temporal modules and extended ones are similar to those behind objects in object-oriented systems. Indeed, a temporal module could be viewed as a temporal object which gives temporal information. The interesting point here is that the two windowing functions, i.e., functions ϕ and τ, in a temporal module are all we need to access the temporal information inside the module. The choice of a relation scheme as another component of a temporal module is only for convenience. We speculate that the temporal module concept can be easily extended to other data models, such as object-oriented temporal database models, e.g., [CC88, KRS90].

Several research issues are raised by this study. The first one is whether there is algebraic query languages which are equivalent to the Tm-calculus and the extended Tm-calculus, respectively. Another one is about the time and space complexities of the two calculi. There are also research problems concerning time units. For example, how do we store time units? What do we want to reason about the time units? We may also want to study some "standard" information conversions (see [WJL93]) in extended temporal modules, e.g., average function, stable function, as well as other various aggregation functions.

References

[And82] T.L. Anderson. Modeling time at the conceptual level. In *Proceedings of the International Conference on Databases: Improving Usability and Responsiveness*, pages 273–297, Jerusalem, Israel, June 1982. Academic Press.

[All83] J.F. Allen. Maintaining knowledge about temporal intervals. *Communications of the Association of Computing Machinery*, 26(11):832–843, November 1983.

[CC88] J. Clifford and A. Croker. Objects in time. *IEEE Data Engineering*, 7(4):189–196, December 1988.

[Cli82] J. Clifford. A model for historical databases. In *Proceedings of Workshop on Logical Bases for Data Bases*, Toulouse, France, December 1982.

[CT85] J. Clifford and A.U. Tansel. On an algebra for historical relational databases: Two views. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 247–265, Austin, TX, May 1985.

[CR87] J. Clifford and A. Rao. A simple, general structure for temporal domains. In *Proceedings of the Conference on Temporal Aspects in Information Systems*, pages 23–30, France, May 1987. AFCET.

[Gab87] D. Gabbay. Modal and temporal logic programming. In A. Galton, editor, *Temporal Logics and their Applications*, pages 197–237. Academic Press, 1987.

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

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

[KRS90] W. Kaefer, N. Ritter, and H. Schoening. Support for temporal data by complex objects. In *16th International Conference on VLDB*, Brisbane, Australia, August 1990.

[MS87] E. McKenzie and R. Snodgrass. Supporting valid time: An historical algebra. Technical Report TR87-008, Computer Science Department, University of North Carolina at Chapel Hill, August 1987.

[NA89] S. B. Navathe and R. Ahmed. A temporal relational model and a query language. *Information Sciences*, 49(2):147–175, 1989.

[SA85] R. Snodgrass and I. Ahn. A taxonomy of time in databases. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 236–246, Austin, TX, May 1985.

[Sno84] R. Snodgrass. The temporal query language TQuel. In *Proceedings of the ACM Symposium on Principles of Database Systems*, pages 204–212, Waterloo, Ontario, Canada, April 1984.

[SS87] A. Segev and A. Shoshani. Logical modeling of temporal data. In *Proceedings of the ACM SIGMOD Annual Conference on Managemen t of Data*, pages 454–466, San Francisco, CA, May 1987.

[Tan93] A. Tansel and et al., editors. *Temporal Databases*. Benjamin/Cummings, Jan 1993.

[Ull88] J. D. Ullman. *Priciples of Database and Knowledge-base systems*. Computer Science Press, 1988.

[WJL91] G. Wiederhold, S. Jajodia, and W. Litwin. Dealing with granularity of time in temporal databases. In *Proc. 3rd Nordic Conf. on Advanced Information Systems Engineering*, Lecture Notes in Computer Science, Vol. 498, (R. Anderson et al. eds.), Springer-Verlag, 1991, pages 124–140.

[WJL93] G Wiederhold, S. Jajodia, and W. Litwin. Integrating temporal data in a heterogeneous environment. In *Temporal Databases*. Benjamin/Cummings, Jan 1993.