# Modeling Synchronized Time Series

Zoé Faget[1,4], David Gross-Amblard[2],
Philippe Rigaux[1,3] and Virginie Thion-Goasdoué[1]

[1] Lamsade, Univ. Paris-Dauphine, [2] Le2i, Univ. Bourgogne,
[3] WebDam, INRIA-Saclay, [4] Armadillo.

**Abstract.** We consider the class of applications that manage *time series* (TS) and propose a data model and a query language that let these applications manipulate TS data sets at a logical level. We introduce the concept of *synchronized time series* (STS) to model the alignment of several time series in a common time space. We show how this concept helps to analyze and compare information extracted from time series, and how it constitutes a convenient tool that supports an extended set of operations. The main contribution of the paper is a formal query language that combines generic operators on TS with user-defined functions. The language can be evaluated in closed form over STS data sets and gives rise to rewriting and optimization techniques. Throughout the paper, we illustrate our approach with several examples drawn from a few representative TS applications.

## 1 Introduction

Many applications deal with time-dependent information which must be monitored, analyzed, and compared. Such information is commonly represented as *time series*, a well-established concept which is ubiquitous in business, financial and industrial applications. Generally speaking, a time series is an ordered sequence of values taken at equal time intervals. Their traditional usage is twofold: (i) infer an understanding of the process that produces the time-dependent values, through analysis techniques, and (ii) use the inferred model to forecast and monitor ongoing production processes. Some of the traditional domains that explicitly produce and exploit time series are, among others, sales forecasting, stock market analysis, process and quality control [1].

The concept of time series also provides a convenient framework for modeling time-dependent data such as business indicators, music content, moving objects, sensor data, etc. In all these application areas, data sets can be viewed as functions that map a time domain (not necessarily the standard calendar) to an application-dependent domain (e.g., sounds/notes, moving object position, meteorological information or any other sensor-collected value). Common to all these applications is also the involvement of the temporal parameter in data processing operators. They can either directly consider the temporal domain (e.g, scaling, windowing, temporal shifts), process values by indirectly considering the implicit order induced by the temporal information, or finally combine these aspects.

The large volume of data produced by intensive time series application calls for an efficient support. This can be achieved through advanced techniques for storage compression [26], algorithm design and implementation [4] and time series indexing [13, 16]. Such techniques are usually supported by a specialized data management system which releases applications from the burden of low-level and intricate implementation concerns. To our knowledge no such system exists at this moment for the domain of time series management. Our objective in this paper is to lay the ground for such a system by proposing one of its core components, namely a TS logical data model and its associated query language.

Our model slightly extends the concept of time series to *Synchronized Time Series* (STS). Intuitively, a STS is a group of TS built on a shared time domain, and modeled as functions. Introducing this concept (wich covers and extends the simple case of single, independent time series) turns out to provide an appropriate framework to enable a representation space where TS can be consistently aligned, compared, and merged. In particular, a new STS intance can be obtained by *deriving* time series, a concept borrowed from the mathematical notion. The derivation of a time series $s$ at an instant $\alpha$ typically "shifts" or "scales" the underlying time domain to define a TS $s'$ at $\alpha$, and provides a powerful and flexible mechanism for general time series alignment.

We formally describe STS as an extension of the relational model, and provide an algebra that operates in closed form over collections of STS. We show that (i) this algebra expresses usual operations on TS (e.g., moving average), (ii) achieves high expressiveness through unbounded composition of operators and (iii) natively incorporates the ability to introduce user-defined function in query expressions, in order to meet the needs of a specific application domain. Finally, we introduce a user-friendly query language to express algebraic operations. We believe that our approach defines a sound basis for the implementation of a specialized query language devoted to time series management at large scale.

An implementation of this language is currently in progress as a central building block in the architecture of a music digital library. We briefly mention this application as an illustration of the practical scope of our approach.

The rest of this paper first exposes the motivation and the context of our work, and smoothly introduces the design of our model through examples drawn from some illutrative applications (Section 2). The data model is given in Section 3 and the user query language, illustrated with example queries, in Section 4. Related work is discussed in Section 5. Due to space limitations, the present text does not cover technical details related to algebraic rewriting and query evaluation, but the interested reader is referred to the long version of the paper available online at *http://neuma.irpmf-cnrs.fr*.

## 2 Overview

Our approach is based on the idea that the notion of time series management corresponds, at the core level, to a limited set of basic operations that can support time series applications beyond their specifics. We also take account of the fact

that the wide range of domains that deal with time series calls for the ability to introduce domain-dependent functions at early steps of the query evaluation process. Modeling the invariant of time series, defining the generic operations and combining this data model with user-defined operations are the main goals of the present paper.

We now introduce a few applications from very distinct domains, showing their main requirements from a data management point of view, as well as some of the concepts that underly our work.

### Industrial application

Typical time series produced by industrial applications relate to the production and/or consumption of products or services. Consider for instance an eletricity producer which monitors the daily consumption of electricity of its consumers for a year. We can represent the time series *elec_ daily* (or *elec_ d* for short) as follows. The time domain $\mathcal{T}$ is the sequences of days from 0 to 364 (for simplicity we assume day 0 is a sunday), denoted *Calendar*. The domain **dom** is the set of positive real values $\mathbb{R}+$. Such a time series is subject to the following operations:

1. *Max consumption in a given period*. Choose a restricted period and compute the min/max consumption or any other scalar value for this period.
2. *Moving average*. Report the weekly average consumption at every day of the year (note that the result is a time series).
3. *Create Aggregated TS*. Report with every sunday the cumulated consumption of the week.
4. *Produce and compare new time series*. Extract a TS for each month of the year, superpose these 12 TS and compute statistic indicators (for instance, take the consumption as a random variable and compute its variance for each day).

Note that the last example implies a change of the time domain which switch from *Calendar* to the mode abstract domain "days of a month". Note also that the superposition creates a set of TS that share this new domain. Actually, this sharing of a common temporal domain, denoted *synchronization* is the following, is an implicit requirement for computing new values from the vertical alignment of TS.

### Moving objects

Consider now a company that manages a fleet of vehicles for goods delivery. Each day, the route of each vehicle is determined according to all the spots that must be visited. These trajectories share a common temporal domain (here, the daily hours of the current day) and can be modelled as synchronized (with the informal meaning explained above) time series that map time values to locations on a map. Queries of choice in that case include:

1. *Project/selection*: show the trajectory of a specific vehicle during a specific period of the day.
2. *Joins*: show the vehicles which are in the same region at the same time.
3. *Nearest neighbor*. Given a vehicle $v$, find at each instant the closest other vehicle of the fleet.

Essentially, this application shares most of its fundamental features with the above industrial example, apart the time domain of interest and the application-dependent operations applicable to the TS values.

### Music modeling
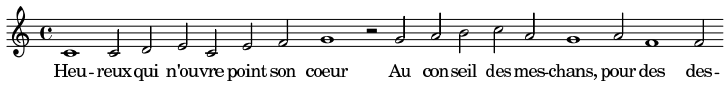



**Fig. 1.** A monodic score

Consider finally a digital library that stores music information and provides query services on music collections. A music piece consists of one or several *parts* which can be modelled as time series, each represented by a score. Fig. 1 is a simple example of a *monodic* piece, but *polyphonic* music pieces (Fig. 2) exhibit a synchronization of several parts.
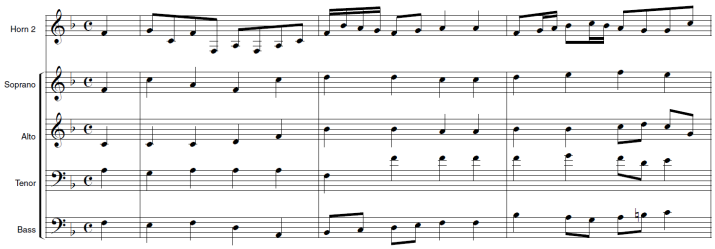



**Fig. 2.** A polyphonic score

The temporal domain of interest here is no longer the *Calendar*, but relates to a quite abstract representation of the temporal features. Essentially, these features are (i) the order of the musical "events" (notes, rests), (ii) the relative duration of the events, and (iii) the required synchronization of the parts. Here are a few examples of queries:

1. *Get the scores whose lyrics contains the word 'conseil'* (selection);
2. *Get the melodic part that corresponds to the word 'conseil'* (selection and temporal join);
3. *Find a melodic pattern* (search by similarity).

**Summary and main concepts**

The examples that precede cover a representative sample of the main operations involved in TS management, and provide guidelines to the design of a generic data model. Basically, we model a TS as a function from a *temporal domain* to an *application domain*. Regarding the temporal domain, the following remarks are noteworthy. First, the "time" dimension cannot be reduced to a mere timestamping of events in terms of day, hours, second, but must rather be seen, at a more abstract level, as an ordered, enumerable set of values, isomorphic to $\mathbb{N}$. Second, and more importantly, a temporal domain (e.g., a set $\mathcal{T}$ isomorphic to $\mathbb{N}$) is not an absolute reference shared by all the time series in a TS collection, but must rather be interpreted as a function domain possibly shared by several TS, and used by them as a common yardstick to compare and measure their respective features (order, duration, synchronization).

We denote as *synchronized time series* (STS) a group of TS that share a common time domain $\mathcal{T}$. STS constitute the basic building block of our model, and play the role of tuples in relational systems. They are assembled in finite *sets* of STS, called *extended relations*, having the same schema. We finally model a database as a set of extended relations.

## 3 The Data Model

We now embark in the formal definition of the model.

### 3.1 Preliminaries

We assume a (values) domain $\mathcal{D} = (\mathbf{dom}, \mathcal{O}, \Gamma)$, where **dom** is a set of *values*, $\mathcal{O}$ is a set of *user operations* on **dom** and $\Gamma$ is a set of *user aggregation functions*. Each *op* of arity $k$ in $\mathcal{O}$ maps a $k$-tuple $[v_1, \ldots, v_k]$ of $\mathbf{dom}^k$ to **dom**. Each $\gamma$ in $\Gamma$ maps a sequence of values from **dom**, $< v_1, \ldots, v_n >$, to **dom**.

The set **dom** always contains two distinguished values: the *neutral value* $\top$ and the *null value* $\bot$. The Boolean operations $\wedge$ (conjunction) and $\vee$ (disjunction) verify, for any $a \in \mathbf{dom}$, $a \wedge \bot = \bot$, $a \wedge \top = a$ and $a \vee \bot = a \vee \top = a$ ($\vee$ and $\wedge$ are commutative). Unless specified otherwise, for any $op \in \mathcal{O}$, *op* is extended to $\wedge$ when at least one operand is $\bot$ or $\top$, i.e. we have for any $a \in \mathbf{dom}$, $op(\bot, a) = \bot$, $op(\top, a) = a$.

The values domain $\mathcal{D}$ is application-dependent. Its instantiation in a system defines the functions and aggregation operators that can be introduced in queries. When no ambiguity arises, we refer to $\mathcal{D}$ as "the domain".

A time domain $\mathcal{T}$ is a countable ordered set isomorphic to $\mathbb{N}$. We define as $\mathcal{L}$ the class of functions from $\mathcal{T}$ to $\mathcal{T}$, also known as internal time functions (ITF). An important sub-class of $\mathcal{L}$ is the set of linear functions of the form $t \rightarrow \alpha t + \beta$. We denote them *temporal scaling functions* in what follows, and further distinguish the families of *warping functions* of the form $warp^{\alpha} : \mathcal{T} \rightarrow \mathcal{T}$, $t \mapsto \alpha t$ and *shifting functions* $shift^{\beta} : \mathcal{T} \rightarrow \mathcal{T}$, $t \mapsto t + \beta$.

Whenever such an operation is well-formed (with the standard meaning), two functions from $\mathcal{L}$ can be composed with the classical mathematical operator $\circ$.

**Definition 1 (Time series)** *Let $\mathcal{D}$ be a values domain on **dom** and $\mathcal{T}$ a time domain. A* time series *$s$ on $\mathcal{D}$ is a function from $\mathcal{T}$ to **dom**. $(TS)$ denotes the set of time series.*

We assume the reader familiar with the basic concepts of the relational model and denote by $(\mathcal{R})$ the set of relation names, $(\mathcal{A})$ the set of attributes names, and $(\mathcal{S})$ the set of time series names.

**Definition 2 (Schema of $(TS)$-relations)** *Let $R$ be the name of a relation, $A_1, \cdots, A_k$ be attribute names from $(\mathcal{A})$, and $S_1, \cdots, S_p$ be TS names from $(\mathcal{S})$. Then $R(A_1, \cdots, A_k, S_1, \cdots, S_p)$ is a relation schema.*

An *instance* of $R$ is a (finite) set of *tuples* $(a_1, a_2, \cdots, a_k, s_1, \cdots, s_p)$, where each $s_i$ is an instance of $S$ (i.e., a time series), and $a_1, \ldots, a_k$ are instances of relational attributes in atomic domains.

**Example 1** *The following schemas correspond respectively to energy consumption, recording the daily consumption of a group of consumers, moving objects trajectories, and music scores (representing piano/voic songs). Each schema mixes atomic-valued attributes and time series. We denote by **TS([dom])** the type of the latter.*

1. *Energy (Id : **int**, consumer: **string**, consumption: **TS(float)**).*
2. *Routes (Id : **int** , name : **varchar**, traj: **TS(point)**)*
3. *Score (Id : **int**, composer: **string**, voice: **TS(vocal)**, piano: **TS(polyMusic)**).*

### 3.2 The time series algebra $\textsc{Alg}_{(TS)}$

We will now define our algebra which consists of two sets of operators: the time series algebra $\textsc{Alg}_{(TS)}$ $(\circ, \oplus, \mathbf{A})$ and the extended relational algebra $\textsc{Alg}_R$. Since $\textsc{Alg}_R$ is very close to the traditional relational algebra, we first focus on $\textsc{Alg}_{(TS)}$.

**Core operators** Any expression from $\textsc{Alg}_{(TS)}$ takes one or several time series as input and produces a time series. It consists of the following primary operators: the *composition* operator $\circ$, the addition operator $\oplus$, and an *aggregation* mechanism which combines a *derivation* operator with a user function from $\Gamma$. Definitions follow.

**Definition 3 (External composition)** *Let $s$ be a time series and $l \in \mathcal{L}$. Then $s \circ l$ is the time series defined as*

$$[s \circ l](t) = s(l(t)).$$

**Example 2** *The following examples apply* scaling *time functions (linear functions on $\mathcal{T}$).*

1. *Recall that $shift^n$ is a shift function parametrized by a constant $n \in \mathbb{N}$. For any $s \in (TS)$ and $t \in \mathcal{T}$, $s \circ shift^n(t) = s(t+n)$. In other words, $s \circ shift^n$ is the time series obtained from $s$ by ignoring the first $n$ events.*
2. *Consider the* $\text{warp}^\alpha$ *function, which scales the time values by a positive integer $\alpha$. For any $\alpha \in \mathbb{N}^*$, for any $t \in \mathcal{T}$, $warp^\alpha(t) = \alpha t$. Then $s \circ warp^\alpha$ is the time series extracted from $s$ where only one out of $\alpha$ instants is considered. This family of functions comes handy when studying frequency peaks. For example, when the time domain $\mathcal{T}$ is* Calendar*, $warp^7$ focuses on a day of the week.*
3. *If we take the* Calendar *time domain (assuming the Calendar starts on a sunday), then for any $s$ defined on* Calendar *into an alphabet* ***dom****, $s_{monday} = s \circ (t \mapsto 7t+1) = s \circ shift^1 \circ warp^7$.*

**Definition 4 (Addition operator)** *Let $s, u \in (TS)$ and $op$ an operation on* ***dom****. Then $s \oplus_{op} u$ is the time series defined as*

$$[s \oplus_{op} u](t) = op(s(t), u(t)).$$

**Example 3** *Let $s_1$ and $s_2$ be two time series defined on* Calendar *into* ***dom****$= \mathbb{R}$, each representing the evolution of a stock market value over a year. Taking the minus operator on $\mathbb{R}$, we can compute the time series $s_1 \oplus_- s_2$. Positive values of this new time series occur when the stock value of $s_1$ is higher than the stock value of $s_2$.*

**Example 4** *Let* $\text{dom}_{\text{notes}}$ *be the alphabet of all musical notes, and* $\text{dom}_{\text{int}}$ *the alphabet of all musical intervals. We can define an operation from* $\text{dom}_{\text{notes}} \times \text{dom}_{\text{notes}}$ *into* $\text{dom}_{\text{notes}}$ *called the* harm *operator, which takes two notes as input and computes the interval between these two notes. Given two musical parts $V_1 =$*soprano *and $V_2 =$*alto*,*

$$V_1 \oplus_{harm} V_2$$

*represents the harmonic progression (i.e., the sequence of intervals between the two voices).*

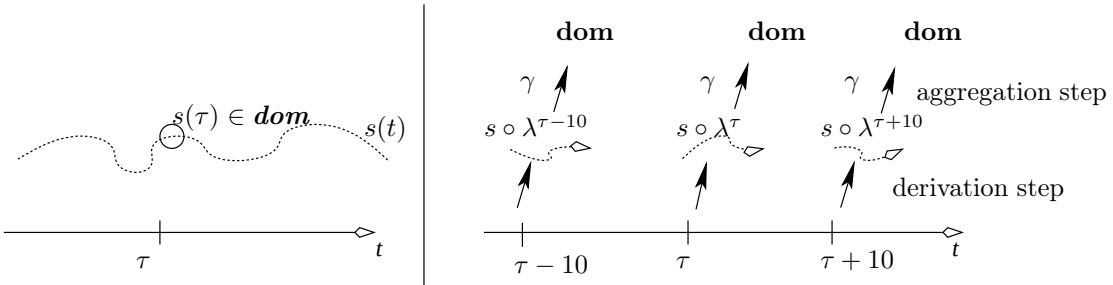
We now turn to the aggregation mechanism. First, note that the operators defined so far operate on a time instant basis, and manipulate atomic values from **dom** delivered by a function $s$ at an instant $t$. While useful, this is not sufficient in terms of expressive power. A typical operation that cannot be obtained is the "*windowing*" which, at each instant, considers a local part of a function $s$ and derives a value from this restricted view.

We introduce an abstract and general mechanism inspired by the mathematical concept of derivation $s'$ of a function $s$ at a given point $x$. The derivation operator takes a time series $s$ as input and produces a sequence of time series $< s'_1, s'_2, \cdots >$, one for each instant. This structure does not fit in our model as defined so far. A possible solution would be to consider that the set of time series is itself a domain $\mathcal{D}$, in order to model time series of time series. We rather choose to pair the derivation operator with an aggregation function, in the spirit of the popular nesting/grouping mechanism used in SQL aggregation. Thus, aggregation in $\text{ALG}_{(TS)}$ operates in two steps:

1. first, at each instant $\tau$, a TS $s'_\tau$ is derived from $s$ thanks to a *derivation operator* $d_\lambda$ where $\lambda \in \mathcal{L}$;
2. second, a *user aggregation function* $\gamma \in \Gamma$ is applied to $s'_\tau$, yielding an element from a values set **dom**.

Fig. 3 illustrates this two-steps process. At each instant $\tau$ a new function is derived. At this point, we obtain a sequence of time series, each locally defined with respect to an instant of the time axis. This sequence corresponds to local views of $s$, possibly warped by temporal functions. Note that this intermediate structure is not covered by our data model.

The second step applies an aggregation function $\gamma$. It takes one (or several, depending on the arity of $\gamma$) derived series, and produces an element from **dom**. The combination of the derivation and aggregation steps results in a time series that complies to the data model.




a. Function $s(t)$ b. Sequence of derived functions $d_\lambda(s)$

**Fig. 3.** The derivation/aggregation mechanism

We now embark in the definitions.

**Definition 5 (Derivation operator)** *Let $\lambda = \{\lambda^i, i \in \mathbb{N}\}$ a family of internal temporal functions and $s$ a time series. The* derivation *of $s$ with respect to $\lambda$ is a mapping from (TS)→(TS)$^{\mathbb{N}}$:*

$$d_\lambda(s) = (s \circ \lambda^0, s \circ \lambda^1 \ldots, s \circ \lambda^n, \cdots).$$

**Example 5** *When windowing a time series, we use the shift function :*

$$d_{shift}(s) = (s \circ shift^0, s \circ shift^1, \ldots, s \circ shift^n, \cdots).$$

*Each element $n$ is a "view" of $s$ starting at $n$.*

Combining the derivation operator with a user aggregation function of the values domain yields the Aggregation operator of $\text{ALG}_{(TS)}$.

**Definition 6 (Aggregation)** *Let $\gamma$ be an aggregation function of arity 1, and $\lambda$ an internal temporal function. Then, for any time series $s$, $\boldsymbol{A}_{[\gamma,\lambda]}(\mathrm{s})$ is an expression of the $\text{ALG}_{(TS)}$ algebra that defines the TS:*

$$[\boldsymbol{A}_{[\gamma,\lambda]}(\mathrm{s})](\mathrm{t}) = \gamma(\mathrm{s} \circ \lambda^{\mathrm{t}}).$$

Recall that $s \circ \lambda^i$ is a time series and $\gamma$ is a function from $(TS) \rightarrow \mathbf{dom}$. Therefore, $\mathbf{A}_{[\gamma,\lambda]}(\mathrm{s})$ is indeed a time series. The extension to aggregation functions of arity $k > 1$ is straightforward.

**Example 6 (Moving average)** *For any integer $\tau \in \mathbb{N}$, $s \circ shift^\tau$ defines the time series derived from $s$ where the first $\tau$ events of $s$ are ignored. Let $avg_{|w|}$ be the user aggregation function which computes the average value of the first $|w|$ events of a time series. The time series that gives the moving average over the window $[t, t + |w|]$ for each instant $t$ is obtained by:*

$$\boldsymbol{A}_{[\mathrm{avg}_{|\mathrm{w}|},\mathrm{shift}]}(\mathrm{s}).$$

**Composed operators.** For convenience, we promote in $\text{ALG}_{(TS)}$ a few operators that correspond to useful and intuitive operations. They can be derived from the core operators presented above.

Our first composed operator is union. By convention, the union operates on two time series $s$ and $u$ only if they don't have conflicted time domain, i.e., if $s(t) \neq \perp$ then $u(t) = \perp$ and vice versa.

**Definition 7 (Union)** *Let $s$ and $u$ be two TS such that their time domains do not conflict. Then $s \uplus u$ is the time series defined as*

$$[s \uplus u](t) = \begin{cases} s(t), \text{ if } s(t) \neq \perp \\ u(t), \text{ if } u(t) \neq \perp \\ \perp \text{ otherwise.} \end{cases}$$

The union operator can be expressed by means of the addition operator: $s \uplus u = s \oplus_\vee u$.

The selection operation that selects parts of a time series $s$ with respect to a condition on values is also a composed operator. Our model encodes condition on values by a mapping $\phi : \mathbf{dom} \rightarrow (\perp, \top)$.

**Definition 8 (Selection)** *The selection operation $\sigma_\phi(s)$, where $s$ is a time series and $\phi$ is a condition on* ***dom****, is defined as follows:*

$$[\sigma_\phi(s)](t) = \begin{cases} s(t), \text{ if } \phi(s(t)) \\ \bot \text{ otherwise.} \end{cases}$$

Selection can be expressed with core operators $\sigma_\phi(s) \equiv s \oplus_\wedge (\phi \circ s)$.

The last composed operator presented here is the projection that selects parts of a time series $s$ with respect to a condition on the time domain. Such a condition is encoded by a mapping $\psi : \mathcal{T} \to (\bot, \top)$ (note that $\psi$ is a time series in the trivial alphabet $(\bot, \top)$).

**Definition 9** *Let $\psi$ be a condition on time. The projection operation $\pi_\psi(s)$ where $s$ is a time series is defined as follows:*

$$[\pi_\psi(s)](t) = \begin{cases} s(t), \text{ if } \psi(t) \\ \bot \text{ otherwise.} \end{cases}$$

Projection operation can be expressed in $\text{ALG}_{(TS)}$ algebra, thanks to the equivalence $\pi_\psi(s) \equiv \psi \oplus_\wedge s$. We conclude this presentation of the model with an extended example.

**Example 7 (Calendar queries)** *We consider electric consumption time series, and require the computation of the following results: weekly consumption ($elec_w$), average daily consumption for each week ($elec_{avg}$), and weeks where sunday was not the day of maximum consumption ($elec_{sunday}$). Let $shift$ be the following family of internal temporal functions:*

$$shift = (shift^0, shift^7, \ldots, shift^{52\times 7}).$$

*We also need three user agregation functions, parameterized by an integer $\alpha$, which all apply to the temporal window $[0, \alpha]$: $sum_\alpha$, $avg_\alpha$, and $max_\alpha$. The time series $elec_w$ and $elec_{avg}$ are obtained by applying the derivation-aggregation operator:*

$$elec_w = \boldsymbol{A}_{\text{sum}_7,\text{shift}}(\text{elec}_\text{d}),$$

$$elec_{avg} = \boldsymbol{A}_{\text{avg}_7,\text{shift}}(\text{elec}_\text{d}).$$

*To find weeks where sunday is not the maximum consumption day, we first compute the time series of electric consumption on sundays:*

$$elec_{sunday} = elec_d \circ warp^7$$

*We then compute the time series of weekly maximum consumption by applying the derivation-agregation operator*

$$elec_{max} = \boldsymbol{A}_{\text{max}_7,\text{shift}}(\text{elec}_\text{d}).$$

*We now have two time series from the time domain $\mathcal{T}$ of weeks (52 time events) into* $\boldsymbol{dom} = \mathbb{R}+$. *The difference operation is well defined on* $\boldsymbol{dom}$, *so we compute* $week_max \oplus_- elec_{sunday}$. *Finally, we apply a selection operator* $\sigma_{>0}$, *and we get our result.*

**Algebraic expressions.** All operators in $\text{ALG}_{(TS)}$ operate in closed form. *Expressions* of $\text{ALG}_{(TS)}$ are defined inductively as usual:

1. if $A$ is an attribute name, then $A$ is an expression (the identity on the domain of $A$);
2. if $e_1, \cdots, e_k$ are expressions of $\text{ALG}_{(TS)}$ and $O$ is an operator of arity $k$, then $O(e_1, \cdots, e_k)$ is an expression.

Expressions in $\text{ALG}_{(TS)}$ can be introduced in relational expressions, as explained next.

### 3.3 The relational algebra $\text{ALG}_R$

The $\text{ALG}_R$ algebra consists of the usual operators $\sigma$, $\times$, $\cup$ and $-$, along with an extended projection $\Pi$. Let $E$ be an expression of $\text{ALG}_R$ that defines a relation of schema $(A_1, \cdots A_k, S_1, \cdots, S_p)$. If $e_1, e_2, \cdots, e_q$ are valid expressions of $\text{ALG}_{(TS)}$ over this schema, then $\Pi_{[e_1,\cdots,e_q]}(E)$ is an expression of $\text{ALG}_R$. Its semantics over an instance $I$ is as follows:

**Definition 10 (Semantics of $\Pi$)** *Let $I$ be an instance of a relational schema and $E$ a valid expression of* $\text{ALG}_R$. *Then*

$$\Pi_{[e_1,\cdots,e_q]}(E) = \{e_1(r), e_2(r), \cdots, e_q(r) | r \in E(I)\}.$$

We also introduce an emptyness test $\emptyset^?$ that takes a time series as input and is modeled as: $\emptyset^?(s) = false$ if $\forall t, s(t) = \perp$, else $\emptyset^?(s) = true$. $\emptyset^?$ can be introduced in selection formulas of the relational $\sigma$ operator. Note that it allows to express conditions on time series as a whole, in contrast with the $\sigma$ operator of $\text{ALG}_{(TS)}$ which operates individualy on each temporal event.

**Example 8** *Consider the relation Score of Example 1. We want to select all scores featuring the word 'Ave' in part $V_1$. We need a pattern matching function $m : (lyrics) \to (\perp, \top)$ such that $m(word) = \top$ if word='Ave', else $\perp$. Lyrics not containing 'Ave' are transformed into the "empty" time series $t \mapsto \perp, \forall t$. The emptyness test can then be applied, hence the algebraic expression:*

$$\sigma_{\emptyset^?(V_3)}(\Pi_{[V_1,V_2,V_3:m\circ V_1]}(Score)).$$

**Example 9 (DTW)** *We want to compute, given a pattern $P$ and a score, the instant $t$ where the Dynamic Time Warping (DTW) distance between $P$ and $V$*

*is less than 5. First, we compute the DTW distance between $P$ and the voice $V_1$ of a score, at each instant, thanks to the following* $\text{ALG}_{(TS)}$ *expression:*

$$e \equiv \boldsymbol{A}_{\text{dtw}_\text{P},\text{shift}}(\text{V}_1)$$

*where $dtw_P$ is the DTW function. Expression e defines a time series that gives, at each instant $\alpha$, the DTW distance between $P$ and the sub-series of $V_1$ that begins at $\alpha$. A selection (from* $\text{ALG}_{(TS)}$*) keeps the values in e below* 5*, all others being set to $\perp$. Let $\psi$ be the formula that expresses this condition. Then:*

$$e' = \sigma_\psi(e).$$

*Finally, expression $e'$ is applied to all the scores in the Score relation with the $\Pi$ operator. An emptyness test can be used to eliminate those for which the DTW is always higher that 5 (hence, $e'$ is empty).*

$$\sigma_{\emptyset^?(e')}(\Pi_{[composer,V_1,V_2,e']}(Score)).$$

## 4 User query language

Since the model is essentially based on a relational approach, a limited extension of SQL suffices to obtain a user-friendly syntax for our algebraic expressions. Basically, the extension consist of **let** clauses to defined variables as expressions from $\text{ALG}_{(TS)}$, and a **construct** clause, corresponding to $\Pi$, and extending the standard **select**. For the sake of simplicity, we illustrate the language with a few examples on our sample schemas. The first query simply retrieves the songs from Gabriel Fauré (composer), along with their *ambitus* (minimal and maximal notes in the `voice` time series).

**from** Score
**construct** id, voice, min(voice), max(voice)
**where** composer='Fauré'

*min*() and *max*() are examples of functions associated to a domain (modern query languages, e.g., XQuery, provide this facility). The language is designed to allow the introduction of user-defined functions so as to adapt to specific needs. In the musical domain, functions could for instance be added to infer some information from one (or several) voices: tonality, genre, mode, whenever such concepts are relevant. Functions can be either aggregation functions (mapping a time series to a value of the domain, e.g., *min*() and *max*()) or scalar functions (mapping scalar values to values).

The latter can be used to construct new time series, as shown by the following example which creates a TS showing how the energy consumption varies with respect to the average value of year 2009. This example involves the definition of a variable `average` which stores the result of a selection on times series (the **restrict-with** clause).

| | |
|---|---|
| **from** | Energy |
| **let** | $average := avg(**restrict** consumption **with** *year(time())* = 2009) |
| **let** | $relativeEvol := **map**(consumption, *minus()*, $average) |
| **construct** | id, $relativeEvol |

Clauses **from** and **construct** (the latter being equivalent to the usual **select**) are inverted with respect to regular SQL. This permits the introduction of complex expressions with a **let** clause that may refer to the base relations declared in the **from**, or to other variables. The **map** clause 'applies' an operator to each item of a time series (in the spirit of the *map* functional programming mechanism).

Here is now a join: we find the pairs of vehicles which are in a given region (represented by variable $a) at the same time. We assume a Boolean user function *PiP* (for "Point In Polygon"), and proceed as follows. Given an auto cross product on relation Routes, a Boolean time series $r1InRegion is created, with values $\top$ (true) when the vehicle is in $a, else $\bot$. A similar time series $r2InRegion is created for the second vehicle of the pair. A third **map** operation computes the logical intersection of $r1InRegion and $r2InRegion: this defines another time series bound to variable $r1Withr2. The query is satisfied if the latter is not empty.

| | |
|---|---|
| **from** | Routes r1, Routes r2 |
| **let** | $r1InRegion := **map**(r1.traj, *PiR()*, $a) |
| **let** | $r2InRegion := **map**(r2.traj, *PiR()*, $a) |
| **let** | $r1Withr2 := **map**($r1InRegion, *and()*, $r2InRegion) |
| **construct** | r1.name, r2.name |
| **where** | **not**(**Empty**($r1Withr2)) |
| **and** | r1.id != r2.id |

We conclude with an example of the derivation operator. Assume for instance that we search for pattern $P$ in a voice of the Score relation, such that the Dynamic Time Warping distance (computed with the *dtw*() function) is less than 5. Here is the query expression.

| | |
|---|---|
| **from** | Psalm |
| **let** | $derived := **derive** voice **with** *shift* |
| **let** | $dtwVal := **apply** *dtw()* **to** ($derived, $P$) |
| **construct** | id, voice |
| **where** | min($dtwVal) $\leq$ 5 |

The first **let** clause "derives" the `voice` as a sequence of temporary TS $\texttt{voice}_\alpha$, one for each instant $\alpha$. The definition of $\texttt{voice}_\alpha$ is obtained as a composition of `voice` with a time scaling function, here the predefined *shift* linear function. Essentially, the temporary function is such that $\texttt{voice}_\alpha(t) = \mathit{voice}(t + \alpha)$, i.e., the function is 'shifted' to the left. It remains to apply the DTW operator to each occurrence of a $\texttt{voice}_\alpha$, thanks to the **apply** clause.

One obtains a new voice, named `$dtwVal`, which is itself a time series giving the DTW distance at each instant. The **where** clause gets rid of those tuples for which the DTW is never less than 5. Note that one could exploit the `$dtwVal` to access precisely to those instants at which the DTW values match a given condition.

## 5 Related work

Managing time-based data has always been a challenging issue for the database community. The literature holds a lot of proposals for modelization, querying, access structures and storage, in particular through temporal databases. So many different temporal data models and query languages, divided in abstract (e.g. Temporal Relational Calculus [3], First Order Temporal Logic [10]) and concrete (e.g. TSQL2 [23], TempSQL [11]) languages [6], have been proposed that it is impossible to describe them in detail in this paper. See [24, 25, 5, 12] for a list of references and surveys.

One has to distinguish version-based and time series data. The difference is that the value of version-based objects may change in an arbitrary and unpredictable frequency (e.g. the salary of an employee) while the change of values of time series data is tightly associated with a particular pattern of time (e.g. musical score or electricity consumption per hour) [14]. Temporal databases are well suited to version-based objects management. Time series are not just a special case of temporal data because managing this data asks for the recording of historical events associated with discrete points in time and applying particular statistical methods to these events [8]. Thus, temporal database management systems cannot easily manage time series (see [9, 19] for details). Data models specially designed for time series have been proposed in [7, 21, 22, 7, 20] and [8]. Another related work is [15] where authors defined an algebra and a query language (an extension of SQL), for ordered relational data manipulation.

In [8], authors define a model based on an object-oriented approach and develop a specialized time series management systems (TSMS) based on this model. Manipulation of data represented as an array in order to be able to perform some operations of matrix algebra (classical statistical operations as reshaping an array, transposing an array, ...) pays a central role in this model. [18] proposes a language for manipulating arrays. [20] defines a variable vector based data model and develops a time series database along the line of an extended relational database management system. [7] and [21] define models very closed to the variable vector representation.

We believe that our model is strictly more expressive than these approaches. In particular, our work brings two uniques features: (i) the two-level representation that tighty associates relational (atomic) data with time series, and our algebra that glue them together, and (ii) our derivation mechanism which locally transforms the time domain and captures standard time series operations (e.g., moving average). In our opinion, they cannot be easily modeled in the known proposals.

## 6 Summary and ongoing work

This paper investigates the modeling issues involved by a high-level and generic management of time series as first-class citizens in a database management system. As is customary in the field, "high-level" means here a data representation and an associated set of operations which do not hinder the design of an efficient physical representation, and allows an independent modification and tuning of the physical storage. "Generic" denotes the abstract design of the data model that should make it irrelevant to application specifics.

By adopting from the beginning an algebraic approach to the management of time series data sets, we directly enable an expressive and stable language that avoids a case-by-case definition of a query language based on the introduction of ad-hoc functions subject to constant evolution. We believe that this constitutes a sound basis for the development of applications that can rely on an expressive and efficient data management. Given the wide range of domains that potentially need to store and manipulate time series, an important aspect of our work is the native ability to introduce domain-dependent functions in query expressions.

We are currently implementing the language as part of the general architecture of a Music Digital Library devoted to music scores management [2]. One of our main mid-term goals is to experiment the association of our query language with optimization and indexing techniques that efficiently retrieve time series from a large data set without having to fully search the repository.

**Acknowledgments**. This work is partially supported by the French ANR NEUMA project, *http://neuma.irpmf-cnrs.fr*.

## References

1. NIST/SEMATECH e-Handbook of Statistical Methods, 2006. http://www.itl.nist.gov/div898/handbook/.
2. L. Abrouk, H. Audéon, N. Cullot, C. Davy-Rigaux, Z. Faget, D. Gross-Amblard, P. Rigaux, A. Tacaille, E. Gavignet, and V. Thion-Goasdoué. The Design and Implementation of NEUMA, a Collaborative Digital Score Library. In *Submitted.*, 2010. Available at *http://neuma.irpmf-cnrs.fr*.
3. F. D. Anger, R. A. Morris, R. V. Rodríguez, and R. A. Mata-Toledo. A temporal relational calculus. In C. J. Barter and M. J. Brooks, editors, *Australian Joint Conference on Artificial Intelligence*, volume 406 of *Lecture Notes in Computer Science*, pages 177–186. Springer, 1988.
4. C. Y. Chan, B. C. Ooi, and A. Zhou, editors. *Proceedings of the ACM SIGMOD International Conference on Management of Data, Beijing, China, June 12-14, 2007*. ACM, 2007.
5. J. Chomicki. Temporal query languages: A survey. In D. M. Gabbay and H. J. Ohlbach, editors, *ICTL*, volume 827 of *Lecture Notes in Computer Science*, pages 506–534. Springer, 1994.
6. J. Chomicki and D. Toman. Abstract versus concrete temporal query languages. In Liu and Özsu [17], pages 1–6.
7. J. Clifford and A. U. Tansel. On an algebra for historical relational databases: Two views. In S. B. Navathe, editor, *SIGMOD Conference*, pages 247–265. ACM Press, 1985.

8. W. Dreyer, A. K. Dittrich, and D. Schmidt. An object-oriented data model for a time series management system. In J. C. French and H. Hinterberger, editors, *SSDBM*, pages 186–195. IEEE Computer Society, 1994.
9. W. Dreyer, A. K. Dittrich, and D. Schmidt. Research perspectives for time series management systems. *SIGMOD Record*, 23(1):10–15, 1994.
10. E. A. Emerson. Temporal and modal logic. pages 995–1072, 1990.
11. S. K. Gadia. A homogeneous relational model and query languages for temporal databases. *ACM Trans. Database Syst.*, 13(4):418–448, 1988.
12. C. S. Jensen and R. T. Snodgrass. Temporal data management. *IEEE Trans. Knowl. Data Eng.*, 11(1):36–44, 1999.
13. E. Keogh. A decade of progress in indexing and mining large time series databases. In *Proc. Intl. Conf. on Very Large Data Bases (VLDB)*, 2006. Tutorial.
14. J. Y. Lee and R. Elmasri. An eer-based conceptual model and query language for time-series data. In T. W. Ling, S. Ram, and M.-L. Lee, editors, *ER*, volume 1507 of *Lecture Notes in Computer Science*, pages 21–34. Springer, 1998.
15. A. Lerner and D. Shasha. AQuery: Query Language for Ordered Data, Optimization Techniques, and Experiments. In *Proc. Intl. Conf. on Very Large Data Bases (VLDB)*, pages 367–378, 2003.
16. A. Lerner, D. Shasha, Z. Wang, X. Zhao, and Y. Zhu. Fast Algorithms for Time Series with applications to Finance, Physics, Music, Biology, and other Suspects. In *Proc. ACM SIGMOD Symp. on the Management of Data*, pages 965–968, 2004.
17. L. Liu and M. T. Özsu, editors. *Encyclopedia of Database Systems*. Springer US, 2009.
18. A. P. Marathe and K. Salem. A language for manipulating arrays. In M. Jarke, M. J. Carey, K. R. Dittrich, F. H. Lochovsky, P. Loucopoulos, and M. A. Jeusfeld, editors, *VLDB*, pages 46–55. Morgan Kaufmann, 1997.
19. D. Schmidt, A. K. Dittrich, W. Dreyer, and R. W. Marti. Time series, a neglected issue in temporal database research? In J. Clifford and A. Tuzhilin, editors, *Temporal Databases*, Workshops in Computing, pages 214–232. Springer, 1995.
20. A. Segev and R. Chandra. A data model for time-series analysis. In N. R. Adam and B. K. Bhargava, editors, *Advanced Database Systems*, volume 759 of *Lecture Notes in Computer Science*, pages 191–212. Springer, 1993.
21. A. Segev and A. Shoshani. Logical modeling of temporal data. In U. Dayal and I. L. Traiger, editors, *SIGMOD Conference*, pages 454–466. ACM Press, 1987.
22. A. Segev and A. Shoshani. The representation of a temporal data model in the relational environment. In M. Rafanelli, J. C. Klensin, and P. Svensson, editors, *SSDBM*, volume 339 of *Lecture Notes in Computer Science*, pages 39–61. Springer, 1988.
23. R. T. Snodgrass. TSQL2. In Liu and Özsu [17], pages 3192–3197.
24. R. B. Stam and R. T. Snodgrass. A bibliography on temporal databases. *IEEE Data Eng. Bull.*, 11(4):53–61, 1988.
25. A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. Snodgrass, editors. *Temporal Databases: Theory, Design and Implementation*. Benjamin/Cummings, 1993.
26. I. Witten, A. Moffat, and T. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan-Kaufmann, 1999.