Interval OLAP: Analyzing Interval Data

Christian Koncilia[1], Tadeusz Morzy[2,⋆], Robert Wrembel[2,⋆], and Johann Eder[1]

[1] Klagenfurt University, Institute of Informatics Systems, Klagenfurt, Austria
[2] Poznan University of Technology, Institute of Computing Science, Poznań, Poland
{koncilia,eder}@isys.uni-klu.ac.at,
{Tadeusz.Morzy,Robert.Wrembel}@cs.put.poznan.pl

Abstract. The ability to analyze data organized as sequences of events or intervals became important by nowadays applications since such data became ubiquitous. In this paper we propose a formal model and briefly discuss a prototypical implementation for processing interval data in an OLAP style. The fundamental constructs of the formal model include: events, intervals, sequences of intervals, dimensions, dimension hierarchies, a dimension members, and an iCube. The model supports: (1) defining multiple sets of intervals over sequential data, (2) defining measures computed from both, events and intervals, and (3) analyzing the measures in the context set up by dimensions.

1 Introduction

It is observed that current applications in use generate huge sets of data. Some of the data have the character of events that last an instant, whereas some of them last for a given time period - an interval. Events typically have a strict order, thus possess a sequential nature. Sequential data can be categorized either as *time point-based* or *interval-based* [13].

Some examples of systems that generate this kind of data include: workflow systems, Web logs, RFID, public transport, and sensor networks. In workflow systems objects arrive to ordered tasks at certain points in time and they are processed there during a certain period of time. By analyzing workflow log data one is able to discover bottlenecks and idle time. In Web log analysis, especially for e-commerce, one may be interested in knowing the navigation path leading to a product purchase. RFID technology is becoming widely used in supply chain management (e.g., just-in-time delivery). Here, by analyzing sequences of events generated by the RFID devices one is able to optimize product transportation routes. In advanced public transportation infrastructures, cf. [12] passenger tracking records are automatically generated by various devices. These records can be used for analyzing the most frequently used routes and, thus, for discovering route bottlenecks, station bottlenecks, and rush hours in various districts. In intelligent installations (e.g., ambient living, jet engines, refineries), numerous

⋆ Work supported from the Polish National Science Center (NCN), grant No. 2011/01/B/ST6/05169.

L. Bellatreche and M.K. Mohania (Eds.): DaWaK 2014, LNCS 8646, pp. 233–244, 2014.

sensors supply their data. Based on the chronologically analyzed data, one can discover patterns of behavior or predict device breaks.

There is a substantial demand for models and tools for analyzing sequential data. Most of the existing OLAP techniques, although very advanced ones, allow to analyze mostly set oriented data without exploiting the existing order among the data. For this reason, it was necessary to create new models and techniques that would be able to store and analyze sequential data efficiently.

Paper Contribution In this paper, we contribute a formal and implementation model, called *I-OLAP*, for an OLAP system that enables the user to define and analyze intervals stemming from sequential data. In particular, we present an extension of the S-OLAP concept [3] to achieve the following features: (1) enable the user to easily define multiple *sets of intervals* over sequential data, (2) define *measures* computed from both, events and intervals, (3) analyze these measures easily along multiple *dimensions*.

Analyzing sequences and intervals could also be done with standard SQL queries. However, we will show that this leads to huge query statements which therefore are nearly unreadable and most notably not maintainable. Therefore, we prototypically implemented a query language which enables the user to analyze sequential data and interval based data.

Paper Organization: This paper is organized as follows. In section 2 we will discuss related work. Section 3 presents our running example and define the set of example queries. Section 4 presents the I-OLAP data model. Section 5 shows how to query interval data, based on our data model. In section 6 we will briefly discuss the implementation of our approach. Section 7 summarizes the paper, outlines open issues and research directions for the future.

2 Related Work

The model which will be presented in this paper is - to the best of our knowledge - the first OLAP model focusing on how to analyze interval data. However, our model is building on different approaches which focus on how to analyze sequential data. These approaches will be briefly discussed in this section.

[3] propose a formal model for time point-based sequential data with the definitions of a fact, measure, dimension, and a dimension hierarchy. Thus, the model allows to analyze sequential data in an OLAP style. However, neither a query language nor a prototype system was built on the model.

In the *S-OLAP* approach [12] propose the set of operators for a query language for the purpose of analyzing patterns, whereas [4,5] concentrate on an algorithm for supporting ranking pattern-based aggregate queries and on a graphical user interface. The drawback of this approach is that it is based on relational data model and storage for sequential data.

Stream Cube [9] and *E-Cube* [11,10] implement OLAP on data streams. Their main focus is on providing tools for OLAP analysis within a given time window of constantly arriving streams of data.

[6,7] address interval-based sequential data, generated by RFID devices. In [6] the authors focus on reducing the size of such data. They propose techniques

for constructing RFID cuboids and computing higher level cuboids from lower level ones. Based on this foundation, [7] propose a language for analyzing paths with aggregate measures, generated by RFID devices.

[17,16] focus on mining sequential patterns on interval-based data applying a class of Apriori and PrefixSpan algorithms.

From the commercial systems only *Oracle* [15] and *Teradata Aster* [2] support SQL-like analysis of sequential data in their OLAP engines but they focus on pattern recognition in time-point-bases sequential data.

To the best of our knowledge, the aforementioned contributions do not support the analysis of interval data. With this respect, there is an evident need for developing a model and a query language capable of discovering and analyzing such data in an OLAP style.

3 Running Example

As a running example, we will use sample data acquired by sensors installed in an intelligent building. Let us assume that: (1) the sensors report the status of lights and temperatures in some rooms, and (2) our data warehouse stores events that report changes, i.e. if a light sensor reports a sequence of events $< \{room1, t1, on\}$, $\{room1, t2, on\}$, $\{room1, t3, on\}$, $\{room1, t4, off\} >$, the second and third event will not be stored. Table 1 depicts the data received from the light and heating sensors in two rooms (room_id 100 and 101). Obviously, the light sensors return boolean values (on, off) whereas the heating sensors report the temperature as float values once per hour. Heating sensor H_1 reports a failure at 2013.03.20 14:08:13. This problem has been fixed 3 hours later.

Table 1. Example light and temperature sensor data

room_id	sensor_id	time	value
100	L_1	2013.03.20 10:08:12	on
100	H_1	2013.03.20 10:08:13	19.2
100	H_1	2013.03.20 11:08:13	20.0
100	H_1	2013.03.20 12:08:13	21.2
100	L_1	2013.03.20 12:24:12	off
100	H_1	2013.03.20 13:08:13	18.0
100	L_1	2013.03.20 13:09:12	on
100	H_1	2013.03.20 14:08:13	failure
100	H_1	2013.03.20 17:08:13	21.2
100	L_1	2013.03.20 17:38:12	off
101	H_2	2013.03.20 9:18:13	19.0
101	L_2	2013.03.20 9:19:12	on
101	H_2	2013.03.20 10:18:13	21.5
101	H_2	2013.03.20 11:08:13	21.6
101	L_2	2013.03.20 19:40:12	off

Typical examples of OLAP queries on this kind of interval data could include: (1) find the floor (sum of all rooms on a floor) where light was on for the longest time per day, (2) find all rooms where light was off and heating was on, i.e. the temperature increased, (3) report the average heating costs per room and day, (4) report the five rooms with the longest/shortest period of time light was on, (5) report the largest number of state changes per day per a light sensor.

4 I-OLAP Data Model

In this section we propose a metamodel and a formal model of interval OLAP (I-OLAP). The elements of the *I-OLAP* metamodel are shown in Figure 1. It consists of *events and its attributes*, *dimensions, hierarchies, and dimension members*, *intervals*, *sequences of intervals*, and *iCubes.*

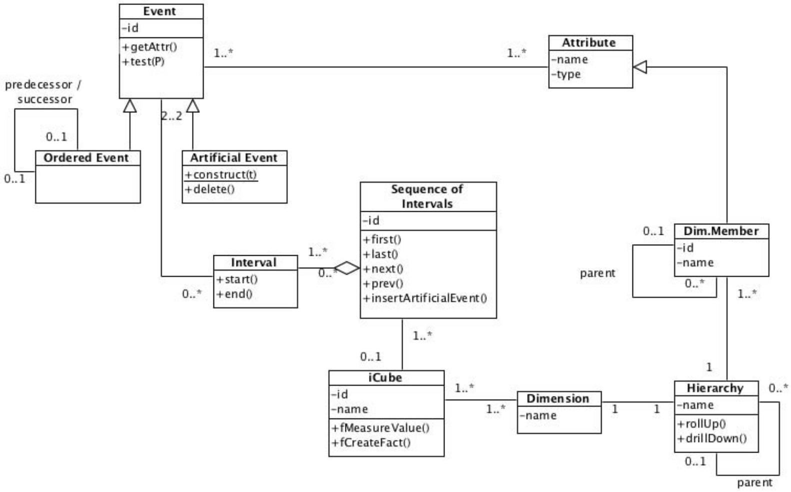


Fig. 1. The I-OLAP metamodel

4.1 Events and Attributes

Event $e_j = (a_{1j}, a_{2j}, \ldots, a_{nj})$, where: a_{ij} is the value of attribute A_i of the j-th elementary event and $a_{ij} \in Dom(A_i)$. $\mathbb{A} = \{A_1, A_2, \ldots, A_n\}$ is the set of attributes of the elementary event, and $Dom(A_i)$ is the domain of the ith attribute (including atomic values plus null). The set of all events $\mathbb{E} = \{e_1, e_2, \ldots, e_m\}$.

Intuitively, we can say that an event is simply a tuple in the original transactional dataset. In our running example the first record could be mapped to event e_1 with assigned values *room_id* = 100, *sensor_id* = L_1, *time* = *2013.03.20 10:08:12*, and *value* = *on*. An example set of events is given in Table 2.

In the model we distinguish two specializations of the event, namely: artificial, and consecutive. The artificial event exists temporarily to answer a given query, cf. Section 5. Consecutive events are used to represent intervals, cf. Section 4.2.

4.2 Intervals

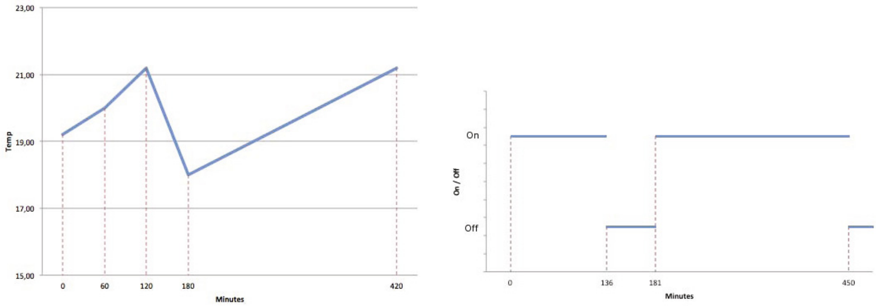
Intuitively, intervals correspond to the 'gap' between any two consecutive events. Figures 2(a) and 2(b) depict intervals for 'Heating' and 'Light'. The intervals are defined over attribute *value*, i.e. the current temperature and light status.

Table 2. Events in our running example

Event ID	Event data
e_1	`100, L1, 2013.03.20 10:08:12, on`
e_2	`100, H1, 2013.03.20 10:08:13, 19.2`
e_3	`100, H1, 2013.03.20 11:08:13, 20.0`
e_4	`100, H1, 2013.03.20 12:08:13, 21.2`
e_5	`100, L1, 2013.03.20 12:24:12, off`
e_6	`100, H1, 2013.03.20 13:08:13, 18.0`
e_7	`100, L1, 2013.03.20 13:09:12, on`
e_8	`100, H1, 2013.03.20 14:08:13, failure`
e_9	`100, H1, 2013.03.20 17:08:13, 21.2`
e_10	`100, L1, 2013.03.20 17:38:12, off`

Two consecutive events form an interval. Events e_1 and e_2 are consecutive if: (1) both of them belong to intervals that belong to the same sequence of intervals and (2) there exists no other event between both events, i.e. $\nexists e_i \in S : e_1.t \leq e_i.t \leq e_2.t$.

Interval $I =< e_n, e_m >$, where $e_n \in \mathbb{E} \wedge e_m \in \mathbb{E}$, e_n and e_m are consecutive events. The set of all defined intervals is denoted as $\mathbb{I} = \{I_1, I_2, ..., I_n\}$.



(a) Temperature intervals (b) Light intervals

Fig. 2. Temperature and Light Intervals

Two basic methods are defined on intervals, namely: (1) `start()` – returns the start event of the interval, (2) `end()` – returns the end event of the interval.

4.3 Sequences of Intervals

Multiple intervals form the sequence of intervals. The order within the sequence is defined by an ordering attribute(s) assigned to all events of all intervals. Hence, within the sequence of intervals all events must have the same ordering attribute(s). Furthermore, an event may be the part of several, different intervals as long as these intervals do not belong to the same sequence of intervals. For example, the user might create three separate sequences of intervals, i.e., about heating, light, and both heating and light.

Sequence of intervals $S =< I_1, I_2, ..., I_n >$, where $I_i \in \mathbb{I}$. The set of all sequences of intervals is denoted as $\mathbb{S} = \{S_1, S_2, ..., S_n\}$.

While defining the **methods on intervals** we were inspired by [3]. The methods include: (1) `first()`, `last()`, `next()`, `prev()` – they allow to iterate over the intervals within a sequence of intervals, (2) `insertArtificialEvent()` – it creates a new, artificial event (cf. Section 5).

4.4 Dimensions, Hierarchies, and Dimension Members

A **dimension** is derived from one attribute assigned to events. Each dimension may have a concept hierarchy associated with it. In order to support galaxy schemas, our model supports 'shared dimensions', i.e. dimensions that may be assigned to multiple cubes. Our running example could for instance have dimensions 'Location', 'Time', and 'Event Type'.

Every dimension consists of one **hierarchy** that represents the root hierarchical element of a cube. This element may consist of multiple sub-elements that, in turn, may consist of multiple sub-elements, thus building a hierarchy. In our running example, the 'Location' dimension could include hierarchy $Building \rightarrow Floor \rightarrow Room$.

The hierarchy assigned to a dimension defines the navigation path a user may use to perform roll-up and drill-down operations, like in the standard OLAP. However, we have to consider that we are aggregating intervals. This problem will be discussed on a general level in Section 5.

Just as hierarchies, **dimension members** are also in a hierarchical order represented by the recursive association of the dimension members. For instance, the 'Location' hierarchy could consist of dimension members $Room100$, $Room102$, etc. Each dimension member is derived from event attributes.

Dimension $D = \{A_D, \mathbb{H}_\mathbb{D}\}$, where A_D is an attribute with $A_D \in \mathbb{A}$ and $\mathbb{H}_\mathbb{D}$ is the set of hierarchical assignments associated with the dimension. Thus, $\mathbb{H}_\mathbb{D} = \{H_1, ..., H_n\}$ with $H_i = \{ID, Name, H.P_{ID}, \mathbb{M}\}$, where ID is a unique identifier, $Name$ is the name of the hierarchy, and $H.P_{ID}$ is the identifier of the parent hierarchy or null if there is no parent.

$\mathbb{M}$ is the set of dimension members assigned to this hierarchy: $\mathbb{M} = \{M_1, ..., M_n\}$, where $M_j = \{ID, Name, M.P_{ID}\}$, where ID is a unique identifier, $Name$ is the name of the dimension member, and $M.P_{ID}$ is the identifier of the parent dimension member or null if there is no parent.

4.5 iCube

iCube is a data cube enabling users to analyze interval-based data. It consists of: (1) entities well known in traditional OLAP, namely dimensions, hierarchies, and dimension members and (2) entities used to analyze interval data, namely sequences of intervals, intervals, events, and attributes.

$iCube = \{\mathbb{S}, \mathbb{D}, \mathbb{F}_{\mathbb{CV}}, \mathbb{F}_{\mathbb{FC}}\}$ where $\mathbb{S}$ is the set of sequences of intervals, $\mathbb{D}$ is the set of dimensions, $\mathbb{F}_{\mathbb{CV}}$ and $\mathbb{F}_{\mathbb{FC}}$ are two different sets of functions. Mandatory set $\mathbb{F}_{\mathbb{CV}}$, called **compute value functions** includes user defined functions for computing fact values (measures). Optional set $\mathbb{F}_{\mathbb{FC}}$, called **fact creating functions** includes user defined functions for creating new measures / facts assigned to an interval.

The compute value functions are used to derive / estimate values from two given consecutive events. For instance, when using the 'Heating' events, the temperatures at $e_1.time$ and $e_2.time$ are defined by $e_1.value$ and $e_2.value$. However, there exists no data for any time point t witch $e_1.time < t < e_2.time$. The user may now define functions to compute values for 'Heating' and 'Light' as shown in Listings 1.1 and 1.2. In these examples, we use simple linear monotonic functions, but any function may be used to compute values.

Listing 1.1. Function computing temperature at a given time point

```
//INPUT: t − timepoint
//OUTPUT: value, in this case the costs
function TempAtT(t) {
 //does t correspond to an event time?
e = fetchEvent where e.time = t
if (e != null) return e.value
 //if it does not match any event, find the
 //corresponding interval for t
i = fetchInterval for Timepoint t
//if there exists no interval, return null
if (i == null) return null
e1 = i.startEvent()
e2 = i.endEvent()
 //assuming that the temperature rises/falls
 //linear and that ut(t) returns the number
 //of seconds of t
return (e2.value−e1.value)/
           (ut(e2.time)−ut(e1.time))∗ut(t)}
```

Listing 1.2. Function computing light status at a given time point

```
//INPUT: t the timepoint
//OUTPUT: a value, in this case the costs
function LightStatusAtT(t) {
 //does t correspond to an event time?
e = fetchEvent where e.time = t
if (e != null) return e.value
 //if it does not match any event, find the
 //corresponding interval for t
i = fetchInterval for Timepoint t
 //if there exists no interval, return null
if (i == null) return null
e1 = i.startEvent()
e2 = i.endEvent()
return e1.value }
```

Now, using functions $LightStatusAtT$ and $TempAtT$, we can compute the light status and temperature at any given point in time. For example, fetching the temperature of room 100 for the time point that corresponds to $t = 170$ can be done by calling $TempAtT(170)$.

Listing 1.3. Example function computing energy cost of an interval

```
//INPUT: e1, e2 − two consecutive events
//OUTPUT: value, in this case costs
function costs(e1, e2) {
  //assuming that light sensors are boolean and return only ON or OFF
  if (e1.value == 'on')
  //assuming that the costs for each minute are 0.02 cents
    return (e2.time − e1.time)∗0.02
  else return 0 }
```

The fact creating functions are used to create facts that do not stem from events, but from sequences or intervals. For instance, in our running example we could assign a user defined operation that for 'Light' computes the costs by multiplying minutes between a 'Light on' and a 'Light off' event with a given cost factor (cf. Listing 1.3). Obviously, this fact cannot be derived from a single event but from sequences or intervals.

The two following methods on iCube are available to create new fact creating functions and compute value functions, namely: (1) `fMeasureValue()` – creates new function $f \in \mathbb{F}_{\mathbb{CV}}$, (2) `fCreateFact()` – creates new function $f \in \mathbb{F}_{\mathbb{FC}}$.

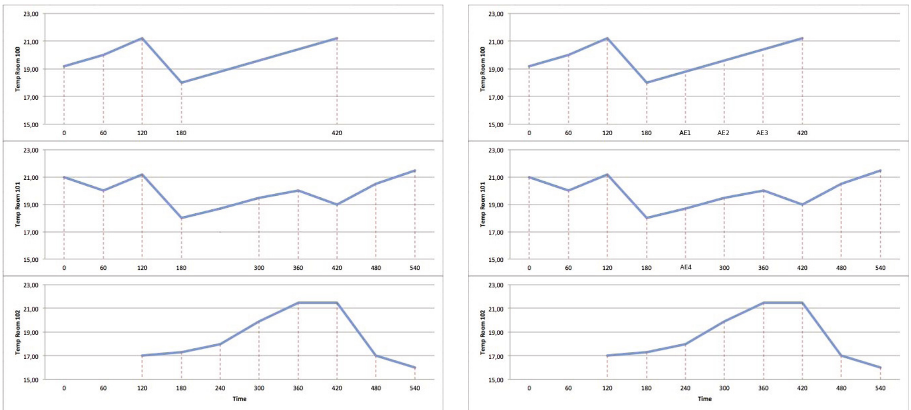
5 Querying I-OLAP Data

In this section, we will discuss how to answer queries on the I-OLAP model. We would like to emphasize that, due to a space limit, we will outline how our query language works, rather than its formal description. Basically, answering an I-OLAP query is done in the three steps discussed in this section.

5.1 Step 1: Getting Query Time Frame

The initial step is to get the time frame defined in a query. We assume that the underlying data warehouse has at least one time dimension (which will usually also serve as an ordering attribute for events). The dimension members selected by the user for the time dimension are extracted. This may be the *All* node (the root node) of the time dimension, i.e. all events, or any subset of dimension members belonging to the time dimension, i.e., the subset of events.

For example, for a given query: 'compute the number of minutes the light has been turned on in room 100 between timestamp 2013.01.01 and 2013.01.31', the time frame would be defined by $t_S = 2013.01.01$ and $t_E = 2013.01.31$.



(a) Before step 2

(b) After step 2

Fig. 3. Sequences of intervals before and after applying step 2

5.2 Step 2: Inserting Artificial Events

Artificial events are used to guarantee a uniform distribution of events over all sequences of intervals. For instance, in the sequences of intervals for the temperature depicted in Figure 3(a) there are no such events defined for time point $t = 240$, $t = 300$, and $t = 360$ - for room 101, and for time point $t = 240$ - for room 101. In order to allow queries to aggregate data over multiple sequences of intervals, we extend each interval sequence with artificial events for t_S and t_E returned by step 1.

Extending a sequence of intervals with an artificial event at time point t is done in two steps: (1) inserting new events and (2) adopting all affected intervals. Figure 3(b) depicts the results of inserting artificial events, denoted as AE_i. Artificial events are inserted by the algorithm outlined in Listing 1.4.

We would like to emphasize that this only happens on a conceptual level. Each meaningful implementation of the model would first select all interval sequences affected by the query and enrich by artificial events only these interval sequences.

Next, adopting affected intervals takes the set of sequences of intervals as an input and creates a uniform event distribution over all sequences of intervals, cf. the pseudocode in Listing 1.2. As a result, the following condition is fulfilled: if there exists an event e with $e.t = T$ in any sequence than there also exist events e' in all other sequences of intervals with $e'.t = T$.

Listing 1.4. Creating an artificial event

```
//INPUT: t − time point
// I − sequence of intervals
//OUTPUT: new sequence of intervals
function createArtificialEvent(t, I) {
    e_prev = event e with max(e.t) ≤ t;
    if (e_prev.t = t) return I
    //create new event
    e_new = e_prev
        e_new.t = t
    foreach (f ∈ I.F_CV)
        e_new.value = f(t)
    end foreach
    //insert new event into sequence
    //of intervals
    I_new = Insert(e_new, I)
    return I_new }
```

Listing 1.5. Uniformly distributing an artificial event among sequences of intervals

```
//INPUT: I set of sequences of intervals
//OUTPUT: new set of sequences of intervals
function createUniformEventDistr(I) {
 foreach (SI ∈ I)
   i = SI.first()
   while (i ≠ NULL)
     foreach (SI' ∈ I)
       j = SI'.first()
       while (j ≠ NULL)
         if (j.start() = i.start) exists = true
         j = SI'.next()
       end while
       if (false = exists)
         I'=I' ∪ createArtificialEvent(i.start(),SI')
     end foreach
     i = SI.next()
   end while
 end foreach
 return I' }
```

5.3 Step 3: Aggregating Measures

In this section we outline how to aggregate data using aggregate functions. Although we illustrate this step with the average function (AVG), the method is applicable to other aggregation functions such as MIN, MAX, SUM, and COUNT. We will show how to aggregate measures using two scenarios, namely: (1) *time point aggregation*, e.g., "what is the average temperature in all rooms at time point t" and (2) *aggregation along time*, e.g. "what is the average temperature in all rooms between time point t_1 and t_2".

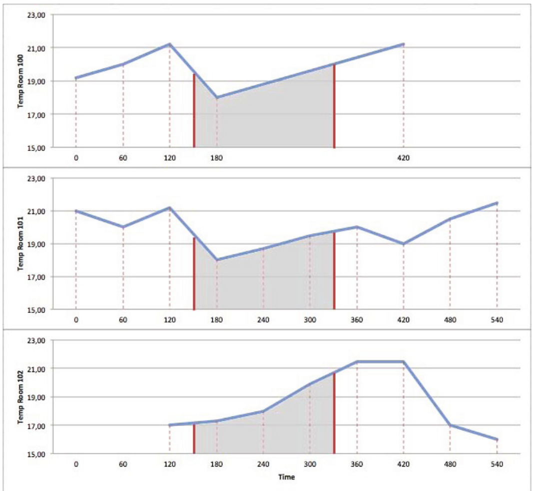


Fig. 4. Aggregation between t_1 and t_2

Table 3. Resulting temperatures for all rooms for $150 \leq t \leq 330$

Room	t=150	t=180	t=240	t=300	t=330	**AVG**
100	19,2	18,0	-	-	20,0	**19,07**
101	19,1	18,2	18,8	19,2	19,6	**18,98**
102	17,0	17,2	18,0	20,0	20,8	**18,60**
Floor						**18,85**

Answering the first query is straight forward. We simply call the corresponding function defined in $\mathbb{F}_{\mathbb{CV}}$ for all facts fulfilling the selection predicate. For instance, in our example we call $TempAtT(t)$ for interval sequences for all rooms.

The second query will be executed as follows. First, we call $TempAtT(t)$ for $t = t_1$ as well as for $t = t_2$. Second, we fetch all events for all sequences of intervals. For each event e, we call $TempAtT(t)$ with $t = e.t$. Figure 4 depicts this technique for $t_1 = 150$ and $t_2 = 330$. The resulting values are given in Table 3.

6 Implementation

We prototypically implemented this approach as a web application using PHP, the *PHP PEG* package[1] (a package used for defining PEGs - parsing expression grammar - and parsing strings into objects) and PostgreSQL.

As we are currently working on a query language enabling the user to analyze sequences in OLAP cubes, called S-SQL (Sequential SQL), we implemented this interval based approach as an extension of S-SQL [1]. Due to space limitations, we cannot give a detailed description of S-SQL. Basically, S-SQL statements enable users to formulate queries in order to analyze sequences using different functions like for instance HEAD(), TAIL() or PATTERN(). The S-SQL prototype consists

[1] PHP PEG has been developed by Hamish Friedlander. Available at: `https://github.com/hafriedlander/php-peg`

of a parser translating S-SQL into objects and an engine which then creates standard SQL statements out of these objects. As an example the query given in listing 1.6 might be used to fetch all sequences of events that fulfill a given pattern (A,*,B) and where the temperature was below 19 degrees at the start and the end of the day.

Listing 1.6. Sample S-SQL Query

```
SELECT *
FROM t1
WITH PATTERN 'a,*,b' BIND (a,b) TO sensor.heating ON SEQUENCE room
WHERE a.value < 19 AND b.value < 19;
```

This simple query would translate into a SQL query with over 40 lines of code (formatted). Other queries we tested resulted in queries with up to 160 lines of code.

We extended the functionality of S-SQL in order to be able to parse and execute statements using functions defined in $\mathbb{F}_{\mathbb{CV}}$ and $\mathbb{F}_{\mathbb{FC}}$. In it's current version these functions have to be defined as PL/pgSQL functions. The web service is used to parse the metadata of a given database and apply these functions to the defined intervals. Internally, the three steps as described in section 5 (getting the query time frame, inserting artificial events and aggregating measures) will be applied to the intervals. This enables us to state queries like:

Listing 1.7. Sample I-SQL Query

```
SELECT AVG(value)
FROM t1
WHERE left(sensor_id,1) = 'H' AND
        time >= '2013-03-20 00:00:00' AND time <= '2013-03-20 23:59:59'
```

This query would return the average temperature of all rooms on March, 20th. The implementation would automatically apply the three steps described above to get a correct result.

7 Summary

In this paper we proposed a formal model for processing interval data in an OLAP style and a prototypical implementation. To the best of our knowledge, no such model has been proposed before. The model supports: (1) defining multiple sets of intervals over sequential data, (2) define measures computed from both, events and intervals, and (3) analyze the measures in the context set up by dimensions - to this end we proposed the *iCube*. The formal model was reflected in an implementation model that we also proposed. We shown how to apply the model to querying I-OLAP data. In the next step we will develop physical data structures for supporting I-OLAP queries and evaluate their performance. Future work will focus on analyzing and developing methods to represent interval data by means of functions, similarly as proposed in [8] - for moving objects and in [14] - for interpolating values returned by sensors.

References

1. Retr. March 31, 2014, http://solap-isys.aau.at
2. Aster nPath,
http://developer.teradata.com/aster/articles/aster-npath-guide
(retr. from March 13, 2014)
3. Bębel, B., Morzy, M., Morzy, T., Królikowski, Z., Wrembel, R.: OLAP-like analysis of time point-based sequential data. In: Castano, S., Vassiliadis, P., Lakshmanan, L.V.S., Lee, M.L. (eds.) ER 2012 Workshops 2012. LNCS, vol. 7518, pp. 153–161. Springer, Heidelberg (2012)
4. Chui, C.K., Kao, B., Lo, E., Cheung, D.: S-OLAP: an olap system for analyzing sequence data. In: Proc. of ACM SIGMOD Int. Conf. on Management of Data, pp. 1131–1134. ACM (2010)
5. Chui, C.K., Lo, E., Kao, B., Ho, W.-S.: Supporting ranking pattern-based aggregate queries in sequence data cubes. In: Proc. of ACM Conf. on Information and Knowledge Management (CIKM), pp. 997–1006. ACM (2009)
6. Gonzalez, H., Han, J., Li, X.: FlowCube: constructing RFID flowcubes for multi-dimensional analysis of commodity flows. In: Proc. of Int. Conf. on Very Large Data Bases (VLDB), pp. 834–845. VLDB Endowment (2006)
7. Gonzalez, H., Han, J., Li, X., Klabjan, D.: Warehousing and analyzing massive RFID data sets. In: Proc. of Int. Conf. on Data Engineering (ICDE) (2006)
8. Güting, R.H., Böhlen, M.H., Erwig, M., Jensen, C.S., Lorentzos, N.A., Schneider, M., Vazirgiannis, M.: A foundation for representing and querying moving objects. ACM Trans. on Database Systems (TODS) 25(1), 1–42 (2000)
9. Han, J., Chen, Y., Dong, G., Pei, J., Wah, B.W., Wang, J., Cai, Y.D.: Stream Cube: An architecture for multi-dimensional analysis of data streams. Distributed and Parallel Databases 18(2), 173–197 (2005)
10. Liu, M., Rundensteiner, E., Greenfield, K., Gupta, C., Wang, S., Ari, I., Mehta, A.: E-Cube: multi-dimensional event sequence analysis using hierarchical pattern query sharing. In: Proc. of ACM SIGMOD Int. Conf. on Management of Data, pp. 889–900. ACM (2011)
11. Liu, M., Rundensteiner, E.A.: Event sequence processing: new models and optimization techniques. In: Proc. of SIGMOD PhD Workshop on Innovative Database Research (IDAR), pp. 7–12 (2010)
12. Lo, E., Kao, B., Ho, W.-S., Lee, S.D., Chui, C.K., Cheung, D.W.: OLAP on sequence data. In: Proc. of ACM SIGMOD Int. Conf. on Management of Data, pp. 649–660 (2008)
13. Mörchen, F.: Unsupervised pattern mining from symbolic temporal data. SIGKDD Explor. Newsl. 9(1), 41–55 (2007)
14. Thiagarajan, A., Madden, S.: Querying continuous functions in a database system. In: Proc. of ACM SIGMOD Int. Conf. on Management of Data, pp. 791–804 (2008)
15. Witkowski, A.: Analyze this! Analytical power in SQL, more than you ever dreamt of. Oracle Open World (2012)
16. Ya-Han, H., Tony Cheng-Kui, H., Hui-Ru, Y., Yen-Liang, C.: On mining multi-time-interval sequential patterns. Data & Knowledge Engineering 68(10), 1112–1127 (2009)
17. Yen-Liang, C., Mei-Ching, C., Ming-Tat, K.: Discovering time-interval sequential patterns in sequence databases. Expert Systems with Applications 25(3), 343–354 (2003)