Handling temporal grouping and pattern-matching queries in a temporal object model*

Marlon Dumas, Marie-Christine Fauvet and Pierre-Claude Scholl
Laboratoire Logiciels, Systèmes et Réseaux, IMAG
Université de Grenoble, BP 72, 38402 St Martin d'Hères (France)
Tel. +33 4 76 82 72 83, Fax. +33 4 76 82 72 87
Marlon.Dumas@imag.fr

Abstract

This paper presents a language for expressing temporal pattern-matching queries, and a set of temporal grouping operators for structuring histories following calendar-based criteria. Pattern-matching queries are shown to be useful for reasoning about successive events in time while temporal grouping may be either used to aggregate data along the time dimension or to display histories. The combination of these capabilities allows to express complex queries involving succession in time and calendar-based conditions simultaneously. These operators are embedded into the TEMPOS temporal data model and their use is illustrated through examples taken from a geographical application. The proposal has been validated by a prototype on top of the O_2 DBMS.

Keywords: temporal databases, temporal object model, temporal query language, pattern-matching queries, temporal grouping, calendar, granularity, O_2.

1 Introduction

In most modern DBMS, time is one of the primitive datatypes provided for describing entities and associations. Depending on the systems, the structuration of temporal data and the facilities offered to manage them are more or less complete. However, the hierarchy of temporal types proposed is generally poor and the notion of granularity seldomly taken into account. Furthermore, in these systems, a database state represents a snapshot of the modeled world as defined by the last update. When the need for maintaining data evolution arises, their history has to be managed at the application level, which increases data and programs complexity. Temporal data models and DBMS aim at overcoming this lack of capabilities [20, 12, 18, 19].

TEMPOS[1] (Temporal Extension Model for Persistent Object Servers) [9, 3, 17, 6] is an extensible temporal data model integrating the basic concepts and facilities required to manage the data historical dimensions on top of a DBMS. The main goal of the work around TEMPOS is to provide a testbed for defining temporal operators and studying their suitability in regard to concrete applications' requirements. In this paper, we present temporal grouping and pattern-matching operators, we show how they may be embedded into the TEMPOS data model, and we illustrate their use through examples taken from a GIS application.

Temporal grouping allows to structure a history following temporal criteria. As such, it can be seen as a temporal extension of SQL's *group by*. We focus here on grouping using calendar-based criteria. This operation is useful in particular for displaying histories or to perform aggregations.

Pattern-matching is a well-known process in many areas of computer science. The idea behind this concept is to compare some piece of data to a standard one of some kind. In the relational framework, some extensions to query languages have been defined to support complex pattern-matching queries [11]. In temporal databases, the need for pattern-matching capabilities arises when reasoning about successive changes of an object or property on time [4]. Indeed, our experience on applications with temporal aspects (time series, GIS, video databases) have shown a great need for expressing complex patterns on data history.

The rest of the paper is organized as follows. In section 2 we review the basic concepts of the TEMPOS model and illustrate them through queries taken from a GIS application. In section 3, the temporal grouping and the pattern-

*This work is partially supported by the french GDR Cassini's PSIG

[1] TEMPOS is developed as part of the STORM project [1] which aims at extending object DBMS to deal with structural, temporal and active features of multimedia data.

matching operators are defined and integrated into the TEMPOS model. Section 4 discusses related works. Finally in section 5 we sketch some open issues that we plan to study in the future.

2 An overview of TEMPOS

TEMPOS is composed of a time model, which defines a hierarchy of simple and complex datatypes for modeling temporal values, and a historical model, which allows to update and query timestamped object properties.

2.1 Model of time

TEMPOS is based upon a discrete, linear and left-bounded time model in which the time line is structured in a *multi-granular* way by means of *time units* [21]. A time unit is a partition of the time line into a set of disjoint contiguous time intervals: each interval is seen as an atomic *granule* and every point of the time line is approximated by the granule which contains it.

If a mapping can be established between each granule of a time unit u1 and a set of consecutive granules of another unit u2, then u2 is said to be *finer than* u1 (u2 $\prec$ u1), or symmetrically, u1 is said to be *coarser than u2*. For instance, the time unit *month* is finer than the unit *year* but coarser than the unit *day*. The $\prec$ relation is used to manage conversions between temporal values at different granularities.

Based on this temporal structure, the TEMPOS model defines a wide variety of temporal datatypes (instant, span, interval, set of instants, calendar) and a set of operations over them which are independent of the units being involved. For a more detailed description of these types and operations the reader may refer to [3, 17].

2.2 Historical attributes

Each attribute has an evolution status which characterizes its behaviour in time and the way it is observed. An attribute is *constant* if it must not change in time, else it may be *historical* if its successive values are meaningful and thus recorded, or else *fleeting* if only its most recent value is meaningful.

The *history* of a historical attribute includes whatever has to be kept from its evolution. It is a set of snapshots taken at the instants considered to be relevant by the application. A *snapshot* is defined here as a temporal relation between a state and the instant at which this state is observed. In order to distinguish these two components of a snapshot, we respectively term them the *structural* and *temporal* values of the snapshot.

The *temporal dimension(s)* of a historical attribute gives the meaning of its snapshots. The most noteworthy distinction is between *valid* and *transaction time* [12]: in the former case, temporal values reflect property changes in the modeled world, whereas in the latter, they reflect database changes.

The *time unit* of a historical attribute defines the granularity at which its snapshots are observed. The *temporal domain* of a historical attribute is the domain of the associated history seen as a temporal function. A temporal domain may evolve in time through specific updating operations.

A history is built from a succession of updates which provide the data reflecting the evolution of the corresponding attribute or some corrections on already recorded snapshots. Obviously, the recorded snapshots do not necessarily concern all instants of the temporal domain, their timestamps are chosen by the user according to a periodicity specific to the nature of the attribute. The structural values of the attribute at those instants not described by an update are defined according to several interpretations:

– *Discrete*: non inputted values are always considered as absent (e.g. the history of the daily amount of books sold in a bookstore).

– *Regular*: the temporal domain is dynamically defined by the update operations (e.g. the history of deposits on a bank account).

– *Stepwise*: structural values are stable between two updates (e.g. the history of an employee's salary).

– *Interpolated*: non inputted values are defined by a temporal interpolation function (e.g. the history of a physical continuous process).

The model allows various representations of histories by grouping instants together in several ways. For instance, a history can be represented as an *I_Chronicle* (ordered sequence of instant-timestamped values), an *X_Chronicle* (coalesced sequence of interval-timestamped values) or a *D_Chronicle*, (set of distinct values timestamped by non-overlapping and non-contiguous sets of intervals). Operations on chronicles allow to switch from one representation to another.

2.3 A modeling example

We illustrate our approach on examples taken from a GIS application. Its goal is to perform a spatio-temporal analysis of people's behaviour in a mountain resort. The modeled data comes from an investigation on the use of the space and the facilities of the site, in which both tourists and inhabitants were asked to detail their daily activities according to a predefined nomenclature.

As a working example, we describe here the definition of the class Person: a person is identified by a number, at any time he/she performs a unique activity (skiing, personal caring, shopping, ...), accompanied by other persons (family, friends, sport guide, ...), in a given place characterized by its functionality (street, information desk, bakery, skiing site, ...). Attribute number is constant. Attributes activity, accompaniment and place are stepwise historical attributes defined at the granularity of the minute. The temporal domain of these attributes is defined statically as [12/1/97..2/28/98], which is the interval during which the survey took place. ThePersons is the persistent root corresponding to the class Person. Figure 1 describes the class Person using ODMG's Object Definition Language notations. In this description, historical attributes are underlined and their temporal characteristics are given in comments.

```
Class Person (extent ThePersons) {
  attribute number: integer,
  attribute activity: string,        // stepwise, granularity minute
  attribute accompaniment : set<string>,
                                     // stepwise, granularity minute
  attribute place: string            // stepwise, granularity minute
}
```

Figure 1: class Person

2.4 Operations on histories

The model includes a set of high level primitives on histories: they allow the user to reason on histories independently of their internal or external representations. In other words, all operations are defined over an abstract view of histories, in which a history is seen as a temporal function represented by a set of <instant, value> pairs.

These operations are mostly taken either from temporal extensions of relational algebra [20] (temporal projection, selection, product and intersection) or from iterators on sequences which are very popular in functional and logic languages (e.g. the map operation which pointwisely applies a function to each element of a sequence). They have been adapted to the TEMPOS historical model and validated, through their suitability on typical examples [9, 17] and their implementation in the prototype [6].

We illustrate some of these operators through queries taken from the application presented in subsection 2.3. For each operator, we give a functional specification followed by some queries in the OQL language extended by the operators of the TEMPOS data model.

The following notations are used: T1 $\longrightarrow$ T2 denotes the type of all functions with domain T1 and codomain T2. {T} is the type of sets of T and <T1, T2, ..., Tn> the type of tuples whose i[th] component is of type Ti ($1 \leq i \leq n$). T1(T2) denotes an instantiation of the parametrized type T1 with type T2: in particular, History(T) denotes the type of histories with structural values of type T. If h denotes a history then TDomain(h), denotes its temporal domain while unit(h) denotes its temporal unit (granularity). This latter notation is also used with other temporal values such as instants and sets of instants (e.g. if i is an instants then unit(i) denotes its granularity). In order to describe function values, we use the lambda notation, λx1, x2, .. $\bullet$ E where the symbol λ introduces the parameter names x1, x2, .. and the symbol $\bullet$ introduces the expression E defining the function. Finally, <v1, v2, ..., vn> denotes a tuple value whose i[th] component is vi ($1 \leq i \leq n$).

2.4.1 Restriction on temporal values

The restriction operation applied to a history h over a set of instants S (h Γ_{in} S), yields the history build from h's snapshots whose timestamp belong to S:

$\_\ \Gamma_{in}\ \_$: History(T), {Instant} $\longrightarrow$ History(T)
h Γ_{in} S = $\{<I,v> \mid <I,v> \in h \wedge I \in S$
{ *precondition: unit(h) = unit(S)* }

R.1 : ***temporal restriction*** For each person, retrieve her/his activity each day at 1pm.

```
select person: p,
       activ: p.activity Γin
       [@"12/1/97 1pm"..@"2/28/98 1pm" |♯"24 hours"]
from ThePersons as p
```
{ *@ introduces an instant litteral and ♯ a duration litteral. The expression [i1..i2 | d] denotes a sequence of periodic instants starting at i1, finishing at i2 and with periodicity d.* }

2.4.2 Restriction on structural values and product

Another type of restriction deals with the structural values of snapshots instead of their temporal values. Given a history h and a predicate P on the structural values of h, h Γ_{if} P yields the history made of h's snapshots whose structural value satisfies P.

$\_\ \Gamma_{if}\ \_$: History(T), (T $\longrightarrow$ boolean) $\longrightarrow$ History(T)
h Γ_{if} P = $\{<I, v> \mid <I, v> \in h \wedge P(v)\}$

The temporal product allows to establish a correspondence between two histories. Since our model supports temporal heterogeneity (i.e. two histories may have different temporal domains) we distinguish the *inner* temporal product from the *outer* one depending on whether the resulting history's temporal domain is respectively the intersection or the union of the temporal domains of the arguments. We present here the inner temporal product.

The inner temporal product of two histories (h1 $*_\cap$h2) is the history whose snapshots are obtained from those snapshots in h1 and h2 which have the same temporal value.

$\_ *_\cap \_$: History(T1), History(T2) $\longrightarrow$ History (<T1,T2>)
h1 $*_\cap$ h2 = $\{<I, <v1, v2>> \mid <I, v1> \in h1$
$\wedge <I, v2> \in h2\}$
{ *precondition: unit(h1) = unit(h2)* }

R.2 : ***temporal product*** For each person, retrieve the history of his/her activities and accompaniments when he/she was not skiing.

select person: p,
WhatWhom: (p.activity Γ_{if} λa $\bullet$ a $\neq$ “skiing”)
$*_\cap$
p.accompaniment
from ThePersons as p

2.5 Prototype development

The TEMPOS model has been implemented on top of the OO DBMS O_2 [6]. This implementation consists of a package of classes corresponding to the types of the time and historical models presented above, and a preprocessor for an OQL extension, TempOQL, in which temporal and historical types are primitive in the same way as integer, real, etc. In particular, TempOQL overrides OQL arithmetic operators to deal with temporal values.

Several applications have been modeled and implemented using TEMPOS. Among them, we can mention an application on economics involving time series and two GIS applications including the one that we describe in this paper.

3 Extending TEMPOS

3.1 Temporal grouping

Temporal grouping allows to structure a history into a history of histories according to some temporal criteria. This operation is useful in particular to perform aggregations on histories and as such, it may be seen as a temporal extension of SQL's *group by*. To illustrate this operation, we present here a specific temporal grouping operator based upon the notion of time unit and discuss how it can be extended to calendars.

The temporal grouping operator h ! u2, h being at granularity u1 (u1 $\prec$ u2), makes a partition on h according to unit u2. The result is a history (at granularity u2) of histories (at granularity u1) : each snapshot is of the form $<$i,hi$>$ where i is an instant at the granularity u2 and hi is a history at the granularity u1 corresponding to the temporal restriction of h to the interval expand(i, u1)[2]. Formally :

_ ! _ : History(T), Unit $\longrightarrow$ History(History(T))
h ! u = { $<$I, vh$>$ | $\exists$ I' $\in$ TDomain(h) approx(I', u) = I
$\wedge$ h Γ_{in} expand(I, unit(h)) = vh }
{ *precondition: unit(h) $\prec$ u* }

For instance, let h be the history of a person's activities observed at the granularity of the minute, h ! day returns a history at the granularity of the day, whose snapshots are themselves histories, each of them representing the history of the person's activities during a given day as shown in figure 2.

Figure 2 suggests that this operator may be used for displaying histories. For example, the query (h ! hour) ! day applied to the history h described above, will structure it like a diary: for each day, one will find the history of the activities of the person grouped by hours.

h	h ! day
<12/30 at 6:20 am, breakfast>	<12/30, [<6h20am : breakfast>,
<12/30 at 6:40 am, driving>	<6:40 am, driving>,
<12/30 at 7:00 am, skiing>	<7:00 am, skiing>,
⋮	⋮
<12/30 at 9:00 pm, dinner>	<9:00 pm, dinner>,
<12/30 at 10:00pm, TV>	<10:00pm, TV>,
<12/30 at 10:40pm, sleep>	<10:40pm, sleep>]>
<12/31 at 7:20 am, breakfast >	<12/31, [<7:20am : breakfast>,
<12/31 at 7:40 am, TV>	<7:40 am, TV>,
<12/31 at 9:00 am, cooking>	<9:00 am, cooking>,
⋮	⋮
<12/31 at 9:00 pm, dinner>	<9:00 pm, dinner>,
<12/31 at 10:00pm, party>	<10:00 pm, party>]>
⋮	⋮

Figure 2: Temporal grouping

On the basis of this temporal grouping operator, it is possible to define aggregation operators. We may define for instance an operator Aggregate(h,u,f) which first structures history h using the operator ! with parameter u, and then pointwisely applies an aggregate operation f to each of the sub-histories of the resulting history.

Aggregate: History(T), Unit, (History(T) $\longrightarrow$ T')
$\longrightarrow$ History(T')
Aggregate(h, u, f) = map(h ! u, $\lambda$$<i,hi>$ $\bullet$ $<$i,f(hi)$>$)
{ *precondition: unit(h) $\prec$ u* }

R.3 : ***Temporal grouping.*** For each person, how long did this person skied each day?
select person : p,
skied : Aggregate(p.activity Γ_{if} λa $\bullet$ a = “ski”,
Day,
Duration)
from ThePersons as p
{ *Duration is a function which calculates the duration of the temporal domain of a history.* }

The Aggregate operation can be easily composed to express more complex queries. Suppose for instance that h is the history of the daily sales of a product. The query:

Aggregate(Aggregate(h, Month, Sum), Year, Max)

calculates for each year when the product was on sale, its maximum monthly sale.

[2] If i is an instant with unit u2 and u1 is a unit such that u1 $\prec$ u2, then expand(i, u1) is an interval at unit u1 denoting the conversion of instant i to unit u1. Conversely, if i is an instant at unit u1 and u2 $\prec$ u1, then approx(i, u2) is an instant with unit u2 corresponding to i's conversion to this unit. For instance, expand(@“january 97”, Day) = [@“1/1/97”..@“31/1/97”] and approx(@“15/1/97”, Month) = @“january 97”

Another grouping operation of particular interest may be obtained by allowing the grouping criteria to be determined by a *calendar* instead of a time unit. A calendar is a structured collection of instants [13] such that each level of structuration corresponds to a time unit, e.g. the collection of all holidays between 1985 and 1998 structured on a per-year basis, or the collection of all mondays and tuesdays between 1985 and 1998 structured by weeks and years. The temporal grouping operator applied on a calendar will then perform two transformations on a history: restrict its temporal domain to the calendar's one, and recursively structure it into a history of histories to fit into the calendar's structure. Again, such an operator could be used either for displaying purposes or to perform aggregations.

[13, 14] present some operations for building up calendars. They have been generalized and integrated into the TEMPOS time model in [3].

3.2 Pattern-matching queries

Pattern matching queries involve retrieval of data by comparison to a standard of some kind [11]. To some extent, all query languages, and temporal query languages in particular, allow to express queries based on simple patterns: *retrieve all persons who have done shopping and skiing*. The goal here is to express more complex patterns such as *retrieve all persons who have done shopping, then skiing during at least 3 hours and then, in a 1 hour delay, have done shopping again*. When applied to histories, such patterns provide a powerful tool for reasoning about succession in time, which is a fundamental issue in temporal databases.

The pattern definition language that we propose integrates classical logics with timed regular expressions [2]. The former are used to build up atomic formulae and to combine them into complex boolean conditions for expressing properties about the structural value of a history at a given instant, while the latter are used to reason about succession in time. In addition, the language comprises an operator for assigning values to variables. This operator is used to memorize previous values of an entity.

The syntax of the atomic formulae of the language is defined as in multi-sorted first-order logics, over an alphabet made up of constants, functions (which model object properties), variables and predefined relations ($<$, $=$, etc.). Other well-formed formulae (wff) are constructed from atomic formulae by using classical propositional operators (and, or and not), an assignment operator [x $\leftarrow$ t]f where x is variable, t is a term and f a formula, and the following temporal operators:

- Sequencing: if f1 and f2 are wff then f1;f2 is a wff.
- Repetition: if f is a wff, θ a comparison operator ($<$, $=$, etc), n an integer term and d is duration term, then $(f)^n$, $(f)^+$ and $(f)^{\theta d}$ are wff too. The first two of this operators are counterparts of the corresponding operators on regular expressions. The third one adds a duration constraint on the patterns.

Let's consider a set of histories of persons' activities. The sequencing operator allows to retrieve which persons performed a given activity immediately after another one. For instance the expression :

activity="skiing" ; activity="shopping"

is used to retrieve persons who skied just after shopping. The power operators allow to reason on repetition of a pattern:

activity = "shopping";
(activity="skiing" or activity = "skating")$^+$;
activity = "shopping"

is used to retrieve persons who went shopping, then skied or skated, and subsequently went shopping again. Finally

(activity="skiing")$^{<\sharp\text{"2 hours"}}$

retrieves people who skied during less than 2 hours.

The semantics of the language is, to some extent, similar to that of Extended Temporal Logics (ETL) [7]. It is defined over pairs $\mathcal{H} =< \mathcal{T}, \mathcal{S} >$ made up of a temporal domain $\mathcal{T}$ (a set of instants at a given granularity), and a $\mathcal{T}$-indexed sequence of valuation functions $\mathcal{S}$ that assign values to constant, function and relation symbols for each instant in the temporal domain. Function and relation symbols are global [7], which means that they have the same interpretation over all valuation functions in $\mathcal{S}$, whereas constants may be either local (i.e. their interpretation may vary depending on the instant considered) or global. Numerical litterals (e.g. 1, 2, 3, ...) will be treated as global constants whereas constants denoting structural values of a history will be local.

In the following, f, f1 and f2 represent wff. The semantics of wff in a given model $\mathcal{H}$, between two instants i and j (j $>$ i) belonging to $\mathcal{T}$ and under a valuation ν of variables, is given by the satisfaction relation defined below. Intuitively, $\mathcal{H}, \nu, i, j \models$ f, iff there is an occurrence of the pattern defined by f starting at i and ending at j.

- $\mathcal{H}, \nu, i, i \models$ f (for an atomic formulae f) iff $\mathcal{S}_i, \nu \models f$ following the definition of $\models$ in first-order logics
- $\mathcal{H}, \nu, i, j \models [x \leftarrow t]f$ iff $\mathcal{H}, \nu[x \leftarrow a], i, j \models f$ where a denotes the value of term t under model $\mathcal{S}_i$ and valuation ν and $\nu[x \leftarrow a]$ denotes the valuation equal to ν except that it assigns value a to variable x
- $\mathcal{H}, \nu, i, j \models f1$ or $f2$ iff $\mathcal{H}, \nu, i, j \models f1$ or $\mathcal{H}, \nu, i, j \models f2$
- $\mathcal{H}, \nu, i, j \models$ not f iff not $\mathcal{H}, \nu, i, j \models f$

- $\mathcal{H}, \nu, i, j \models f1; f2$ iff there is a $k \in \mathcal{T}$, $k \geq i$, such that $\mathcal{H}, \nu, i, k \models f1$ and $\mathcal{H}, \nu, k+1, j \models f2$
- $\mathcal{H}, \nu, i, j \models (f)^1$ iff $\mathcal{H}, \nu, i, j \models f$
- $\mathcal{H}, \nu, i, j \models (f)^n$ $(n > 1)$ iff there exists a $k \geq i, k \in \mathcal{T}$ such that $\mathcal{H}, \nu, i, k \models f$ and $\mathcal{H}, \nu, k+1, j \models (f)^{n-1}$
- $\mathcal{H}, \nu, i, j \models (f)^+$ iff there is an $n \geq 1$ such that $\mathcal{H}, \nu, i, j \models (f)^n$
- $\mathcal{H}, \nu, i, j \models (f)^{\theta d}$ iff there is an $n \geq 1$ such that $\mathcal{H}, \nu, i, j \models (f)^n$ and $(i - j)\ \theta\ d$

We will be using the following abbreviations (f1 and f2 being wff formulae):

- $f1; ; f2 \equiv f1; (true)^+; f2$
- $f1; \theta d; f2 \equiv f1; (true)^{\theta d}; f2$

We now introduce a pattern-matching operator Grep on histories which checks if a given history contains at least one occurrence of a pattern.

Grep: History(T), Pattern $\longrightarrow$ boolean
Grep(h, p) = true $\Leftrightarrow$ there exist i and j in TDomain(h) such that $< TDomain(h), \mathcal{M} >, \nu, i, j > \models p$.

$\mathcal{M}$ is the sequence of interpretations in which relation symbols and global constant symbols are interpreted as usual, function symbols are interpreted by the corresponding functions, attributes and methods in the database schema, and the only local constant symbol appearing in pattern p is interpreted at each instant I $\in TDomain(T)$ by the structural value of h at instant I. Valuation ν is determined by the context (program or query).

This operator is useful for reasoning about succession in time as shown by the following queries:

R.4 : ***Pattern-matching.*** How many persons went skiing just after shopping and took dinner at most three hours after skiing

```
count (select p
       from ThePersons as p
       where Grep(p.activity,
                  ||act = "shopping";
                   (act = "skiing")+;< ♯"3 hours";
                   act = "dinner"||))
```

{ *notice that the only free symbol in the pattern is act, so this symbol represents the structural value of the history of activites at a given instant* }

In addition, this operator can be combined with the temporal grouping operator to express rather complex aggregate operations on histories in an elegant manner:

R.5 : ***Temporal grouping and pattern-matching.*** How many persons went everyday skiing after lunch[3].

```
count (select p
from ThePersons as p
where Empty((p.activity ! Day) Γif
            λ<i,h> • not(Grep(h,||act = "lunch";;
                                 act = "skiing"||))))
```

{ *p went everyday skiing after lunch* $\Leftrightarrow$ *there is no day in which p didn't ski after lunch* }

Finally, the following query shows the use of the assignment operation:

R.6 : ***Temporal grouping and pattern-matching.*** How many persons did the same activity twice on the same day

```
count (select p
from ThePersons as p
where not(Empty((p.activity ! Day) Γif
         λ<i,h> • Grep(h,|| [a ← act](true;
                           act ≠ a;; act = a ||)))))
```

{ *p did activity a, then stopped doing it, and subsequently restarted doing it* }

3.3 Pattern-matching in non-convex temporal domains

The notion of sequencing that we introduced above does not take into account the fact that the temporal domain of a history may contain "holes". For instance, lets consider the history of employees' departments. The temporal domain of this history represents the instants at which the employee is working in the company. Since the employee may quit the company and subsequently reinstate it, this temporal domain may contain "holes". The query "has e changed from the toys department to the drugstore department", has thus two interpretations: it may mean "is there an instant such that e's department at instant i is *toys* and at i+1 it is *drugstore*", or it may mean "are there two instants i and j such that j > i, e's department at instant i is *toys* and it is *drugstore* at j, and for all instants k such that i < k < j, e is not in the company".

To tackle this problem, a new sequencing operation is introduced (noted %) such that f1 % f2 is matched if there are four instants $i_1 \leq i_2 < j_1 \leq j_2$ such that f1 is matched between i_1 and i_2, f2 is matched between j_1 and j_2 and there is no instant in $\mathcal{T}$ between i_2 and j_1. In other words: f2 occurs after f1 and nothing occurs between them. To distinguish, this sequencing operation from the one presented above we term them respectively *soft* and *hard* sequencing.

With this distinction the two interpretations of the query presented above may be expressed respectively by patterns
$||$*dep = "toys" ; dep = "drugstore"*$||$ and
$||$*dep = "toys" % dep = "drugstore"*$||$.

[3]Recall that the attribute activity is defined at the granularity "minute"

The distinction between *soft* and *hard* applies to the repetition operators. Formally speaking, the syntax of the pattern-matching language is extended by the following rules:

- Soft sequencing: if f1 and f2 are wff then f1 % f2 is a wff.
- Soft repetition: if f is a wff, θ a comparison operator ($<$, $=$, etc), n an integer term and d is duration term, then $(f)^{\sim n}$, $(f)^{\sim +}$ and $(f)^{\sim \theta d}$ are wff too.

The semantics of these new syntactical rules are:

- $\mathcal{H}, \nu, i, j \models f1 \ \% \ f2$ iff there is a $k \in \mathcal{T}$ such that $\mathcal{H}, \nu, i, k \models f1$ and there exists $l > k, l \in \mathcal{T}$ such that $\mathcal{H}, \nu, l, j \models f2$ and for all m such that $k < m < l$, $m \notin \mathcal{T}$
- $\mathcal{H}, \nu, i, j \models (f)^{\sim 1}$ iff $\mathcal{H}, \nu, i, j \models f$
- $\mathcal{H}, \nu, i, j \models (f)^{\sim n}$ $(n > 1)$ iff there exists a $k \geq i, k \in \mathcal{T}$ such that $\mathcal{H}, \nu, i, k \models f$ and there is $l > k$ such that $\mathcal{H}, \nu, l, j \models (f)^{\sim (n-1)}$ and for all m such that $k < m < l$, $k \notin \mathcal{T}$
- $\mathcal{H}, \nu, i, j \models (f)^{\sim +}$ iff there is an $n \geq 1$ such that $\mathcal{H}, \nu, i, j \models (f)^{\sim n}$
- $\mathcal{H}, \nu, i, j \models (f)^{\sim \theta d}$ iff there is an $n \geq 1$ such that $\mathcal{H}, \nu, i, j \models (f)^{\sim n}$ and $(i - j) \ \theta \ d$

4 Related works

Many temporal extensions to data models have been proposed both in the relational framework [20, 19] and in the object-oriented one [18]. However, almost none of them integrate features such as temporal grouping and pattern-matching.

To our knowledge, the only temporal data model that has been extended to integrate pattern-matching facilities is ParaSQL [4]. In this work, a spatio-temporal pattern-matching language is proposed which allows to express queries about succession in time in addition to spatial relations between regions. Unlike our work, the semantics of this temporal pattern language is based on the notion of *time cursor*. We think that regular expression operators are more suitable for this kind of language because their usage is widespread and because they have a well-defined declarative semantics. Moreover, there are several automata-based techniques that may be used to efficiently implement them. Indeed, recent results [2] show that Kleene's theorem on regular expressions can be extended to timed regular expressions, which means in particular, that it is possible to build a timed automaton recognizing the language defined by a timed regular expression.

In the context of video databases, [15] describes an implementation of a pattern-matching language based on regular expressions. Furthermore, in active databases, many languages for defining patterns of events have been proposed [5] which integrate regular expression operators too.

An issue closely related to temporal pattern-matching queries is that of finding approximate matches of a sequence in a collection of time-series. Such operation may be used to answer queries such as *"find companies whose stock prices move similarly"* [8].

As most of the grouping operators that have been defined in the relational, object-oriented and datawarehouse framework (e.g. the *classification* operator defined in [10]), the temporal grouping operation introduced in this paper derives from the following general operation : given a mapping F from set S to set Q, classify the elements in a subset R of S according to their image by F. In particular, our temporal grouping operator is, to some extent, similar to that of TSQL2. The main originality of our approach is that we define temporal grouping as a mapping from histories to "histories of histories". As a result, the operator can be easily composed with other operators of the historical model.

Finally, the issue of temporal grouping is related to the one of constructing calendars [13, 14], although this latter deal with pure temporal values (instants and collections of instants) instead of histories.

5 Conclusion

We have briefly presented a historical model which unifies many of the concepts and functionalities which have been recognized as necessary for extending DBMS with temporal facilities, and we have integrated into it two new operators dealing with temporal grouping and pattern-matching queries.

The temporal grouping operator structures a history in a diary-like way. This operator may be used either for visualization or aggregation purposes.

The proposed temporal pattern-matching language is based on classical logics and timed regular expressions. This allows to give a declarative semantics to the language similar to that of many temporal logics. In addition, regular expressions provides a framework for efficient implementation.

The suitability of our model and the proposed extensions in regard to user requirements have been tested on several applications taken from economical databases, GIS and video databases. Moreover, the feasibility of our proposal has been validated by a prototype implementation on top of the O_2 DBMS.

We are currently implementing our pattern-matching operator. This implementation is based on techniques originally developed in the context of synchronous programming languages [16]. The idea behind these techniques is to translate properties expressed as regular expressions over boolean variables into an equivalent boolean dataflows network (a sequential circuit) which explores all the branches of a non-deterministic automaton whose size is proportional to that of the regular expression involved. The grep algorithm has thus a reduced space complexity and a time complexity proportional to the number of snapshots in the history.

We also plan to define a DDL and a DML based on the TEMPOS model which seamlessly extends ODMG's ODL and OQL, and to study optimization issues on these languages.

Acknowledgments: we wish to thank Jean-François Canavaggio who worked on the TEMPOS model definition during his PhD thesis as well as Sonia Chardonnel and the LAMA laboratory at Grenoble with whom we modeled the application presented in this paper.

References

[1] M. Adiba. STORM : an object-oriented, multimedia DBMS. In K.Nowsu, editor, *Multimedia Database Management Systems*, chapter 3, pages 47–88. Kluwer Academic Publishers, 1996.

[2] E. Asarin, P. Caspi, and O. Maler. A Kleene Theorem for Timed Automata. In *proc. of the 12th IEEE Symposium on Logic in Computer Science*, Warsaw, Poland, June 1997.

[3] J.-F. Canavaggio. TEMPOS, un modèle d'historiques pour un SGBD temporel. Thèse de doctorat, Université Joseph Fourier, Grenoble (France), Novembre 1997.

[4] T. Cheng and S. Gadia. A pattern matching language for spatio-temporal databases. In *proc. of the 3rd International Conference on Information and Knowledge Management (CIKM)*, Gaithersburg, MD (USA), November 1994.

[5] C. Collet and T. Coupaye. Primitive and Composite Events in NAOS. In *actes des 12e Journées Bases de Donnéees Avancées*, Cassis (France), août 1996. pages 331–349.

[6] M. Dumas, J.-F. Canavaggio, M.-C. Fauvet, and P.-C. Scholl. TEMPOS: managing time and histories on top of OO DBMS. In *proc. of EDBT'98 demo session*, Valencia (Spain), March 1998.

[7] E.A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume 2. Elsevier Science Publisher, 1990.

[8] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast Subsequence Matching in Time-Series Databases. In *proc. of ACM SIGMOD*, pages 419 – 429, 1994.

[9] M.-C. Fauvet, J.-F. Canavaggio, and P.-C. Scholl. Modelling histories in object DBMS. In *proc. of the 8th International Conference on Database and Expert Systems Applications (DEXA)*, Toulouse (France), September 1997. Springer Verlag. LNCS 1308.

[10] M. Gyssens and L. Lakshmanan. A fondation for multi-dimensional databases. In *Proceedings of the 23rd VLDB conference*, Athens, Greece, 1997.

[11] J.P. Held and J.V. Carlis. Match : A new high-level relational operator for pattern matching. *Comm. of the ACM*, 30(1):62 – 75, January 1987.

[12] C.-S. Jensen, J. Clifford, R. Elmasri, S. Gadia, P. Hayes, and S. Jajodia. A consensus glossary of temporal database concepts. *ACM SIGMOD Record*, 23(1), March 1994.

[13] B. Leban, D. McDonald, and D. Forster. A representation for collections of temporal intervals. In *proc. of the 5th National Conference on Artificial Intelligence (AAAI'86)*, Philadelphia, 1986.

[14] J. Y. Lee, R. Elmasri, and J. Won. Specification of calendars and time series for temporal databases. In *proc. of International Conference on Conceptual Modeling - ER'96*, Cottbus, Germany, October 1996. Springer Verlag, LNCS No 1157.

[15] K. Nakayama, K. Yamaguchi, and S. Kawai. I-Regular Expression : Regular Expression with Continuous Interval Constraints. In *proc. of the 6th CIKM conference*, pages 40 – 50, Las Vegas, USA, November 1997.

[16] P. Raymond. Recognizing regular expressions by means of dataflows networks. In *proc. of the 23rd International Colloquium on Automata, Languages, and Programming, (ICALP'96)* Paderborn, Germany. LNCS 1099, Springer Verlag, July 1996.

[17] P.-C. Scholl, M.-C. Fauvet, and J.-F. Canavaggio. Un modèle d'historique pour un SGBD temporel. *TSI, Numéro thématique "Bases de données"*, 17(3), Mars 1998.

[18] R. T. Snodgrass. Temporal object-oriented databases: a critical comparison. In W. Kim, editor, *Modern database systems. The object model, interoperability and beyond*, chapter 19. Addison Wesley, 1995.

[19] R. T. Snodgrass, editor. *The TSQL2 temporal query language*. Kluwer Academic Publishers, 1995.

[20] A. U. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. Snodggrass, editors. *Temporal Databases*. The Benjamins/Cummings Publishing Company, 1993.

[21] X. S. Wang, S. Jajodia, and V. S. Subrahmanian. Temporal modules : an approach toward federated temporal databases. *Information Systems*, 82, 1995.