Formalism for a Multiresolution Time Series Database Model

Aleix Llusà Serra[a,*], Sebastià Vila-Marta[a], Teresa Escobet Canal[a]

[a]*Department of Electronic System Design and Programming*
Universitat Politècnica de Catalunya
Av. Bases de Manresa 61–73, 08242 Manresa, ES-CT

Abstract

We formalise a specialized database management system model for time series using a multiresolution approach. These special purpose database systems store time series lossy compressed in a space-bounded storage. Time series can be stored at multiple resolutions, using distinct attribute aggregations and keeping its temporal attribute managed in a consistent way.

The model exhibits a generic approach that facilitates its customization to suit better the actual application requirements in a given context. The elements the meaning of which depends on a real application are of generic nature.

Furthermore, we consider some specific time series properties that are a challenge in the multiresolution approach. We also describe a reference implementation of the model and introduce a use case based on real data.

Keywords: Time series, database systems, multiresolution, lossy compression, approximate queries

1. Introduction

Data collection processes proliferate due to the emergence of embedded systems and sensor networks, providing opportunities to collect large amounts of data. These data should be analysed and processed by information systems to be useful. A typical processing, for instance, is to detect eventual sensor failures or malfunctions and, if it is possible, to reconstruct the faulty data. The acquired data instances hold a timestamp. Therefore, correctness criteria must include both data values and their timestamps. The sequences of data values collected at specific timestamps are formalised as *time series*.

A time series is a collection of chronological observations. In general, we continuously acquire a time series from phenomena monitoring. On the one hand, we can record observations at regular intervals, such as hourly or daily ones, resulting in equally spaced time data. On the other hand, we can record observations at irregular intervals, such as recording when a pump is open or closed, resulting in unequally spaced time data. Time series data are often voluminous [1, 2], thus efficiently storing and accessing them can be complex. Moreover, this is especially critical when developing small embedded systems with constrained resources (capacity, energy or processing power) [3]. Additionally, unequally spaced time data increases the difficulty of processing.

The literature describes several attempts to build systems devoted to managing and store time series data. These systems are generically known as *Time Series Database Management Systems* (TSMS), [4, 5]. However, as shown below, most of them exhibit some drawbacks when trying to solve the challenging issues of time series in the temporal data domain.

Time series can be stored and managed by *relational*

*Corresponding author
Email addresses: aleix@dipse.upc.edu (Aleix Llusà Serra), sebastia.vila@upc.edu (Sebastià Vila-Marta), teresa.escobet@upc.edu (Teresa Escobet Canal)

database management systems that are usually queried using *Structured Query Language* (SQL). Nonetheless, some authors [4, 6–8] notice that the use of SQL systems as a time series backend suffers from some drawbacks.

NOSQL or NEWSQL products are being developed to increase the performance and flexibility of SQL systems [7–10]. It is natural to consider them to store time series data. Indeed, the continuous acquisition nature of the time series poses an issue when trying to store and analyse all the data [11].

We can apply compression techniques in two distinct styles to face the challenges posed by time series data. First, to get an approximation to the original signal that facilitates to do pattern search analysis or finding similarities[1, 5, 12]. Second, as a compression and aggregation approach that leverages the storage of massive *data streams* [13, 14]. Nonetheless, handling time series like data streams neither considers adequately the time dimension nor computes the evolution of aggregated parameters along the time, which is interesting for monitoring purposes.

RRDtool [15] is a system that stores time series aggregated using different resolutions. These characteristics allow to compact the data and facilitates faster visualisations. In spite of this, because *RRDtool* is a particular application, aggregation operations are limited to network monitoring.

1.1. Contributions

This paper formalises a model for TSMS that stores and manages time series data. This model exhibits several unusual characteristics:

- It organises the data in an aggregated way and it allows to store time series using different time resolutions. We name this feature *multiresolution*. Thus, being multiresolution the most salient characteristic of our model, we call the formalised system *Multiresolution Time Series Database Management System* (MTSMS). The model is designed to satisfy the requirements of bounded storage computers such as sensor systems.

- It is a *lossy storage* solution. Multiresolution allows for a lossy storage solution that selects only the relevant data. In some sense, multiresolution is close to the lossy compression methods used in multimedia applications, that discard meaningless data in favour of size.

- It considers the time sampling irregularities of time series and operates coherently with the time dimension of time series.

- It offers a degree of genericity to cope with the semantic characteristics of the actual data.

  Multiresolution requires aggregating several data instances into a single one. We abstracted this through an *aggregation function* bound to the precise semantics of the actual data. Because of this, aggregation functions are set as an independent object of the main model. Users can define new aggregation methods better suited for particular fields.

  The model also formalises the concept of time series *representation function*. This concept allows users to define different operators considering the behaviour of time series in different contexts. This issue is important to manage the precise semantics of the stored time series.

- It is soundly formalised using set algebra and, particularly, relational algebra.

Our model shares some of its characteristics with other known approaches. The analysis of *RRDtool* [15] inspired the multiresolution. However, we provide a sound formalisation that lacks in [15] and a degree of genericity unavailable in *RRDtool*. Based on this facility, in our model we can define particular time series aggregations like those of

RRDtool. To formalise time series we follow the same approach used to formalise bitemporal data for a relational DBMS. The model favours more recent data over the older one, which is in common with some other methods, like that of Cormode et al. [13].

We remark that the only goal of the model formalised here is to manage time series data. In practical applications, it would be usual to complement this model with a standard database system to handle all the remaining data if needed. For instance, time series metadata such as units of values, sensor localization or classification tags would be stored in a standard DBMS.

1.2. Outline

This manuscript is organised as follows. Section 2 introduces previous work that concerns TSMS and MTSMS. The motivation for multiresolution is set out in Section 3. We describe the model in two steps. First, in Section 4 we formalise a TSMS model devoted to the basic elements and operations of time series. Second, in Section 5 we formalise a MTSMS model that extends the previous one with multiresolution capabilities. In Section 6 we describe an implementation of the integrated TSMS and MTSMS model. Section 7 is devoted to a real data multiresolution database example. Finally, Section 8 offers some conclusions.

2. Previous work

For the sake of completeness here we describe some previous work related to time series storage. We organise this in three subsections. First, we introduce some previous approaches to database management systems for time series. Second, we explain how some authors applied compression techniques to leverage time series storage. Third, we review time series storage systems based on the data streams paradigm.

2.1. Database approaches

According to some authors, TSMS should be considered as a specialised relational DBMS [5]. Segev and Shoshani [16] propose a structured language for querying TSMS. Their time series structures include the notion of regularity and temporal representation and their operations are SQL-like. Dreyer et al. [4] suggest the requirements of a particular purpose TSMS and base the model on five basic structural elements: events, time series, groups, metadata and time series basis. They implement a TSMS named *Calanda* which includes calendar operations, it allows grouping of time series, and it operates with simple queries. They exemplify it using financial data. In [6] *Calanda* is compared with temporal systems designed for time series.

Other authors consider array database systems well suited to TSMS. *SciDB* [7] and *SciQL* [8] are array database systems intended for science applications, in which time series play a principal role. They structure time series into arrays to achieve multidimensional analysis and they store other data into tables. *SciDB* is based on arrays which, according to the authors, allow to represent time series. In contrast, *SciQL* defines time series as a mixture of array, set, and sequence properties and exhibits some managing characteristics for time series that include dealing with regularities, interpolation or correlation queries.

Bitemporal DBMS, sometimes referred directly as *temporal data*, is a database field that inherently considers temporal dimension of data. Bitemporal data manages historical data and events in databases by associating pairs of *valid* and *transaction* time intervals to data. Bitemporal data and time series data are not exactly the same, and so they cannot be treated interchangeably [6], however, there are some similarities that can be considered. DBMS research represents bitemporal data as relations extended with time intervals attributes and enlarge relational operations to deal with time related aspects [17, 18].

2.2. Compression approaches

Oetiker's *RRDtool* [15, 19] is a free software database management system designed for monitoring systems. Because of this, it is focused on a particular kind of data, gauges and counters, and it lacks general time series operations. *RRDtool* can store data at diverse time resolutions. Plonka et al. [20] evaluated *RRDtool* performance and found limitations when storing a vast number of different time series. They suggested a caching system on top of *RRDtool* as a solution. Weigel et al. [21] advocate for a similar approach that caches queries by aggregate parameters. In Weigel's paper, the authors state that other systems only show subsets of data, but they also consider necessary to show data in their complete time span. They developed the software package known as *TSDS* that fully stores time series and then query them by date ranges or by applying different filters and operations to the data.

Deri et al. [22] suggested *Tsdb*, a lossless compression storage TSMS for time series that share the same time instants of acquisition. Different series are stored grouped by the acquisition time instead of in an isolated way. Deri et al. compared *Tsdb*, *RRDtool*, and a relational product. They found that as a consequence of its structure, *Tsdb* achieves a better measure store time but a worse measure retrieval time than other products. The need to be continuously regrouping data is the cause of the differences. However, when the measures share the same time, *Tsdb* considers them as the same time series and measure retrieval time improves. In these circumstances, it would be interesting to use the *Tsdb* implementation architecture of shared time arrays in a MTSMS to achieve a better storage performance.

Some of the lossy compression techniques for time series pursuit an optimal approximation representation. They seek to balance between the least amount of data that can reconstruct the original signal and the data that gives the least error. Keogh et al. [12] cite some possible approximation representations for time series such as Fourier transforms, wavelets, symbolic mappings or piecewise linear representation. They remark the last one as very usual due to its simplicity and develop a system called *iSAX* [2, 23] to analyze and index massive collections of time series. They argue that the main problem is the indexing of time series, and they propose some efficient methods. The first method proposed is based on a constant piecewise approximation. The time series representation obtained with *iSAX* allows to reduce the stored space and faster indexing while maintaining the quality achieved by more sophisticated methods. These compression techniques are candidates for being used as attribute aggregate functions in the MTSMS model.

2.3. Data stream approaches

Cougar [14] is a sensor database system that maintains two main structures: a relational structure for sensor properties and a set of data sequences for time series coming from sensors. *Cougar* time series have specific operations that can combine relations and sequences. *Cougar* target field is sensor networks, where data are stored distributed in different locations. Queries in *Cougar* are resolved by combining sensor data in a data stream abstraction. According to the authors, this improves the processing performance.

To compute statistical aggregates, some authors consider time series as data streams. Cormode et al. [13] developed some aggregation techniques that give more weight to recent data and that allow to run fast approximate queries on compressed data.

Dou et al. [24] create index structures as multiresolution aggregates, like average, count, or top, for historical data managed in a flash storage. They consider a specific storage solution based on a register with pointers similar to the multiresolution storage in *RRDtool* [19].

3. Multiresolution motivation

An important characteristic of the model formalised in this paper is *multiresolution*. In a previous work, we analysed the requirements, and we summarised the main target for multiresolution systems [25].

In this section, we motivate the advantages of the multiresolution approach. First, we intuitively introduce the concept of multiresolution through an example. Then, we discuss the benefits of this formulation.

Figure 1 shows an example of the multiresolution approach. In the upper part, there is a representation of a signal being monitored. The signal values range from 0 to 10 along the time. There are two specific time instants marked in the figure:

1. The first, marked with the word *init*. It refers to the instant when the database started to receive signal samples.
2. The second, marked with the word *now*. It refers to the instant when the process made the snapshot.

Note that time coordinates are assumed to be positive for the time instants after *init* and negative otherwise. The time before *now* corresponds to the past and the time after *now* to the future. The data before *init* are unknown to the database.

At the bottom of Figure 1 there is a diagram that shows how multiresolution works. The first row displays the signal's sample values that correspond to the above plot. The sampling frequency is of one unit of time. The second and the third rows show an actual schema of a multiresolution database. It consists of two summaries for the time series resolutions. Every summary has a distinct resolution. The first of the rows computes the mean of the sampled values every three units. The second computes the mean every five units. In this example, the aggregation function corresponds to a statistical mean. The values before *init* are not acquired, and they are marked as unknown (u). Future values are also marked as unknown u until time advances.

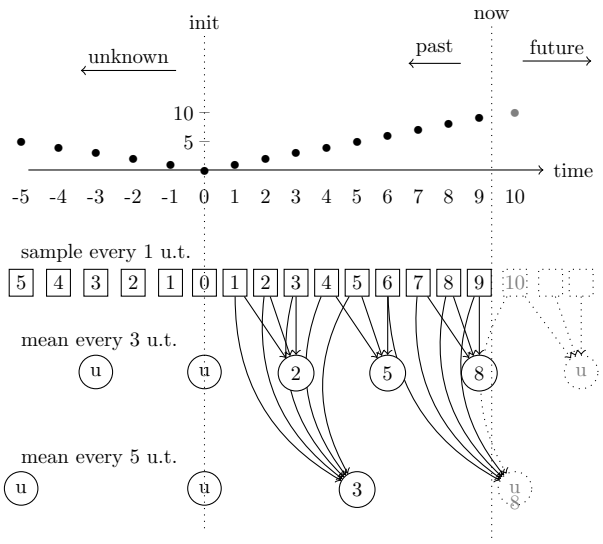

Figure 1: Multiresolution snapshot diagram with regular sampling

Therefore, the first summary for the original signal at time instant *now* corresponds to values $\{u, u, 2, 5, 8, u\}$.

The multiresolution approach enhances TSMS features in several aspects:

- Voluminous data. The monitoring systems capture an enormous amount of data from sensors. To be able to process these data, their volume must be reduced. The multiresolution approach allows to select and store only the most interesting segments of data. We understand these segments as different resolution views for the same time series. The user can configure how these segments are extracted and summarised by defining different time steps and functions. Multiresolution also facilitates the graphing of huge time series. It allows to select the best time range and time step that makes the graph fit on the screen. Because we cannot appreciate more data on the screen, there is no need to render it.

- Data validation. The use of monitoring systems to capture data is a current practice. However, the nature of these systems has some drawbacks that have an impact on the obtained data. Quevedo et al. [26] note that the main problems arise when the moni-

toring system cannot capture data, producing errors known as gaps, or when the monitoring system captures data erroneously. The multiresolution attribute functions are designed to cope well with validation, filtering and reconstruction of these unknown data to keep a consistent history.

- Data time regularisation. To monitor with a non-constant sampling rate has a side effect that induces irregularities in data. According to Kopetz [27] there are two main reasons for sampling rate variation: either sampling jitters in periodic sampling or non-periodic rates caused by event-based sampling. Multiresolution regularises the time interval while processing a time series. As a consequence, every obtained time series segment has a regular time resolution. This feature can also be used to query the time series using some other resolution. For instance, a daily acquired time series can be queried using a yearly step.

- Data summaries. A goal of a database system is to answer the user queries about the stored information. The multiresolution approach allows a lossy compression storage solution. In some sense, it is an online way to compute and store data summaries, i.e. data of interest. These stored data summaries allow faster queries of voluminous data. However, we should determine the organization of the summaries a priori, and we should consider the context where the future queries will be issued.

4. Time series model

Following the traditional database models, a TSMS model consists of two components: a data model and a set of operations. *Measures* and *time series* are the main objects of our TSMS model. In this section, we describe and formalise the TSMS model.

4.1. Data model

Roughly speaking a *time series* is a set of observations collected at specific time instants. An observation may consist of a single value or multiple values collected at the same time instant. We refer a pair of time and observed values as a *measure*. Then, a time series is a correspondence between times and values. Additionally, time series are also described as a set of measures.

We name *time domain* the set $\mathcal{T}$ of all the possible time values. $\mathcal{T}$ can be either a finite or an infinite set and usually it is a closed set. Although time is a complex issue [28], in this paper we will assume that $\mathcal{T}$ is the set of affinely extended real numbers $\bar{\mathbb{R}} = \mathbb{R} \cup \{+\infty, -\infty\}$. This assumption avoids the complex details of time modelling while being powerful enough for our purposes. Next, we define the main time-related concepts using this naive approximation.

Definition 1 (Time concepts). Let $\mathcal{T} = \bar{\mathbb{R}}$ be the domain of time. We name an element $t \in \mathcal{T}$ as *time instant*. Let $s, t \in \mathcal{T}$ be two time instants. We define the *duration of time* between s and t as the value $d \in \mathcal{T}$ which measures the distance in time units between the two time instants, that is $d = |s - t|$.

The *value* is an attribute that indicates the magnitude of a measure. The domain for the values can be of any data type. Valid domains for values include integers, real numbers, strings, and richer data structures such as arrays, lists, or even other time series. In the sequel, the domain for values will be denoted by $\mathcal{V}$. Without loss of generality, in this paper we will assume that the domain of values is the set of projectively extended reals $\mathbb{R}^* = \mathbb{R} \cup \{\infty\}$.

A measure represents an actual value measured at a particular time instant. We define it below.

Definition 2 (Measure). Let $v \in \mathcal{V}$ be a value and let $t \in \mathcal{T}$ be the time instant when the value was acquired. We define a *measure* m as the tuple $m = (t, v)$. The domain

of a measure m, written as $\operatorname{dom} m$, is the domain of its value.

Let $m = (t, v)$ be a measure. In what follows, $V(m)$ denotes the value v and $T(m)$ denotes the time t.

Order between measures plays a significant role. Given two measures we define two distinct *order relations*.

Definition 3 (Semitemporal order). Let m and n be two measures. We name *semitemporal order* the binary relation written $m \leq n$, defined as $m \leq n \iff (T(m) < T(n) \vee (T(m) = T(n) \wedge V(m) = V(n)))$.

Definition 4 (Temporal order). Let m and n be two measures. We name *temporal order* the binary relation written $m \leq^t n$ and defined as $m \leq^t n \iff T(m) \leq T(n)$.

Note that the semitemporal order is a partial order and the temporal order is a total order.

Intuitively speaking, a *time series* is an ordered set of measures of the same phenomena. Sometimes they are also called *time sequences* [29]. We define it as follows.

Definition 5 (Time series). Let $S = \{m_0, \ldots, m_k\} \subset \mathcal{T} \times \mathcal{V}$ be a finite set of measures of the same type. Then, S is a *time series* iff $\forall i, j : i, j \in [0, k] \wedge i \neq j : T(m_i) \neq T(m_j)$. We define the domain of a time series S as the domain of its measures, denoted by $\operatorname{dom} S$.

Observe that although measures in S are expected to be of the same phenomenon, from a formal standpoint we only require the domain of all values to be the same.

A time series does not contain two measures at the same time. Therefore, taking account of the temporal order, a time series is a totally ordered set.

The *cardinality* of a time series $S = \{m_0, \ldots, m_k\}$, noted as $|S|$, is the number of measures that it contains. An *empty time series* is noted as $\emptyset$. Needless to say, $|\emptyset| = 0$.

Although we defined values as scalars, it is easy to extend the concept. Following [30], a time series can record more than one phenomenon if they share the same acquisition time instants. This kind of series is known as *multivalued time series*. Let S be a multivalued time series and let its domain be $\operatorname{dom} S = \mathcal{V}_1 \times \cdots \times \mathcal{V}_n$. Then, we write its measures as $m = (t, v_1, v_2, \ldots, v_n)$.

A time series is *regular* when its measures are evenly spaced in time, according to [29]. Let $S = \{m_0, m_1, \ldots, m_{k-1}, m_k\}$ be a time series, where $T(m_0) < T(m_1) < \cdots < T(m_{k-1}) < T(m_k)$, and let $d \in \mathcal{T}$ be a time duration. Then S is *regular* when $d = T(m_1) - T(m_0) = \cdots = T(m_k) - T(m_{k-1})$.

4.2. Operations

We can manipulate time series using the operations defined in this section. Like the relational model operations, operations over time series ignore the actual semantics of the data. In a real application, it should be decided whether an operation is semantically coherent or not and thus if it should be applied. For example, the addition of values coming from two different phenomena could be semantically wrong.

In this section, we formalise three groups of operations, one in each of the subsections that follow. The set operations, that consider times series as sets; the sequence operations, that consider time series as sequences; and the temporal operations, that manipulate the time series assuming they are representations of functions.

4.2.1. Set operations

In what follows, we describe how to apply common set operators to time series. We rely on how the relational model of DBMS describes operations based on set algebra [31].

Consider a time series S. S is a finite ordered set (by the temporal order). Then, if S nonempty, S has a maximum and a minimum. The *maximum* of S, denoted as $\max S$, is an element of S such that $\forall m \in S : \max S \geq^t m$. Because $\max S$ is not defined when $S = \emptyset$, we are interested in the concept of supremum.

Recall that the time domain is the set of affinely extended real numbers $\bar{\mathbb{R}}$. Then, for an empty subset $T = \emptyset$ of $\bar{\mathbb{R}}$ we know that its supremum is $\sup(T) = -\infty$ [32].

Following the affinely extended reals, we apply the concept of supremum to sets of measures, i.e., time series.

Assume that m is an infinite measure. To be consistent with the affinely extended reals supremum, we consider $T(m) = -\infty$. $V(m)$ could be any arbitrary value. However, we choose an infinite value for simplicity. Then, we define an infinite measure m as $m = (-\infty, \infty)$.

Henceforth, we say that the *supremum* of a time series S, noted as $\sup S$, is the measure defined as follows:

$$\sup S = \begin{cases} \max S & \text{when } S \text{ nonempty} \\ (-\infty, \infty) & \text{otherwise} \end{cases}$$

Dually, we can define the *minimum* of S, noted as $\min S$, and the *infimum* of S, noted as $\inf S$.

The *membership* operation defines when a measure belongs to a time series. We define two distinct membership operations which consider the semitemporal order (Definition 3) and the temporal order (Definition 4). The two distinct membership definitions will induce two different ways to consider time series and its operations.

Let S be a time series and m be a measure. We say that m belongs to S (plain *membership*), denoted as $m \in S$, when $\exists x \in S : x = m$. We also say that m belongs temporally to S (*temporal membership*), denoted as $m \in^{\mathrm{t}} S$, when $\exists x \in S : T(m) = T(x)$.

The two distinct membership criteria induce two meanings for inclusion. Let R and S be two time series. We say that R is *included* in S, written $R \subseteq S$, when all the elements of R belong to S. Analogously, we say that R is *included temporally* in S, noted $R \subseteq^{\mathrm{t}} S$, when all the elements of R belong temporally to S.

The *union* of two sets is a set containing elements from both sets. The traditional set union operations do not apply to time series because the result time series may have repeated time values. Thus, we give a slightly modified concept for the union.

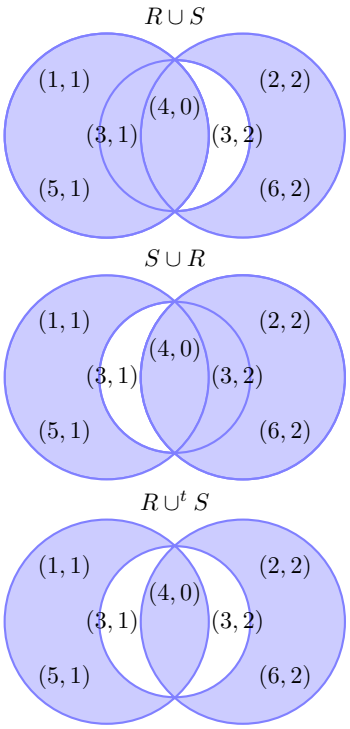

Figure 2: Venn diagrams for set and temporal set union operations of TSMS

The union operation requires both time series to have the same domain, as is also true with the union operation of relational algebra [31].

Let R and S be two time series and let $\operatorname{dom} R = \operatorname{dom} S$. The *union* of R and S, noted $R \cup S$, is a new time series $R \cup S = \{m | m \in R \vee (m \in S \wedge m \notin^{\mathrm{t}} R)\}$. The *temporal union* of R and S, noted $R \cup^{\mathrm{t}} S$, is a time series $R \cup^{\mathrm{t}} S = \{m | (m \in R \wedge m \in S) \vee (m \in R \wedge m \notin^{\mathrm{t}} S) \vee (m \in S \wedge m \notin^{\mathrm{t}} R)\}$. It is interesting to emphasise that the union is a non-commutative operation while the temporal union is a commutative one.

Example 1. Let $R = \{(1,1), (3,1), (4,0), (5,1)\}$ and $S = \{(2,2), (3,2), (4,0), (6,2)\}$ be two time series. The union of R and S is $R \cup S = \{(1,1), (2,2), (3,1), (4,0), (5,1), (6,2)\}$. Because union is not symmetric, $S \cup R = \{(1,1), (2,2), (3,2), (4,0), (5,1), (6,2)\}$. The temporal union results in $R \cup^{\mathrm{t}} S = S \cup^{\mathrm{t}} R = \{(1,1), (2,2), (4,0), (5,1), (6,2)\}$. Figure 2 shows Venn diagrams for all three cases, where the

coloured area depicts the result time series. In every diagram, the central intersection area contains measures that share both time and value attributes, as it is measure $(4, 0)$. The left central area contains the measures in R that only share the time attribute with a measure in S, as it is measure $(3, 1)$. The right central area has a symmetrical meaning. The left and right outer areas are the remaining measures of R and S respectively.

Time series *difference* can also be defined. Similar to the union, the difference requires both time series to have the same domain. Let R and S be two time series and let $\operatorname{dom} R = \operatorname{dom} S$. The *difference* between R and S, written $R - S$, is a time series $R - S = \{m | m \in R \wedge m \notin S\}$. The *temporal difference* between R and S, denoted $R -^t S$, is a time series $R -^t S = \{m | m \in R \wedge m \notin^t S\}$.

Based on union and difference we can define *intersection* as $R \cap S = R - (R - S)$ and *symmetric difference* as $R \ominus S = (R - S) \cup (S - R)$. We can also define the corresponding temporal operations.

Relational DBMS extend the set operators by some more, such as selection, rename or join. This kind of operators also makes sense for time series. To illustrate this possibility we define the join operator.

Roughly speaking, the join of two time series is the combination of measures sharing the same time attribute. Let R and S be two time series. The *join* of R and S, denoted $R \bowtie S$, is a multivalued time series $R \bowtie S = \{(t, v_1, v_2) | (t, v_1) \in R \wedge (t, v_2) \in S\}$. Note that $\operatorname{dom}(R \bowtie S) = \operatorname{dom} R \times \operatorname{dom} S$. Noted that join requires both time series measures to share exactly the same times. When time series diverge, the temporal function operations explained later can be applied to adjust the time instants to join requirements.

A DBMS requires computational operators to enable calculations between pieces of data. Relational DBMS supply operators such as extend, aggregate or summarise [31]. For time series, we define the more general computational operators *map* and *fold*.

The map operator transforms a time series S into a new time series R by applying a function to every measure. Let S and R be two time series, let $\mathcal{V} = \operatorname{dom} \mathcal{S}$ and $\mathcal{V}' = \operatorname{dom} \mathcal{R}$, and let $f : \mathcal{T} \times \mathcal{V} \to \mathcal{T} \times \mathcal{V}'$ be a function over a measure returning a measure. The *map* of f over S is a new time series defined as $\operatorname{map}(S, f) = \{f(m) | m \in S\}$. Note that $\operatorname{dom}(\operatorname{map}(S, f)) = \mathcal{V}'$. In order to be as generic as possible, we do not impose any restrictions on defining f. As a consequence, when f operates with time attributes the result of the map operation may be invalid as a time series. However, when f only operates with value attributes the result is always valid. Therefore, the user must assure the proper definition for f.

The fold operator recursively combines every measure of a time series. Assuming that $\mathcal{P}(C)$ is the powerset of C, we define fold as follows. Let $S = \{m_0, \ldots, m_k\}$ and R be two time series, let $\mathcal{V} = \operatorname{dom} S$, let $\mathcal{V}' = \operatorname{dom} R$ and let $f : \mathcal{P}(\mathcal{T} \times \mathcal{V}') \times (\mathcal{T} \times \mathcal{V}) \to \mathcal{P}(\mathcal{T} \times \mathcal{V}')$ be a function over a time series and a measure, which returns a time series. The *fold* of S by f with initial value R is a new time series defined as $\operatorname{fold}(S, R, f) = f(\cdots(f(f(f(R, m_0), m_1), m_2)\cdots), m_k)$.

The classical aggregation operator combines the data of a time series into a single value. It is worth noting that it is a special case of fold.

Let $S = \{m_0, \ldots, m_k\}$ be a time series, let $\mathcal{V} = \operatorname{dom} S$, let m be a measure with $\operatorname{dom} m = \mathcal{V}$, and let $f : (\mathcal{T} \times \mathcal{V}) \times (\mathcal{T} \times \mathcal{V}) \to \mathcal{T} \times \mathcal{V}$ be a function over two measures returning a measure. The *aggregate* of S by f with initial value m is a new time series defined as $\operatorname{aggregate}(S, m, f) = f(\cdots(f(f(f(m, m_0), m_1), m_2)\cdots), m_k)$.

Example 2. Let $S = \{(1, 1), (2, 3), (4, 1)\}$ be a time series. Map operator allows computing a new time series whose values result from time attribute plus a duration of time. We define the map function $f(t, v) = (t, t+5)$. Then $\operatorname{map}(S, f) = \{(6, 1), (7, 3), (9, 1)\}$. As we will use it later, we name this operation *translation* of a time series.

The fold operator allows, for instance, to select the mea-

sures having its value equal to one. We define the fold function $f(R, m) = R \cup R'$ where $R' = \{m\}$ if $V(m) = 1$ or $R' = \emptyset$ otherwise. Let m be any measure, note that $f(\emptyset, m) = R'$. Then $\text{fold}(S, \emptyset, f) = \{(1, 1), (4, 1)\}$.

The aggregate operator allows, for example, to compute the measure that results from the sum of all the values. To illustrate it, we define the aggregate function $f(m, n) = (0, V(m) + V(n))$. Now, $\text{aggregate}(S, (0, 0), f) = (0, 5)$, where 5 is the sum of all the values of S. Note that time is meaningless in this computation.

Finally, we describe how using the operators defined before, we can implement *binary computational* operators between two time series. This implementation illustrates the power of the operators defined so far.

The strategy requires first to join the two time series and then apply the computational operations. Let S and R be two time series and $\odot$ be a binary operator on the value domain. We can extend the operator $\odot$ to the time series as $S \odot R = \text{map}(S \bowtie R, f)$ being f the function $f(t, v, w) = (t, v \odot w)$. The extended operator allows to extend real binary operations such as sum, $R + S$, to time series.

Example 3. Let $R = \{(1, 2), (2, 2), (4, 0)\}$ and $S = \{(1, 1), (2, 2), (4, 1)\}$ be two time series. The sum of R and S is defined as $R + S = \text{map}(S \bowtie R, f)$ being f the function $f(t, v, w) = (t, v + w)$. Then the sum results $R + S = \{(1, 3), (2, 4), (4, 1)\}$.

As binary computational operators depend on join operator, it must be recalled that join requires both time series measures to share exactly the same times. As aforementioned, when time series diverge, the temporal function operations explained later can be applied to adjust the time instants to join requirements.

4.2.2. Sequence operations

Sequence operations manipulate time series considering measures as being totally ordered by time. We define three basic operations: *slice*, *successor* and *concatenation*.

The classical interval concept can be applied to the time domain. In this context, given two time instants s and t, we use the notation $[s : t]$, $(s : t)$, $[s : t)$ and $(s : t]$ respectively for the closed interval, open interval, right open and left open interval. Following [29], to slice a time series S means to extract a new time series $R \subseteq S$ constrained to a given time interval. We denote this operation as the original time series followed by the interval, therefore, $S[s : t] = \{m | m \in S \wedge T(m) \in [s : t]\}$. We can use other intervals to slice a time series in the same fashion, for instance $S(s : t] = \{m | m \in S \wedge T(m) \in (s : t]\}$.

The ordinary time order allows to define the concepts of successor and predecessor for the measures of a time series. Let $S = \{m_0, \ldots, m_k\}$ be a time series and m be an arbitrary measure. We say that $m_i = \text{next}_S(m)$ is the measure *next* to m in S if and only if $m_i = \inf(S(T(m) : +\infty])$. We also say that $m_i = \text{prev}_S(m)$ is the measure *previous* to m in S if and only if $m_i = \sup(S[-\infty : T(m)))$. Infinite measures are obtained when next and previous are applied to supremum and infimum measures respectively: $\text{next}_S(\sup S) = (+\infty, \infty)$ and $\text{prev}_S(\inf S) = (-\infty, \infty)$.

To concatenate two time series means to compute a new time series with the measures of the first time series followed in time order by the measures of the second one. The concatenation requires both time series to share the same domain. Let R and S be two time series and let $\text{dom}\, R = \text{dom}\, S$. The *concatenation* of R and S, denoted as $R||S$, is a time series that contains all the measures of R together with those of S that do not intersect with the time interval of R. That is, $R||S = R \cup (S - S[T(\inf R) : T(\sup R)])$.

4.2.3. Temporal function operations

We can think a time series as a discrete representation of an original temporal function. In this section, we devise some operations that manage time series according to this point of view. Then, these temporal function operations

may return new measures that are not contained in the time series.

Let $S \subset \mathcal{T} \times \mathcal{V}$ be a time series. A *temporal representation function* for S is a function f defined as $f : \mathcal{T} \to \mathcal{V}$.

Let S be a time series. Assume that we want to obtain f, a temporal representation function for S. It is easy to see that there is not an unique temporal representation function for S. We denote as $S(t)^r$ the temporal representation function of S obtained by applying the method r.

Below, we exemplify the concept of representation function using two different methods based on impulse and constant piecewise functions.

Definition 6 (Dirac representation). Dirac delta (DD) is a method of representation based on the Dirac delta function. Let S be a time series. We define $S(t)^{\text{DD}}$ as the following DD representation function:

$$S(t)^{\text{DD}} = \begin{cases} V(m) & \text{if } \exists m \in S : t = T(m) \\ 0 & \text{otherwise} \end{cases}$$

Definition 7 (Zohe representation). Zero-order hold everted (ZOHE) is a method of representation based on the *zero-order hold* signal reconstruction method. It is a piecewise constant function built from left-continuous step functions. Let S be a time series. We define $S(t)^{\text{ZOHE}}$ as the following representation function:

$$S(t)^{\text{ZOHE}} = \begin{cases} V(m) & \text{if } \exists m \in S : t \in \big(T(\text{prev}_S(m)) : T(m)\big] \\ 0 & \text{if } t > T(\max(S)) \end{cases}$$

The graph of a function allows to obtain and interpret the continuous nature of a time series. When we can plot the domain of time and value attributes, then the graph is equivalent to a graphical representation. Let S be a time series, let r be a representation method and let $\mathcal{T}$ be the time domain. The *graph* of a time series S, denoted as $\text{graph}(S)$, is the set of pairs $\text{graph}(S) = \{(t, S(t)^r) | t \in \mathcal{T}\}$ where $S(t)^r$ is a temporal representation function for the time series S.

We use the concept of representation to formalise some set and sequence operators as temporal operators.

We define a temporal interval operation to introduce this concept. Let S be a time series, let $[s : t]$ be an interval of two time instants and let r be a representation method. The *temporal interval*, denoted as $S[s : t]^r$, returns a new time series with measures in the interval temporal range. That is, $S[s : t]^r = S(u)^r$ for all $u \in [s : t]$. This is a general definition difficult to implement, therefore for every representation a particular temporal interval must be interpreted:

- Let $S(t)^{\text{DD}}$ be the DD representation for S. The DD *temporal interval* is $S[s : t]^{\text{DD}} = S[s : t] \cup \{m\} \cup \{n\}$ where $m = (s, 0)$ and $n = (t, 0)$.
- Let $S(t)^{\text{ZOHE}}$ be the ZOHE representation for S. The ZOHE *temporal interval* is $S[s : t]^{\text{ZOHE}} = S(s : t] \cup \{m\}$ where $m = (t, v)$ and $v = V(\inf(S[t : +\infty]))$.

From temporal interval, we can define other operators such as temporal selection, temporal concatenation, or temporal join. As an example, we give the definition of the temporal selection and the temporal join operations.

The temporal selection over a time series allows to change the resolution in the context of a representation function. Let S be a time series, let $T = \{t_0, t_1, \ldots, t_k\}$ be a set of time instants, and let r be a representation method. The *temporal selection*, denoted as $S[T]^r$, is a time series of measures in T and times computed in coherence with the representation method r. That is, $S[T]^r = S[t_0 : t_0]^r \cup S[t_1 : t_1]^r \cup \cdots \cup S[t_k : t_k]^r$. If t is a time instant, note that the temporal selection depends on the temporal interval operation $S[t : t]^r$, which is equivalent to the notion of temporal representation function over a single time instant. That is, $S[t : t]^r = \{(t, S(t)^r)\}$.

The temporal selection operation also allows to regularise an irregular time series. Let S be a time series, let $d, e \in \mathcal{T}$ be the desired regularity parameters, and let $k \in \mathbb{N}$ be a limit for the scope of the range. A regularised S can

be obtained with $S[T]^r$ where $T = \{e + nd | n \in \mathbb{N} \wedge n \leq k\}$ is a set of evenly spaced time instants.

Another use case for the temporal selection is for joining two time series that do not share the same times. Let R and S be two time series that do not share times, i.e. their time sets are dissimilar $\{t|(t,v) \in R\} \neq \{t|(t,v) \in S\}$. Then, we can not apply the previously defined $R \bowtie S$. However, we can define a more elaborated join based on the temporal selection, which we name *temporal join*. Let r be a representation method and let $T = \{t|(t,v) \in R\} \cup \{t|(t,v) \in S\}$ be the union of both time sets. The *temporal join*, denoted as $R \bowtie^r S$, returns a new time series $R \bowtie^r S = R[T]^r \bowtie S[T]^r$.

Using the temporal join, we can also implement the binary operations for time series that have dissimilar time sets. Next, we redo Example 3 to extend the binary sum using temporal join.

Example 4. Let $R = \{(1,2),(2,2),(4,0)\}$ and $S = \{(1,1),(3,2),(4,1)\}$ be two time series and let $r =$ ZOHE be a representation method. The sum of R and S is defined as $R + S = \text{map}(S \bowtie^r R, f)$ being f the function $f(t,v,w) = (t, v+w)$.

The union of both time sets is $T = \{1,2,3,4\}$. The corresponding temporal selection over both time series is $R[T]^r = \{(1,2),(2,2),(3,0),(4,0)\}$ and $S[T]^r = \{(1,1),(2,2),(3,2),(4,1)\}$. The temporal join of both is $R \bowtie^r S = \{(1,2,1),(2,2,2),(3,0,2),(4,0,1)\}$. Then the sum results $R + S = \{(1,3),(2,4),(3,2),(4,1)\}$.

5. Multiresolution model

In Section 3 we intuitively introduced the concept of multiresolution through an example. In this section, we formalise a model for MTSMS. At the end of the section, we will offer some illustrative examples of the previous formalisation.

A MTSMS is a TSMS that stores time series using a lossy compression approach. The MTSMS model is based on the concepts of *measures* and *time series* as defined in Section 4. A MTSMS stores a time series as a structure that we call *multiresolution time series*. A multiresolution time series is a collection of *resolution subseries* that store a view of the original time series in a given resolution. The operator that adds data to a resolution subseries requires accumulating transient measures in a *buffer*. This procedure allows to aggregate original data to obtain the expected resolution and finally store them in a *disc*.

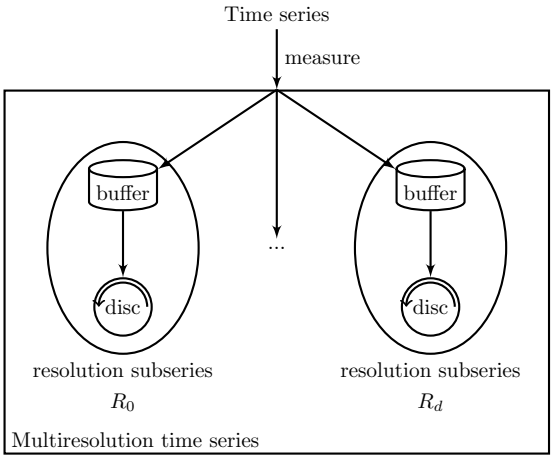

Figure 3: Architecture of MTSMS model

Figure 3 shows the architecture of a MTSMS for a single multiresolution time series. The original time series gets stored in multiple resolution subseries. Each resolution subseries has a particular time resolution and attribute aggregation policy. Discs are size bounded so they only contain a fixed amount of measures. When a disc becomes full, it discards a measure. Thus, a multiresolution database is bounded in size and the time series gets stored in several storage bounded time subseries.

Regarding operations, the MTSMS model requires two kinds of operators. Some operators should be devoted to set up the time intervals between measures and to aggregate the attributes. Some other operators should be dedicated to query the multiresolution schema and to extract the time series data.

Following, we define the MTSMS model structure, its structural operators, the operations to query a multires-

olution time series, and the *attribute aggregate functions*. Although schema manipulation operations could be defined, in this paper we exclusively focus on structure and data query operators.

5.1. Structure

A *buffer* is a container for a time series. The aim of a buffer is to regularise the time series using a constant *resolution step* and an *attribute aggregate function*. We name the regularisation action as *consolidation*. We defined the attribute aggregate functions in Section 5.3.

Definition 8 (Buffer). Let S be a time series, let $\tau \in \mathcal{T}$ be the last consolidation time, let $\delta \in \mathcal{T}$ be the resolution step and let f be an attribute aggregate function. We define a *buffer* B as the tuple $B = (S, \tau, \delta, f)$.

An empty buffer B is defined as $B = (\emptyset, t_0, \delta, f)$, i.e., an empty time series, an initial consolidation time $t_0 \in \mathcal{T}$, a resolution step δ, and a function f. Given a buffer, all the consolidation time instants can be determined as $\tau_n = t_0 + n\delta$ for all $n \in \mathbb{N}$.

A buffer has two main structural operations. The first one adds a measure to the buffer, and the second one consolidates the buffer.

Let $B = (S, \tau, \delta, f)$ be a buffer and let m be a measure. The addition of m to B, noted as $\text{addB}(B, m)$, returns a new buffer $\text{addB}(B, m) = (S', \tau, \delta, f)$ where $S' = S \cup \{m\}$.

Let $B = (S, \tau, \delta, f)$ be a buffer. The consolidation of B, noted as $\text{consB}(B)$, returns a new buffer and a new measure $\text{consB}(B) = (B', m')$ where $B' = (S[\tau + \delta : +\infty], \tau + \delta, \delta, f)$ and $m' = f(S, \tau, \delta)$. This resulting measure summarises the data of S comprised in the given interval. After the consolidation, we can remove from the buffer the consolidated part of the time series. Therefore, we discard historical data.

We apply the consolidation of a buffer to the first non-consolidated time instant. We obtain the total consolidation by successive applications of the operator. Consolidation requires the measures to be added by time order and to consolidate the buffer when the time of some measure is bigger than the buffer's next consolidation time. Let $B = (S, \tau, \delta, f)$ be a buffer, and let $m = \sup S$ be the maximum measure of B. We say that B is consolidable if and only if $T(m) \geq \tau + \delta$.

A *disc* is a finite capacity container of measures. A time series stored in a disc has its cardinal bounded. When the cardinal of the time series is to overcome the limit, some measures need to be discarded.

Definition 9 (Disc). Let $k \in \mathbb{N}$ and S, $|S| \leq k$, be a time series. We define a *disc* D as the tuple $D = (S, k)$.

An empty disc is noted as $(\emptyset, k)$. It is the tuple of an empty time series and a bound k.

The main operation on a disc is to add a measure while keeping under control the cardinal of the times series. Let $D = (S, k)$ be a disc and let m be a measure. The addition of m to D, written as $\text{addD}(D, m)$, is a new disc $\text{addD}(D, m) = (S', k)$ where

$$S' = \begin{cases} S \cup \{m\} & \text{if } |S| < k \\ (S - \{\min S\}) \cup \{m\} & \text{otherwise} \end{cases}$$

A *resolution subseries* is a structure that regularises and aggregates a time series. This structure is composed of a buffer, which contains the time series to be regularised, and a disc, which contains the regularised time series.

Definition 10 (Resolution subseries). Let B be a buffer and let D be a disc. We define a *resolution subseries* R as the tuple $R = (B, D)$.

The operators of a resolution subseries extend the buffer and disc ones. Let $R = (B, D)$ be a resolution subseries, and let m be new a measure. The addition of m to R, noted as $\text{addR}(R, m)$, is a new resolution subseries $\text{addR}(R, m) = (B', D)$ where $B' = \text{addB}(B, m)$ is the addition of the measure to the buffer. The consolidation of R, noted as $\text{consR}(R)$, is a new resolution subseries $\text{consR}(R) = (B', D')$, where $(B', m') = \text{consB}(B)$ is the

consolidation of the buffer, and $D' = \text{addD}(D, m')$ is the addition of the consolidated measure to the disc. A resolution subseries is consolidable only when its buffer is consolidable.

A *multiresolution time series* is a set of resolution subseries referred to the same time series. We store a time series regularised with distinct resolutions across the resolution subseries, as previously shown in Figure 3.

Definition 11 (Multiresolution time series). Let $M = \{R_0, \ldots, R_k\}$ be a finite set of resolution subseries. Then M is a *multiresolution time series.*

Therefore, to define a multiresolution time series we must define the number of resolution subseries and its corresponding parameters (δ, τ, f, k). Usually, there are no repeated pairs of (δ, f) parameters among a multiresolution series, thus they act as key attributes.

The operators of a multiresolution time series apply to every resolution subseries contained. Let $M = \{R_0, \ldots, R_k\}$ be a multiresolution time series and let m be a measure. The addition of a measure to every resolution subseries, noted as $\text{addM}(M, m)$, is a new multiresolution time series $\text{addM}(M, m) = \{R'_0, \ldots, R'_k\}$ where $R'_i = \text{addR}(R_i, m)$. The consolidation of all resolution subseries, noted as $\text{consM}(M)$ is a new multiresolution time series $\text{consM}(M) = \{R'_0, \ldots, R'_k\}$ where

$$R'_i = \begin{cases} \text{consR}(R_i) & \text{if } R_i \text{ consolidable} \\ R_i & \text{otherwise} \end{cases}$$

5.2. Queries

There are two basic time series queries for a MTSMS: a query to extract a time subseries from a resolution subseries, and a query to obtain a total time series from all consolidated data.

Let M be a multiresolution time series and let (δ, f) be a pair of key attributes. The query operator denoted as $\text{SerieDisc}(M, \delta, f)$ computes a time series such that $\exists (B, D) \in M : B = (S, \tau, \delta, f) \wedge D = (\text{SerieDisc}(M, \delta, f), k)$ where S, τ, k are bound variables. We assume that there are no repeated (δ, f) pairs in M.

Let $M = \{R_0, \ldots, R_k\}$ be a multiresolution time series and let $S_0, \ldots, S_k$ be the time series corresponding to the resolution subseries $R_0, \ldots, R_k$. Assume that the attribute aggregation functions of all R_i are the same and the resolution steps of all R_i are distinct. We define the query operator $\text{TotalSeries}(M)$, the time ordered concatenation of all time subseries, as follows. $\text{TotalSeries}(M) = S_{i_0} || S_{i_1} || \cdots || S_{i_k}$ where $i_0, \ldots, i_k$ is a permutation of $[0, k]$ such that $\delta_{i_0} < \delta_{i_1} < \cdots < \delta_{i_k}$, being δ_i the resolution step of the resolution subseries R_i. Recall that $R||S$ is the concatenation of two time series R and S, which we defined in Section 4.2.2.

The operator TotalSeries obtains the better possible resolution.

Using these basic time series queries, we can define more elaborated queries for a MTSMS by using TSMS operations. For example, let L and M be two multiresolution time series with the same multiresolution parameters. We can compute the sum of both as $\text{TotalSeries}(L) + \text{TotalSeries}(M)$. Recall that $R + S$ is the sum of the values of two time series R and S that we defined in Example 3 as a binary computational operator.

When two multiresolution time series do not have the same multiresolution parameters, then $\text{TotalSeries}(L) + \text{TotalSeries}(M)$ must be solved using temporal join, as we showed in Example 4. Nevertheless, dealing with temporal join operations can be cumbersome as they depend on a representation method. In this regard, we can consider some multiresolution enhancements:

- If L and M share a resolution subseries with the same τ and δ parameters, then it is possible to compute $\text{SerieDisc}(L, \delta, f_1) + \text{SerieDisc}(M, \delta, f_2)$ with the basic join. The uninstantiated variables f_1 and f_2 are any attribute aggregate functions, although usually it makes more sense to be the same $f_1 = f_2$.

- If L and M share a resolution subseries with only the same δ, then the previous point can be applied if one time subseries is be translated. Let $R = \text{SerieDisc}(L, \delta, f_1)$ and $S = \text{SerieDisc}(M, \delta, f_2)$ be the time subseries desired and let τ_R and τ_S be the corresponding last consolidation time for each δ. Let $f(t, v) = (t, t + \tau_R - \tau_S)$ be a map function and let $S' = \text{map}(S, f)$ be the result of the translation operation, as we showed in Example 2. Then, we can apply $R + S'$ with the basic join.

- Otherwise, when L and M do not share any resolution subseries with the same δ, the temporal join must be used in the sum $\text{TotalSeries}(L) + \text{TotalSeries}(M)$. However, as L and M are lossy storage solutions for the original time series they represent, $\text{TotalSeries}(L) + \text{TotalSeries}(M)$ computes less data than applying the sum directly to the original time series.

5.3. Attribute aggregate function

Attribute aggregate functions are a particular case of TSMS aggregate operations used to summarise time series data while consolidating a buffer.

Let S be a time series, let δ be a resolution step and let τ be a consolidation time. An *attribute aggregate function* f calculates a new measure $m = f(S, \tau, \delta)$. From τ and δ, we obtain the time interval $[\tau : \tau + \delta]$. Then, the resulting measure m is interpreted to summarise the measures of S for the time interval $[\tau : \tau + \delta]$.

An attribute aggregation function follows this general scheme. First, it obtains a time subseries S' according to the consolidating interval using a slice operator. For example, $S' = S[\tau : \tau+\delta]$. Second, it applies a TSMS aggregation function on this time subseries to obtain m. For instance, $m = \text{aggregate}(S', n, f)$, being f an aggregation function and n an initial measure, as defined in Section 4.2.1.

We can use many different attribute aggregate functions to summarise a time series. For instance, it is possible to calculate a statistical indicator of the time series such as the average or a more complex digital signal processing operation as proposed in [8]. Furthermore, during the aggregation process we can consider the representation of a time series and some of its pathologies.

Given the diversity of attribute aggregate functions, no global assumptions can be made about them. Each user should decide which combination of aggregation and representation fits better to the measured phenomenon. Therefore, the MTSMS model must have a generic design that allows the users to define their aggregate functions.

In what follows we will give some examples of usual attribute aggregation functions. These functions compute a new measure given a set of known measures. Then, an attribute aggregation function should compute a new *time* and a new *value* from the set of known measures.

Usually, an attribute aggregation function returns measures that match the buffer consolidating times. Assume, for instance, that f is an attribute aggregation function and let $m = f(S, \tau, \delta)$. Then, the time of m is usually computed as $T(m) = \tau + \delta$. However, in some cases it is preferable for $T(m)$ not to match the buffer consolidating times. For instance, the resulting measure can be aggregated from a time subseries S' using an open interval $S' = S(\tau : \tau + \delta)$, a closed interval $S' = S[\tau : \tau + \delta]$, or other combinations as it is $S' = S(\tau - d : \tau + \delta - d]$, where d is a time duration that delays the consolidation to $T(m) = \tau + \delta - d$. This time offset can also be variable. For example, consider an aggregate function that returns the first measure of the interval $m = \min(S[\tau : \tau + \delta))$, then the resulting time fulfils that $\tau \leq T(m) < \tau + \delta$.

Assume that f is an attribute aggregation function and let $m = f(S, \tau, \delta)$. An attribute aggregation function f should compute the value of m. Next, there are some examples that illustrate how to compute $V(m)$ based on the temporal function time series operators. That is, the time series aggregated is interpreted by the temporal representation function $S(t)^r$ as has been described in Sec-

tion 4.2.3. In these example functions, we leave the time series representation r uninstantiated.

- The *maximum* computes $V(m)$ as $V(m) = \max_{\forall t \in [\tau:\tau+\delta]} S(t)^r$. It summarises S with the maximum of the measure values in the interval $[\tau : \tau + \delta]$.
- The *last* computes $V(m)$ as $V(m) = S(\tau + \delta)^r$. It summarises S with the value at $\tau + \delta$ time instant.
- The *mean* computes $V(m)$ as $V(m) = \frac{1}{\delta} \int_{\tau}^{\tau+\delta} S(t)^r dt$. It summarises S with the mean of the function in the interval $[\tau : \tau + \delta]$.

We can instantiate the time series representation in the previous examples in several ways. In what follows, we exemplify this by instantiating r as DD and ZOHE.

Dirac delta attribute aggregation functions interpret the resulting time as centered on the interval $T(m) = \frac{2\tau+\delta}{2}$. The resulting value $V(m)$ depends on the attribute. Let $S' = S[\tau : \tau + \delta]^{\text{DD}}$ be the selection of measures by Dirac delta temporal interval. Then,

- The $maximum^{\text{DD}}$ is such that $V(m) = \max\big(0, \max_{\forall n \in S'} V(n)\big)$.
- The $last^{\text{DD}}$ is such that $V(m) = V(\max S')$.
- The $mean^{\text{DD}}$ is such that $V(m) = \frac{1}{\delta} \sum_{\forall n \in S'} V(n)$. Note that for the Dirac delta function $\int \text{DD}(t)dt = 1$. Note that $\sum_{\forall n \in S'} V(n)$ is a sum of values that could be implemented as aggregate$(S', (0, 0), f)$ where $f(m, n) = (0, V(m) + V(n))$, as shown in Example 2.

ZOHE attribute aggregation functions interpret the resulting time as the right limit of the interval $T(m) = \tau + \delta$. The resulting value $V(m)$ depends on the attribute, let $S' = S[\tau : \tau + \delta]^{\text{ZOHE}}$ be the selection of measures by ZOHE temporal interval. Then,

- The maximum$^{\text{ZOHE}}$ is such that $V(m) = \max_{\forall n \in S'} V(n)$.
- The $last^{\text{ZOHE}}$ is such that $V(m) = V(\max S')$.
- The mean$^{\text{ZOHE}}$ is such that $V(m) = \frac{1}{\delta}\big[(T(o)-\tau)V(o) + \sum_{\forall n \in R} (T(n) - T(\text{prev}_S\, n))V(n)\big]$ where $o = \min S'$ and $R = S' - \{o\}$. Note that mean$^{\text{ZOHE}}$ is a sum of values that could be implemented as mean$^{\text{ZOHE}}$ $= \frac{1}{\delta}$ aggregate$(S', (0, 0), f)$ where $f(m, n) = (0, V(m) + v)$ and

$$v = \begin{cases} (T(n) - \tau)V(n) & \text{if } n = \min S' \\ (T(n) - T(\text{prev}_{S'}\, n))V(n) & \text{else} \end{cases}$$

RRDtool, [15], uses an aggregation function similar to mean$^{\text{ZOHE}}$ to summarise velocity counter data by keeping the area below the original signal.

It is interesting to note that some attribute aggregation patterns are very similar. For instance, the maximum and last attribute aggregation schemes differ basically in the interval selection operation. However, other patterns have a more elaborated interpretation depending on the actual representation used, as it is the case of mean$^{\text{ZOHE}}$ and $mean^{\text{DD}}$.

To summarise the model we have formalised in this section, we show a basic multiresolution example.

Example 5. We define a multiresolution schema for a time series, we consolidate the database and we query its data. Let $S = \{(1, 6), (5, 2), (8, 5), (10, 0), (14, 1), (19, 6), (22, 11), (26, 6), (29, 0)\}$ be a time series and let $M = \{R_0, R_1\}$ be a multiresolution time series where each resolution parameters are $\tau_0 = 0$, $\delta_0 = 5$, $f_0 = \text{mean}^{\text{ZOHE}}$, $k_0 = 4$ and $\tau_1 = 0$, $\delta_1 = 10$, $f_1 = \text{maximum}^{\text{ZOHE}}$, $k_1 = 2$. Therefore R_0 will be consolidated at time instants 5, 10, 15, 20, 25, 30... and R_1 at 10, 20, 30...

We add all measures of S to M, and then we consolidate it until it can not be consolidated further. As $T(\max S) = 29$, the last consolidation times are $\tau_0 = 25$ and $\tau_1 = 20$, so let M_{29} be the multiresolution time series at this state.

Then, the two time subseries consolidated are obtained by querying SerieDisc$(M_{29}, 5, \text{mean}^{\text{ZOHE}}) = \{(10, 3), (15, 2), (20, 7), (25, 8)\}$ and SerieDisc$(M_{29}, 10,$

maximum$^{\text{ZOHE}}$) = $\{(10, 6), (20, 11)\}$. Regarding buffers, let S_0 and S_1 be the M_{29} buffer's time series, note that $S_0 = \{(26, 6), (29, 0)\}$ and $S_1 = \{(22, 11), (26, 6), (29, 0)\}$.

In this particular example, TotalSeries(M_{29}) = SerieDisc($M_{29}, 5, \text{mean}^{\text{ZOHE}}$) as R_0 has twice the resolution of R_1 and k_0 is bigger than k_1.

We extend the previous example to show some of the MTSMS enhancements. We exemplify what happens when we add a new measure to a previously consolidated multiresolution time series.

Example 6. Let S, M and M_{29} be the same entities defined in the previous example. Now, consider that we acquire a new measure. Let $m = (31, 4)$ be this new measure, and let $S' = S \cup \{m\}$ be the updated time series.

We add this measure to the previously consolidated multiresolution time series and, as now it is again consolidable, we consolidate it. Then, let $M_{31} = \text{consM}(\text{addM}(M_{29}, m))$ be the multiresolution time series at this new state.

Alternatively, following the same procedure as in Example 5, we can add all measures of S' to M, and then consolidate it as many times as possible. This procedure also results in the multiresolution time series M_{31}. However, this approach does not notice that M_{29} has already been defined.

Whichever approach is taken, we can query the two new consolidated time subseries by applying SerieDisc($M_{31}, 5, \text{mean}^{\text{ZOHE}}$) = $\{(15, 2), (20, 7), (25, 8), (30, 2)\}$ and SerieDisc($M_{31}, 10, \text{maximum}^{\text{ZOHE}}$) = $\{(20, 11), (30, 11)\}$.

This example shows that the new state of the multiresolution time series can be computed online. Following the acquisition stream of the time series, we can add the measure m to the previously computed multiresolution time series M_{29} and then consolidate it to obtain M_{31}. There is no need to store either the original time series S or S'. Furthermore, the successive states of M store compacted summaries of the original data. At any time during the acquisition of the original time series, we can query the consolidated time subseries and visualise them immediately.

6. Reference implementation

In this section, we briefly describe a reference implementation of the models offered in Sections 4 and 5. We give this implementation as a proof of concept. It does not attempt to be either an efficient or a complete database system. We implement the TSMS and MTSMS models using the Python [33] programming language.

The implementations of the two models, TSMS and MTSMS, are organised respectively in two separated Python libraries: `Pytsms` and `RoundRobinson`. `RoundRobinson` strongly depends on `Pytsms` following the dependency of MTSMS on TSMS. The code of this implementation can be found in [34].

The reference implementation follows the object orientated paradigm, and it observes a clear mapping between the model and the object classes. We use *Unified Modeling Language* (UML) diagrams to define the class structures. We realized the model operations as object methods that we do not present in UML diagrams, due to space limitations.

6.1. Pytsms

`Pytsms` is the reference implementation of the model concepts of measure, time series, and temporal representation function. Figure 4 shows the relationships among these objects in a UML diagram. A `TimeSeries` object is an aggregation of `Measure` objects. `TimeSeries` and `Representation` objects are associated, i.e., each `TimeSeries` has a default representation, and a `Representation` operates over a `TimeSeries`.

A `TimeSeries` object has many methods. We classify them based on their functionality. First, a `TimeSeries` object includes methods to manipulate the structural model.

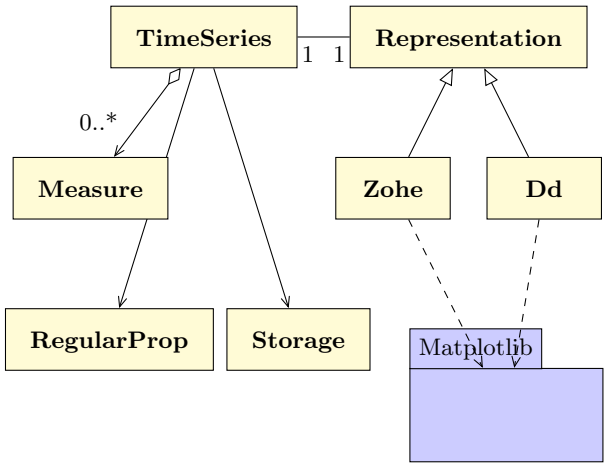

Figure 4: Pytsms UML class diagram

We implemented `TimeSeries` as a subclass of the predefined `set` Python type. Second, a `TimeSeries` object has methods that implement set, sequence, and temporal function operators, as described in Section 4.2. Third, accessory operations of `TimeSeries` are grouped into two *visitor* objects: `RegularProp`, that includes the operations to regularize, and `Storage`, that has the methods for storing and retrieving time series from the file system. Recall that *visitor* is a design pattern that allows to add new functionality to objects without modifying them [35, 36].

Figure 4 displays two specialisations for the `Representation` abstract class that refer to the representation functions given in Section 4.2.3. Each specialisation defines the graph and the temporal interval operations. Besides, a `Representation` has also a method to plot a coherent graphical representation of a time series. For this purpose, we use the Python *Matplotlib* library.

Example 7. In this example we reproduce an interpreter session working with `Pytsms`. First, we define the two time series `r` and `s` from Example 1, and after that we apply to the series some operations: union, temporal union, concatenation, closed interval, ZOHE temporal interval, ZOHE temporal selection, and the test of regular property. Note that we abbreviated `Measure` to `m`. The log of the session is as follows:

```
# Import the required objects
>>> from pytsms import TimeSeries, Measure as m
>>> from pytsms.representation import Zohe
>>> from pytsms.properties import isRegular

# Define the two time series
>>> r = TimeSeries([m(1,1),m(3,1),m(4,0),m(5,1)])
>>> s = TimeSeries([m(2,2),m(3,2),m(4,0),m(6,2)])

# Manipulate the two time series
>>> r.union(s)
TimeSeries([m(1,1), m(2,2), m(3,1),
            m(4,0), m(5,1), m(6,2)])
>>> r.union_temporal(s)
TimeSeries([m(1,1), m(2,2), m(4,0), m(5,1), m(6,2)])
>>> r.concatenate(s)
TimeSeries([m(1,1), m(3,1), m(4,0), m(5,1), m(6,2)])
>>> s.interval_closed(2,5)
TimeSeries([m(2,2), m(3,2), m(4,0)])
>>> s.interval_temporal(2,5,Zohe)
TimeSeries([m(3,2), m(4,0), m(5,2)])

# Check for regularity
>>> s.accept(isRegular())
False
# regularise to {0,2,4} by ZOHE method
>>> sr = s.selection_temporal(range(0,6,2),Zohe)
>>> sr
TimeSeries([m(0,2), m(2,2), m(4,0)])
>>> sr.accept(isRegular())
True
```

6.2. RoundRobinson

`RoundRobinson` is the reference implementation of the multiresolution time series model. It includes objects like resolution subseries, buffers, discs, and attribute aggregate functions. Figure 5 exhibits the relationships among these objects in a UML diagram. A `MultiresolutionSeries` object is an aggregation of `Resolution` objects. A

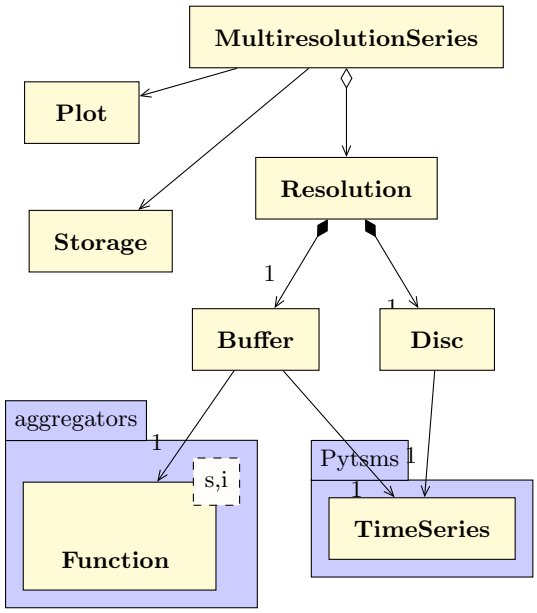

Figure 5: RoundRobinson UML class diagram

`Resolution` object is composed by one `Buffer` object and one `Disc` object. We associate each `Buffer` object to one `TimeSeries` object, from the `Pytsms` library. We also associate each `Disc` object to one `TimeSeries`. Both `TimeSeries` objects correspond respectively to the buffer's and the disc's time series in the MTSMS model. Furthermore, we associate each `Buffer` to one attribute aggregate function. We define an aggregate function as a `Python` function with two parameters: a `TimeSeries` `s` and a consolidation time interval `i`.

`MultiresolutionSeries` is a subclass of the predefined *set* Python class. It adds some operations grouped into two objects: `Storage` and `Plot`. The methods of a `Storage` object are devoted to store and retrieve multiresolution time series from the file system. The methods of a `Plot` object are employed to plot the time subseries of the multiresolution schema.

We apply the method `addResolution` of a `MultiresolutionSeries` object to define the multiresolution schema structure by adding resolution subseries. We configure a new resolution using four parameters: `delta`, `k`, `f`, and `tau`. These parameters allow to create the corresponding buffer and disc. A `MultiresolutionSeries` object relies on the methods `add`, `consolidable`, and `consolidate`, to operate on the corresponding methods of the contained resolution subseries.

We can query a `MultiresolutionSeries` object using one of two methods: `seriedisc` and `totalseries`. The method `seriedisc` returns the `TimeSeries` object that corresponds to the disc identified by the parameters `delta` and `f`. The method `totalseries` returns the `TimeSeries` object that results from the concatenation of all `seriedisc` objects sorted by `delta`.

In the module `aggregators`, we implemented some default attribute aggregate functions. However, users can define new aggregators as well. For instance, we defined the ZOHE aggregate functions explained in Section 5.3, which basically aggregate data over the temporal interval `s.interval_temporal(s,t,Zohe)`. The Example 7 shows this aggregate function.

Example 8. In this example, we use `Pytsms` to set the time series `S`. Also, we employ `RoundRobinson` to define the multiresolution time series M obtained from Example 5. Then, we apply consolidation, and we query the result.

```
# Import the required objects
>>> from pytsms import TimeSeries, Measure as m
>>> from roundrobinson import MultiresolutionSeries
>>> from roundrobinson.aggregators import mean_zohe,
...                                       maximum_zohe

# Define the original time series
>>> s = TimeSeries([m(1,6),m(5,2),m(8,5),m(10,0),
...                 m(14,1),m(19,6),m(22,11),
...                 m(26,6),m(29,0)])

# Define the multiresolution time series
>>> M = MultiresolutionSeries()
# Define the multiresolution schema
>>> M.addResolution(delta=5,k=4,f=mean_zohe,tau=0)
```

```
>>> M.addResolution(delta=10,k=2,f=maximum_zohe,tau=0)

# Add the measures
>>> for m in s: M.add(m)
# M is consolidable
>>> M.consolidable()
True
# Consolide until no more consolidable
>>> while M.consolidable():
...     M.consolidate()

# Query the consolidated discs
>>> M.seriedisc(5,mean_zohe)
TimeSeries([m(10,3), m(15,2), m(20,7), m(25,8)])
>>> M.seriedisc(10,maximum_zohe)
TimeSeries([m(10,6), m(20,11)])
# Query the total time series
>>> M.totalseries()
TimeSeries([m(10,3), m(15,2), m(20,7), m(25,8)])
```

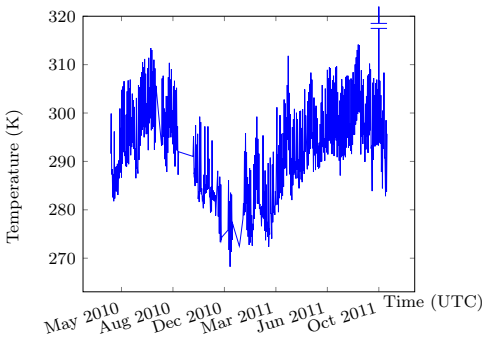

Figure 6: Example of a temperature time series data

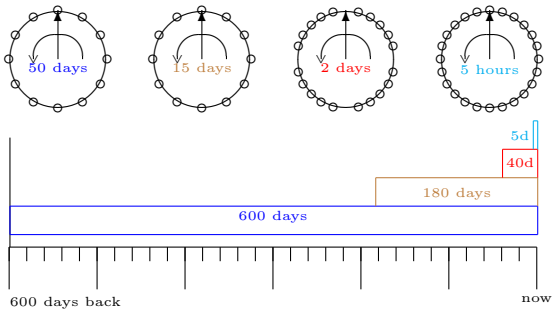

Figure 7: Schema of multiresolution

7. Case study

In this section, we introduce a real case study. Actual data come from a temperature distributed sensor monitoring system [37]. We focus on the data of a particular sensor. We use `Pytsms` and `RoundRobinson` implementations to create a MTSDB and to query it.

Data. Figure 6 shows the original data for one year and a half. The plot interpolates the measures linearly. In this plot, we can see that there are missing data and some outlying observations. There are 146 709 stored values.

Schema. We design a MTSDB schema that stores a multiresolution time series with higher resolution at recent times and with lower resolution at older times. Figure 7 illustrates this schema. At the top, there are four different discs and, at the bottom, there is a timeline showing the resolution subseries along the time. The configuration of the discs is as follows: (i) a measure every 5 h in the fourth disc, which has a capacity of 24 measures and thus it spans 5 days; (ii) a measure every 2 days in the third disc, with a capacity of 20 measures and thus spanning 40 days; (iii) a measure every 15 days in the second disc, with a capacity of 12 thus spanning 180 days and; (iv) a measure every 50 days in the first disc that, with a capacity of 12 measures results in a span of 600 days. The last span is longer than the original time series, then at least one resolution keeps some data about the complete original time interval.

Attribute aggregate functions. To illustrate this example we consolidate all the resolution subseries using the mean $^{\text{ZOHE}}$ aggregate function, and the two highest resolution subseries using the maximum $^{\text{ZOHE}}$ aggregate function.

Consolidation. Figure 8 shows the time subseries after consolidating the MTSDB. Each graphic corresponds to the possible SerieDisc queries, i.e., every resolution disc time series from the MTSDB. Each title shows the resolution subseries and its cardinal, and each attribute aggregate function has a different colour. Time series are plotted using ZOHE representation function $S(t)^{\text{ZOHE}}$. The time axis has UTC units rounded to nearest time points, and temperature axis has Kelvin units. In the plot, we mark outliers

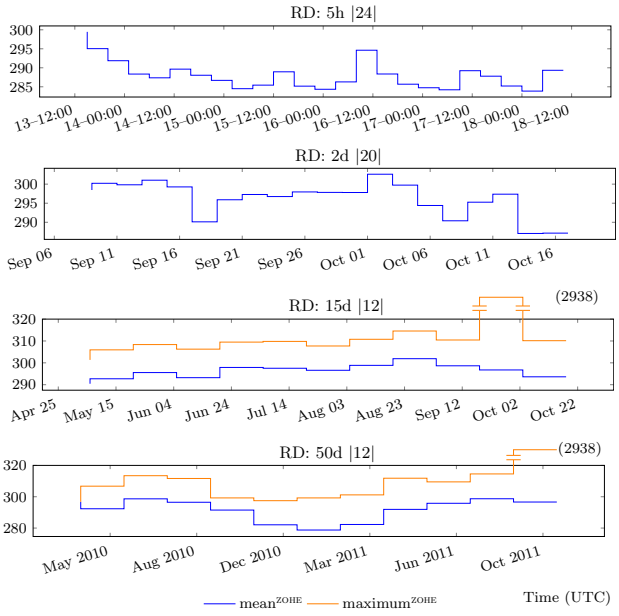

Figure 8: Resolution subseries in the MTSDB

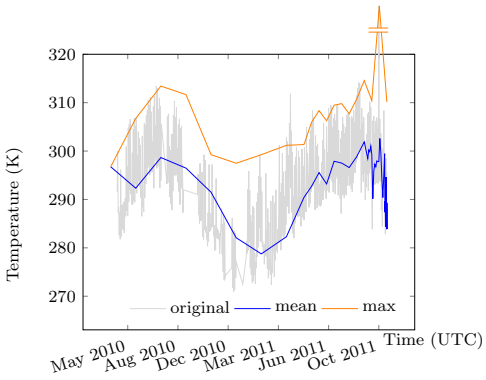

Figure 9: TotalSeries for the mean$^{\text{ZOHE}}$ and maximum$^{\text{ZOHE}}$ resolutions

as discontinuities. The fourth plot's 2938 K maximum illustrates this.

In all the four plots, we can see that the mean aggregate function has filled missing data and filtered outlier observations because the aggregate function comes from a ZOHE interpretation. In the 50 days step resolution, the first data point consolidated is previous to the original time series. However, it is consolidated with the first known data as its aggregation comes from ZOHE interpretation.

Figure 9 shows the TotalSeries queries for the mean $^{\text{ZOHE}}$ aggregate function resolution and for the maximum $^{\text{ZOHE}}$ resolution. Each resulting time series is plotted by interpolating its measures linearly. Note that this linear rendering seems displaced to the right because of ZOHE aggregation. Comparing this figure with the original series in Figure 6, we observe that it resembles an incremental low-pass filter, since we applied mean aggregation. Also, the maximum aggregation resembles an envelope function.

In conclusion, this MTSDB schema does not store the complete original data, but a compression of the original function that contains more data for recent times. Each of the SerieDisc time series is regular with δ. Although TotalSeries is not a regular time series, it has piece-wise regularity as a concatenation of every disc's δ. The purpose of this example is to show how we compute the multiresolution for a time series. In this case, we acquired the bulk data previously, and thus we calculated multiresolution offline. However, a MTSMS is designed to consolidate while the original data are being acquired so that the multiresolution computation spreads along the acquisition and the computing time becomes less critical.

8. Conclusions

In this paper, we formalised a MTSMS model. The foundation of MTSMS is the TSMS model, which we also formalised. We structured our models based on set theory, and heavily inspired on relational algebra. We have gone a bit further, and we proposed a TSMS model that includes set, sequence, and temporal function behaviour. We also motivated the interest of multiresolution and its advantages. As a reference implementation, we developed a *Python* package focused on the basic algebra, i.e., without the extended DBMS capabilities.

The main purpose of a MTSMS is to store compactly a *time series* and to operate its temporal dimension consistently. It stores time series using *multiresolution time series*, that is, it stores a time series at multiple resolutions called *resolution subseries*. Any resolution subseries has two key features: its resolution step and its *attribute*

aggregate function, that is used to compact the data. According to this structure, a multiresolution time series is configured with a few parameters. These parameters are the number of resolution subseries, and for every subseries: the resolution step, the initial consolidation time, the attribute aggregate function, and the capacity. If we tune these parameters properly, a multiresolution database can keep the desired data from a time series.

We have shown some aggregation functions examples with simple aggregation statistics, mean and maximum, and simple *representation methods*, DD and ZOHE. More attribute aggregation functions could be designed based on methods from other fields, such as data streaming or time series data mining. Especially, it would be interesting to design aggregations that coped with uncertain data. The model allows users to customise a multiresolution database according to the actual requirements of a given context.

The queries over MTSMS obtain time series from stored multiresolution time series. In this way TSMS operators can be applied if needed. The SerieDisc time series being *regular* facilitates these operations. However, the lossy storage implies that some operations will give approximate queries and that not every TSMS operation will be semantically correct for a multiresolution time series. Therefore, the correct planning of the multiresolution schema is mandatory.

Compared to other TSMS, we introduce a compression solution that stores only the data that can be required in later queries. We do not intend to reconstruct the original signal. Our multiresolution solution copes well with some difficult aspects of time series: regularity, data validation and data volume. The decompression time is minimal as data in discs get stored directly as a time series. As a consequence, computing a query has a small time cost. Moreover, if the query is an aggregation or resolution already calculated in a MTSMS consolidation, then the response is immediate.

We formalised our model using set algebra without specifying any particular query language. Because of this, our model is independent of specific implementations or query languages. In future work, a particular query language might be defined that facilitates the comparison of our multiresolution solution with other approaches. As our model is heavily inspired on relational algebra, we may implement this query language using academically relational query languages, such as Tutorial D [31]. An additional benefit of this procedure will be to illustrate whether the multiresolution time series use cases are cumbersome when we use relational languages.

When we apply a MTSMS to store time series data, we discard some data given its lossy nature. In future work, it would be interesting to apply information theory to measure the information lost depending on the configuration of the multiresolution schema. We think that it is possible to get inspired by the approaches used by some authors on multimedia lossy compression techniques. For example, we could evaluate whether a human can distinguish some specific features in a time series visualisation based on the original time series and based on some multiresolution time series. Alternatively, given a query, we could compare the difference between the results obtained by applying the query to a multiresolution time series or applying the query to the original time series.

Acknowledgements

We would like to thank the anonymous reviewers.

The research presented in this paper has been supported by Spanish research projects TEC2012-35571 and DPI2014-58104-R, and Universitat Politècnica de Catalunya predoctoral grant.

[1] T.-C. Fu, A review on time series data mining, Engineering Applications of Artificial Intelligence 24 (1) (2011) 164–181. doi:10.1016/j.engappai.2010.09.007.

[2] J. Shieh, E. Keogh, iSAX: Indexing and mining terabyte sized time series, in: Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD'08, ACM, Las Vegas, 2008, pp. 623–631.

[3] Y. Yao, J. Gehrke, The Cougar approach to in-network query processing in sensor networks, SIGMOD Record 31 (3) (2002) 9–18. doi:10.1145/601858.601861.

[4] W. Dreyer, A. K. Dittrich, D. Schmidt, Research perspectives for time series management systems, SIGMOD Record 23 (1) (1994) 10–15. doi:10.1145/181550.181553.

[5] M. Last, Y. Klein, A. Kandel, A. K, Knowledge discovery in time series databases, IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics 31 (1) (2001) 160 –169. doi:10.1109/3477.907576.

[6] D. Schmidt, A. K. Dittrich, W. Dreyer, R. W. Marti, Time series, a neglected issue in temporal database research?, in: Proc. of the International Workshop on Temporal Databases: Recent Advances in Temporal Databases, Workshops in Computing, Springer, Zürich, Switzerland, 1995, pp. 214–232.

[7] M. Stonebraker, J. Becla, D. J. DeWitt, K.-T. Lim, D. Maier, O. Ratzesberger, S. B. Zdonik, Requirements for science data bases and SciDB, in: Proc. of the Fourth Biennial Conference on Innovative Data Systems Research, CIDR'09, CIDR, 2009.
URL http://www.cidrdb.org

[8] Y. Zhang, M. Kersten, M. Ivanova, N. Nes, SciQL: bridging the gap between science and relational dbms, in: Proceedings of the 15th Symposium on International Database Engineering & Applications, IDEAS'11, ACM, 2011, pp. 124–133. doi:10.1145/2076623.2076639.

[9] P. Atzeni, C. S. Jensen, G. Orsi, S. Ram, L. Tanca, R. Torlone, The relational model is dead, SQl is dead, and i don't feel so good myself, SIGMOD Record 42 (1) (2013) 64–68. doi:10.1145/2503792.2503808.

[10] M. Stonebraker, SQL databases v. NoSQL databases, Communications of the ACM 53 (4) (2010) 10–11. doi:10.1145/1721654.1721659.

[11] E. Keogh, P. Smyth, A probabilistic approach to fast pattern matching in time series databases, in: Proceedings of the 3rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD'97, ACM, NewPort Beach, California, 1997, pp. 24–20.

[12] E. Keogh, K. Chakrabarti, M. Pazzani, S. Mehrotra, Locally adaptive dimensionality reduction for indexing large time series databases, in: Proceedings of ACM SIGMOD International Conference on Management of Data, SIGMOD'01, ACM, Santa Barbara, California, USA, 2001, pp. 151–162.

[13] G. Cormode, F. Korn, S. Tirthapura, Time-decaying aggregates in out-of-order streams, in: Proceedings of the 27th ACM Symposium on Principles of Database Systems, PODS'08, ACM, 2008, pp. 89–98. doi:10.1145/1376916.1376930.

[14] P. Bonnet, J. Gehrke, P. Seshadri, Towards sensor database systems, in: Proceedings of the Second International Conference on Mobile Data Management, MDM'01, Springer-Verlag, Hong Kong, 2001, pp. 3–14. doi:10.1007/3-540-44498-X_1.

[15] T. Oetiker, RRDtool, round robin database (1998–2013).
URL http://oss.oetiker.ch/rrdtool

[16] A. Segev, A. Shoshani, Logical modeling of temporal data, in: Proceedings of the Association for Computing Machinery Special Interest Group on Management of Data, SIGMOD'87, ACM Press, 1987, pp. 454–466. doi:10.1145/38713.38760.

[17] C. S. Jensen, R. T. Snodgrass, Temporal data management, IEEE Transactions on Knowledge and Data Engineering 11 (1) (1999) 36–44. doi:10.1109/69.755613.

[18] C. J. Date, H. Darwen, N. A. Lorentzos, Temporal Data and the Relational Model, Morgan Kaufmann, San Francisco, US-CA, 2002. doi:10.1016/B978-155860855-9/50047-7.

[19] T. Oetiker, MRTG the multi router traffic grapher, in: Proceedings of the 12th Systems Administration Conference, LISA'98, USENIX Association, Boston, Massachusetts, 1998, pp. 141–148.

[20] D. Plonka, A. Gupta, D. Carder, Application buffer-cache management for performance: Running the world's largest MRTG, in: Proceedings of the 21st Systems Administration Conference, LISA'07, USENIX Association, Dallas, Texas, 2007, pp. 63–78.

[21] R. Weigel, D. Lindholm, A. Wilson, J. Faden, TSDS: high-performance merge, subset, and filter software for time series-like data, Earth Science Informatics 3 (1) (2010) 29–40. doi:10.1007/s12145-010-0059-y.

[22] L. Deri, S. Mainardi, F. Fusco, Tsdb: A compressed database for time series, in: Proceedings of the 4th International Conference on Traffic Monitoring and Analysis, TMA'12, Springer-Verlag, 2012, pp. 143–156. doi:10.1007/978-3-642-28534-9_16.

[23] A. Camerra, T. Palpanas, J. Shieh, E. J. Keogh, iSAX 2.0: Indexing and mining one billion time series, in: Proceedings of the 10th IEEE International Conference on Data Mining, ICDM'10, IEEE, Sydney, Australia, 2010, pp. 58–67.

[24] A. Dou, S. Lin, V. Kalogeraki, D. Gunopulos, Supporting historic queries in sensor networks with flash storage, Inf. Syst. 39 (2014) 217–232. doi:10.1016/j.is.2012.04.002.

[25] A. Llusà Serra, T. Escobet Canal, S. Vila Marta, A model for a multiresolution time series database system, in: Proceedings of the 12th International Conference on Artificial Intelligence, Knowledge Engineering and Data Bases, AIKED'13, WSEAS Press, Cambridge, UK, 2013, pp. 55–60.
URL http://www.wseas.us/e-library/conferences/2013/CambridgeUK/AISE/AISE-08.pdf

[26] J. Quevedo, V. Puig, G. Cembrano, J. Blanch, J. Aguilar, D. S. ang G. Benito, Validation and reconstruction of flow meter data in the barcelona water distribution network, Control Engineer-

ing Practice 18 (6) (2010) 640–651.

[27] H. Kopetz, Real-Time Systems, 2nd Edition, Real-Time Systems Series, Springer, New York, US, 2011. doi:10.1007/978-1-4419-8237-7.

[28] B. Dowden, Time supplement — the internet encyclopedia of philosophy (IEP) (2010) [cited 2013-12-10].
URL http://www.iep.utm.edu/time-sup/

[29] M. L. Hetland, A survey of recent methods for efficient retrieval of similar time sequences, in: M. Last, A. Kandel, H. Bunke (Eds.), Data mining in time series databases, no. 57 in Series in Machine Perception and Artificial Intelligence, World Scientific, Singapore, 2004, Ch. 2, pp. 23–41.

[30] J. Aßfalg, Advanced analysis on temporal data, Ph.D. thesis, Fakultät für Mathematik, Informatik und Statistik der Ludwig Maximilians Universität München (2008).
URL http://edoc.ub.uni-muenchen.de/8798

[31] C. J. Date, An Introduction to Database Systems, 7th Edition, Addisson-Wesley, Boston, US-MA, 2000.

[32] D. W. Cantrell, Affinely extended real numbers (2012) [cited 2014-02-06].
URL http://mathworld.wolfram.com/AffinelyExtendedRealNumbers.html

[33] Python Software Foundation, The Python standard library – Python 2.7.7 documentation [cited 2014-06-13].
URL http://docs.python.org/2/library/

[34] A. Llusà Serra, RoundRobinson and Pytsms code v0.3 (2012–2014) [cited 2014-07-20].
URL http://escriny.epsem.upc.edu/svn/rrb/src/roundrobinson/tags/0.3/

[35] T. Ziadé, Expert Python Programming, Packt Publishing, Birmingham, UK, 2008.

[36] R. C. Martin, Agile Software Development: Principles, Patterns, and Practices, Prentice Hall, 2002.

[37] C. Alippi, R. Camplani, C. Galperti, A. Marullo, M. Roveri, An hybrid wireless-wired monitoring system for real-time rock collapse forecasting, in: 7th International Conference on Mobile Adhoc and Sensor Systems, MASS'10, IEEE, 2010, pp. 224 – 231. doi:10.1109/MASS.2010.5663999.