sciendo

Int. J. Appl. Math. Comput. Sci., 2019, Vol. 29, No. 1, 17–30
DOI: 10.2478/amcs-2019-0002

amcs

ONTOLOGY–BASED ACCESS TO TEMPORAL DATA WITH ONTOP: A FRAMEWORK PROPOSAL

ELEM GÜZEL KALAYCI [a], SEBASTIAN BRANDT [b], DIEGO CALVANESE [a], VLADISLAV RYZHIKOV [c], GUOHUI XIAO [a,*], MICHAEL ZAKHARYASCHEV [c,d]

[a] KRDB Research Centre for Knowledge and Data
Free University of Bozen-Bolzano, Piazza Domenicani 3, 39100 Bolzano, Italy
e-mail: xiao@inf.unibz.it

[b] Siemens CT
Otto-Hahn-Ring 6, 81739 München, Germany

[c] Department of Computer Science and Information Systems
Birkbeck, University of London, Malet St., London WC1E 7HX, UK

[d] National Research University Higher School of Economics
3 Kochnovsky Proezd, 125319, Moscow, Russia

Predictive analysis gradually gains importance in industry. For instance, service engineers at Siemens diagnostic centres unveil hidden knowledge in huge amounts of historical sensor data and use it to improve the predictive systems analysing live data. Currently, the analysis is usually done using data-dependent rules that are specific to individual sensors and equipment. This dependence poses significant challenges in rule authoring, reuse, and maintenance by engineers. One solution to this problem is to employ ontology-based data access (OBDA), which provides a conceptual view of data via an ontology. However, classical OBDA systems do not support access to temporal data and reasoning over it. To address this issue, we propose a framework for temporal OBDA. In this framework, we use extended mapping languages to extract information about temporal events in the RDF format, classical ontology and rule languages to reflect static information, as well as a temporal rule language to describe events. We also propose a *SPARQL*-based query language for retrieving temporal information and, finally, an architecture of system implementation extending the state-of-the-art OBDA platform Ontop.

Keywords: metric temporal logic, ontology-based data access, *SPARQL* query, Ontop.

1. Introduction

Analysis of log sensor data is an important problem in industry as it reveals crucial insights into the performance and conditions of devices. The outcomes of this analysis, known as retrospective diagnostics, enable IT experts to improve the capabilities of real-time systems monitoring abnormal or potentially dangerous events developing in devices, in particular, the systems that perform predictive diagnostics. For complex devices (including those we consider in the use case below), such events do not amount to simply measurable instances (say, the temperature above 100°C). Instead, they involve a number of measurements from sensors attached to a device, each having a certain temporal duration and occurring in a certain temporal sequence.

In this paper we focus on a use case by Siemens, which maintains thousands of devices related to power generation, including gas and steam turbines. It monitors these devices and provides operational support for them through a global network of more than 50 remote diagnostic centres that are linked to a common database centre. Siemens wants to employ retrospective and predictive diagnostics in order to anticipate problems with turbines and take appropriate countermeasures. A

*Corresponding author

major challenge in this task comes from the combined need of dealing with complex *temporal information* and with *heterogeneous* data, considering that the various turbine models have different schemas of the underlying databases storing sensor measurements.

To deal with heterogeneity of data, we rely on *ontology-based data access* (OBDA), which was first suggested by Calvanese *et al.* (2007) and Poggi *et al.* (2008) as a means to detach the conceptual layer of classes and properties, to be exposed to end-users, from the complex structure of the underlying data sources, which thus can be hidden to users. In fact, those classes and properties are mapped to the data source schemas by means of a declarative specification. In addition, an ontology is used to model the domain of interest by asserting conceptual relations (for instance, isA) between the classes and properties. In the solution we propose here, OBDA allows us to detach the conceptual view of an event located in time—such as 'HighRotorSpeed of turbine tb01 in the period from 2017-06-06 12:20:00 to 2017-06-06 12:20:03'—from concrete databases that store the log of the sensors of that turbine.

There are several systems implementing the OBDA approach, some of which (e.g., Ontop[1] and Morph[2]) are freely available, while others (e.g., Stardog[3], Mastro[4], and Ultrawrap[5]) are distributed under commercial licences. For a recent survey of OBDA, we refer to the work by Xiao *et al*. (2018).

Unfortunately, none of the available OBDA systems supports access to *temporal data* well enough to work with the events relevant to our use case. On the one hand, the common mapping languages are not tailored towards extracting validity time intervals of conceptual assertions. On the other hand—and this is a more serious limitation—the supported ontology languages do not allow one to construct classes and properties whose temporal validity depends on the validity of other classes and properties, which is essential for defining complex temporal events. In fact, the OBDA systems used industrially are based on lightweight (non-temporal) ontology languages, such as the *OWL 2 QL* profile of the Web Ontology Language *OWL 2*, in order to guarantee maximally efficient query answering by a reduction to standard database query evaluation. When limited to a classical ontology language, one approach to enable the extraction of temporal events is by extending the end-user query language with various temporal operators (see, e.g., Gutiérrez-Basulto and Klarman, 2012; Baader *et al*., 2013; Borgwardt *et al*., 2013; Möller *et al*., 2013; Klarman and Meyer, 2014; Özçep *et al*., 2014; Kharlamov *et al*., 2016). However, this approach leaves the burden of encoding the complex events in temporal queries to the end-user. In the Siemens scenario, this is a prohibitive limitation since the end-users are service engineers who are not trained in temporal knowledge representation.

Therefore, we are interested in a more expressive setting, where the *ontology* language is extended by temporal operators that are capable of defining complex temporal events. Extensions of lightweight ontology languages with temporal operators of *linear temporal logic* (*LTL*) such as 'next time', 'always', and 'eventually' have been suggested by Artale *et al.* (2013; 2015a) and Gutiérrez-Basulto *et al.* (2016a). However, in our use case and other similarly complex scenarios, sensors tend to send data at irregular time intervals. Moreover, even if sensor data arrives regularly, due to deadband settings the data might be stored in a data collector only when the last arrived sensor measurement is within a certain value distance from the previously transmitted one. To cope with this situation, one could replace point-based *LTL* with interval-based temporal logics. Thus, Artale *et al.* (2015b) and Kontchakov *et al*. (2016) proposed extensions of ontology languages and datalog programs with the Allen operators on temporal intervals used in the Halpern–Shoham logic, $\mathcal{HS}$ (Halpern and Shoham, 1991). Unfortunately, it is not possible to express in $\mathcal{HS}$ numerical constraints such as 'within the next 10 minutes, main flame will continuously be on for at least 10 seconds.' A more suitable temporal representation formalism for our use case is the *metric temporal logic*, *MTL*, over a dense timeline, which was originally introduced for modeling and reasoning about real-time systems (Koymans, 1990; Alur and Henzinger, 1993). In fact, the extension *datalogMTL* of datalog with *MTL* operators proposed by Brandt *et al.* (2017a) appears to be both capable of capturing the events of interest for the diagnostic tasks at Siemens and suitable for the OBDA scenario. It is to be noted that *MTL* extensions of the expressive ontology language $\mathcal{ALC}$ over discrete time have been recently considered by Gutiérrez-Basulto *et al.* (2016b) and Baader *et al*. (2017).

In this paper, through a running example from the Siemens use case, we present a framework for temporal OBDA that employs as an ontology language to describe temporal events $datalog_{nr}MTL$ the non-recursive version of *datalogMTL*. This framework also supports the standard *OWL 2 QL* language to model static knowledge (such as a configuration of modules of a turbine), and extends it with non-recursive datalog rules to describe static knowledge of a more complex structure. We outline extensions of the standard mapping language *R2RML* (Das *et al*., 2012) and the query language *SPARQL* to extract information on the validity intervals of temporal predicates. Finally, we discuss an implementation of the proposed framework in the OBDA system Ontop.

[1] `http://ontop.inf.unibz.it`.
[2] `https://github.com/oeg-upm/morph-rdb`.
[3] `http://stardog.com`.
[4] `http://www.obdasystems.com/mastro`.
[5] `https://capsenta.com`.

2. Framework for temporal OBDA

Recall that, in classical (non-temporal) OBDA, an *OBDA specification* is a triple $\mathcal{P} = \langle \mathcal{O}, \mathcal{M}, \mathcal{S} \rangle$, where $\mathcal{O}$ is an *OWL 2 QL* ontology, $\mathcal{S}$ a database schema, and $\mathcal{M}$ a set of *R2RML* mapping assertions, each associating to a class or property in $\mathcal{O}$ an *SQL* query over $\mathcal{S}$. An *OBDA instance* is a pair $\langle \mathcal{P}, D \rangle$, where D is a database instance satisfying schema $\mathcal{S}$. Intuitively, by *applying* the mapping $\mathcal{M}$ to the data instance D, which consists in executing the *SQL* queries in the mapping assertions over D and populating the corresponding classes and properties using the returned values, we would obtain an RDF graph $\mathcal{M}(D)$ that reflects the content of D at the ontology level. The ontology $\mathcal{O}$ complements the data with background knowledge and provides a convenient vocabulary for user queries, which are formulated in the W3C standard language *SPARQL*. A *certain answer* to such a *SPARQL* query $\boldsymbol{q}(\vec{x})$ over $\langle \mathcal{P}, D \rangle$ is any tuple $\vec{a}$ from D for which $\boldsymbol{q}(\vec{a})$ holds in all models of $\mathcal{O}$ and $\mathcal{M}(D)$. To find the certain answers, the OBDA system rewrites the *ontology-mediated query* (OMQ for short) $(\mathcal{O}, \boldsymbol{q})$ into an *SQL* query $\boldsymbol{q}'(\vec{x})$ over $\mathcal{S}$ that satisfies the following condition: for every data instance D complying with $\mathcal{S}$ and every tuple $\vec{a}$ in it, we have that $\mathcal{O}, \mathcal{M}(\mathcal{D}) \models \boldsymbol{q}(\vec{a})$ if and only if $\vec{a}$ is an answer to $\boldsymbol{q}'(\vec{x})$ over D. Thus, answering ontology-mediated queries is reduced to standard database query evaluation. Consequently, it is in AC^0 for data complexity. For more details and further references, we refer to the survey by Xiao *et al.* (2018).

In the remainder of this section, we present our framework for *temporal* OBDA by introducing temporal OBDA specifications and instances, as well as a query language for those instances based on a variant of τ-*SPARQL* (Tappolet and Bernstein, 2009).

2.1. Temporal OBDA specification.

Since we want to develop temporal OBDA by extending the standard non-temporal OBDA paradigm, we now call the *OWL 2 QL* ontology $\mathcal{O}$ above a *static ontology* and $\mathcal{M}$ a set of *static mapping assertions*. In what follows, we will extend the *static* vocabulary Σ_s of classes and properties occurring in $\mathcal{O}$ and $\mathcal{M}$ by a disjoint *temporal* vocabulary Σ_t. We now describe static ontologies in greater detail using an example from the Siemens use case.

2.1.1. Static ontology.

At Siemens, the devices used for power generation are monitored by various types of sensors that report the temperature, pressure, vibration, rotor speed, and other relevant measurements. In order to model the static knowledge regarding the machines and their deployment profiles, sensor configurations, component hierarchies, and functional profiles, Siemens designed an *OWL 2 QL* ontology (Kharlamov *et al.*, 2017), a snippet of which is given in Example 1 below using the syntax of description logics (Baader *et al.*, 2007).

Example 1. The signature Σ_s of the Siemens static ontology $\mathcal{O}$ contains the following classes (in the first three lines) and properties (in the fourth line)[6]:

$$
\begin{array}{l}
\mathsf{Train},\ \mathsf{Turbine},\ \mathsf{GasTurbine},\ \mathsf{SteamTurbine},\\
\mathsf{TurbinePart},\ \mathsf{PowerTurbine},\ \mathsf{Burner},\ \mathsf{Sensor},\\
\mathsf{RotationSpeedSensor},\ \mathsf{TemperatureSensor},\\
\mathsf{isMonitoredBy},\ \mathsf{isPartOf},\ \mathsf{isDeployedIn}.
\end{array}
$$

Some of the axioms (inclusions and equivalences between classes) from $\mathcal{O}$ are shown below:

$$
\begin{aligned}
\mathsf{GasTurbine} &\sqsubseteq \mathsf{Turbine},\\
\mathsf{SteamTurbine} &\sqsubseteq \mathsf{Turbine},\\
\mathsf{RotationSpeedSensor} &\sqsubseteq \mathsf{Sensor},\\
\mathsf{TemperatureSensor} &\sqsubseteq \mathsf{Sensor},\\
\mathsf{PowerTurbine} &\sqsubseteq \mathsf{TurbinePart},\\
\mathsf{Burner} &\sqsubseteq \mathsf{TurbinePart},\\
\exists\mathsf{isMonitoredBy} &\sqsubseteq \mathsf{TurbinePart},\\
\exists\mathsf{isMonitoredBy}^- &\sqsubseteq \mathsf{Sensor},\\
\exists\mathsf{isPartOf} &\equiv \mathsf{TurbinePart},\\
\exists\mathsf{isPartOf}^- &\sqsubseteq \mathsf{Turbine},\\
\exists\mathsf{isDeployedIn} &\sqsubseteq \mathsf{Turbine},\\
\exists\mathsf{isDeployedIn}^- &\sqsubseteq \mathsf{Train}.
\end{aligned}
$$

For a property P, the expression $\exists P$ denotes the domain of P, while $\exists P^-$ denotes the range of P. Thus, the last two axioms say that the domain of the property $\mathsf{isDeployedIn}$ is $\mathsf{Turbine}$ and the range is Train. ♦

Unfortunately, *OWL 2 QL* has a limited expressive power and is not able to capture all the static knowledge that is relevant to the Siemens use case. In particular, it does not allow predicates of arity greater than 2 and intersection on the left-hand side of inclusions. A well-known language with these missing constructs is standard datalog (Abiteboul *et al.*, 1995). Note, however, that answering datalog queries is P-complete for data complexity, and so it cannot be reduced to database query evaluation in general. A typical example of a datalog rule from our ontology is given in the next section.

2.1.2. Static rules.

In the Siemens use case, some turbine parts are monitored by a number of different sensors, say, a temperature sensor and a rotation speed sensor. This situation can be readily described by a datalog rule with a ternary predicate in the head and a complex body, such as the one in the example below, but not by *OWL 2 QL* axioms.

[6]In description logic parlance, classes are called *concepts* and correspond to unary predicates, while properties are called *roles* and correspond to binary predicates.

Example 2. The datalog rule

$$\begin{aligned}&\mathsf{ColocTempRotSensors}(tb, ts, rs) \leftarrow\\ &\quad \mathsf{Turbine}(tb),\ \mathsf{isPartOf}(br, tb),\ \mathsf{Burner}(br),\\ &\quad \mathsf{isMonitoredBy}(br, ts),\ \mathsf{TemperatureSensor}(ts),\\ &\quad \mathsf{isPartOf}(pt, tb),\ \mathsf{PowerTurbine}(pt),\\ &\quad \mathsf{isMonitoredBy}(pt, rs),\ \mathsf{RotationSpeedSensor}(rs)\end{aligned}$$

is supposed to say that a temperature sensor ts and a rotation speed sensor rs are co-located in the same turbine tb if tb has a part br, which is a burner, monitored by ts, and has another part pt, which is a power turbine, monitored by rs. ♦

We denote by $\mathcal{R}$ a set of static datalog rules and by Σ_s the static signature that contains the symbols from both $\mathcal{O}$ and $\mathcal{R}$. Note that the rules in $\mathcal{R}$ may contain classes and properties from $\mathcal{O}$. In order to make sure that answering queries mediated by $\mathcal{O} \cup \mathcal{R}$ is reducible to database query evaluation, we impose two restrictions on $\mathcal{R}$: *(i)* it has to be non-recursive and *(ii)* the predicates in the head of rules in $\mathcal{R}$ cannot occur in $\mathcal{O}$.

The static ontology language considered so far is supposed to represent time-independent knowledge and falls short of capturing temporal events that are required in the Siemens use case.

2.1.3. Temporal rules.

Siemens is interested in detecting abnormal situations in the working equipment as well as in monitoring running tasks in order to see whether they proceed ordinarily. A typical event that is crucial to monitor is a *normal start of a turbine*. This event is rather complex and composed of various subevents that are distributed over time and characterized by a temporal duration. One of these subevents, *Purging Is Over*, is described in the example below.

Example 3. *Purging Is Over* is a complex temporal event for a given turbine tb, which is characterized by the following:

(*i*) there is a pair of sensors co-located in the turbine tb, one of which is a rotor speed sensor rs and the other one a temperature sensor ts;

(*ii*) the temperature sensor ts detects that the main flame was burning for at least 10 seconds;

(*iii*) at the same time, the following happened within the preceding 10 minutes:
- the rotor speed sensor rs measured a speed of at most 1000 rpm for at least 30 seconds, and
- within the following 2 minutes, the rotor speed sensor rs measured a speed of at least 1260 rpm for at least 30 seconds.

We illustrate the described event in Fig. 10.

Here, we assume that the horizontal axis represents time and $\mathsf{PurgingIsOver}(tb)$ holds at a moment of time if prior to that moment $\mathsf{MainFlameOn}(ts)$, $\mathsf{LowRotorSpeed}(rs)$, and $\mathsf{HighRotorSpeed}(rs)$ had occurred following the depicted pattern.

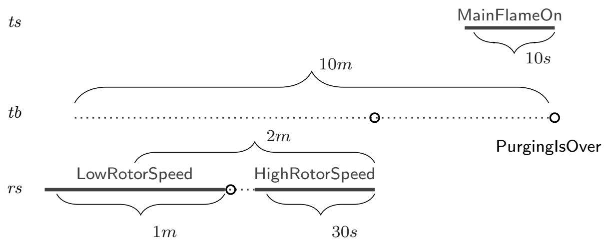

Fig. 1. Event of Example 3.

In our examples, we shall also use the temporal event *Main Flame Is On*, which happens for a given temperature sensor ts when the main flame had been above the threshold (of 1.0) for 10 seconds continuously in the past. ♦

Temporal diagnostic patterns of this sort can be described by means of a *datalog$_{nr}$MTL* program (Brandt *et al.*, 2017a), which is a set of non-recursive datalog rules extended with temporal operators of the metric temporal logic *MTL* under the continuous semantics over the real numbers $(\mathbb{R}, <)$ (Alur and Henzinger, 1993). In such programs, we require a (countably infinite) list of *temporal predicates* (with the corresponding arities) that is disjoint from the list of static predicates. Intuitively, each temporal predicate may be true on some domain objects at certain moments of time and false on other domain objects and time instants. Under this semantics, static predicates are assumed to be time-independent, that is, to hold true or false on a given tuple of domain objects at all times. The event *Purging Is Over* can be formalized by the following *datalog$_{nr}$MTL* program $\mathcal{T}$.

Example 4. The program $\mathcal{T}$ consists of five rules:

$$\begin{aligned}&\mathsf{PurgingIsOver}(tb) \leftarrow \mathsf{MainFlameOn}(ts),\\ &\quad \diamondminus_{(0,10m]}(\boxminus_{(0,30s]}\mathsf{HighRotorSpeed}(rs),\\ &\qquad\quad \diamondminus_{(0,2m]}\boxminus_{(0,1m]}\mathsf{LowRotorSpeed}(rs)),\\ &\quad \mathsf{ColocTempRotSensors}(tb, ts, rs),\end{aligned}$$

$$\mathsf{MainFlameOn}(ts) \leftarrow \boxminus_{[0s,10s]}\mathsf{MainFlameUpTH}(ts),$$

$$\mathsf{MainFlameUpTH}(ts) \leftarrow \mathsf{mainFlame}(ts, v),\ v \geq 1.0,$$

$$\mathsf{HighRotorSpeed}(rs) \leftarrow \mathsf{rotorSpeed}(rs, v),\ v > 1260,$$

$$\mathsf{LowRotorSpeed}(rs) \leftarrow \mathsf{rotorSpeed}(rs, v),\ v < 1000.$$

Here, $\mathsf{ColocTempRotSensors}$ is the static (time independent) predicate from Example 2. The unary numerical built-in predicates $v \geq 1.0$, $v > 1260$, and $v < 1000$ are also static. The predicates $\mathsf{rotorSpeed}$, $\mathsf{HighRotorSpeed}$, $\mathsf{LowRotorSpeed}$, and $\mathsf{MainFlameUpTH}$ are temporal; $\mathsf{rotorSpeed}(rs, v)$ holds true at a time instant t if and only if v is the rotor speed measured by the sensor rs at t; $\mathsf{HighRotorSpeed}(rs)$ holds true exactly at those time instants t where $\mathsf{rotorSpeed}(rs, v)$ holds for some value $v > 1260$; and similarly for LowRotorSpeed and MainFlameUpTH. The temporal predicate MainFlameOn is defined using the *MTL* operator $\boxminus_{[0s,10s]}$: namely, $\mathsf{MainFlameOn}(ts)$ holds true at time instant t if $\mathsf{MainFlameUpTH}(ts)$ holds at *all* time instants $t' \in [t-10s, t]$. Finally, PurgingIsOver is a temporal predicate that, in addition to the $\boxminus$ operator, uses the *MTL* operator $\diamondminus$. For example, $\diamondminus_{(0,10m]}$ is interpreted as follows: $\diamondminus_{(0,10m]}\,\varphi$ holds true at t if and only if φ holds true at *some* time instant $t' \in [t-10m, t)$. ♦

We denote by Σ_t the set of temporal predicates from $\mathcal{T}$ and note that, in *datalog$_{nr}$MTL* programs like $\mathcal{T}$, only predicates from Σ_t can occur in the head of the rules, whereas their bodies can contain predicates from both Σ_t and Σ_s. (Thus, intuitively, the temporal rules in $\mathcal{T}$ define temporal predicates in terms of both temporal and static ones.) In the example above, Σ_t comprises the predicates

mainFlame, rotorSpeed,
PurgingIsOver, MainFlameOn, MainFlameUpTH,
HighRotorSpeed, LowRotorSpeed.

We emphasise once again that *datalog$_{nr}$MTL* programs are required to be non-recursive. Without such a restriction, the data complexity of answering ontology-mediated queries in our framework becomes P-hard, which makes a reduction to database query evaluation impossible. For non-recursive *datalog$_{nr}$MTL* rules, the AC^0 data complexity follows from the work by Brandt *et al.* (2017a), if we restrict the *OWL 2 QL* ontologies to the *DL-Lite$_{rdfs}$* fragment (Calvanese *et al.*, 2007). Currently, we are working on extending this result to full *OWL 2 QL* in a way similar to the extension of *OWL 2 QL* with the temporal operators of linear temporal logic *LTL* (Artale *et al.*, 2015a).

2.1.4. Databases and mappings. In our approach, we assume that databases have generic schemas. However, since in temporal OBDA, we have to deal with temporal data, we are particularly interested in databases with tables containing timestamp columns.

Example 5. An example data schema $\mathcal{S}$ for the Siemens data, including sensor measurements and deployment details, can look as follows (the primary keys of the tables are underlined):

tb_measurement(tstmp, s_id, value),
tb_sensors(s_id, s_type, mnted_part, mnted_tb),
tb_deployment(turbine_id, turbine_type, deployed_in),
tb_components(turbine_id, comp_id, comp_type).

Table 1. Example Siemens use case tables.

tb_measurement		
tstmp	s_id	value
17-06-06 12:20:00	rs01	570
17-06-06 12:21:00	rs01	680
17-06-06 12:21:30	rs01	920
17-06-06 12:22:50	rs01	1278
17-06-06 12:23:40	rs01	1310
	...	
17-06-06 12:32:30	mf01	2.3
17-06-06 12:32:37	mf01	1.8
17-06-06 12:32:45	mf01	0.9
	...	

tb_sensors			
s_id	s_type	mnted_part	mnted_tb
rs01	0	pt01	tb01
mf01	1	b01	tb01
	...	...	

tb_components		
turbine_id	comp_id	comp_type
tb01	pt01	0
tb01	b01	1
	...	

Three snippets of data from the Siemens use case tables tb_measurement, tb_sensor, and tb_components, are given in Table 1. ♦

In classical OBDA, *mapping assertions* take the form $\varphi(\vec{x}) \rightsquigarrow \psi(\vec{x})$, where $\varphi(\vec{x})$ is a query over the schema $\mathcal{S}$ and $\psi(\vec{x})$ is an atom with a predicate from Σ_s and variables $\vec{x}$ (Xiao *et al.*, 2018).

Example 6. Given the static ontology $\mathcal{O}$ and the signature Σ_s from Example 1, as well as the schema $\mathcal{S}$ from Example 5, the following are examples of mapping assertions:

```
SELECT s_id AS X FROM tb_sensors
WHERE s_type = 1   ⇝   TemperatureSensor(X),

SELECT component_id AS X FROM tb_components
WHERE component_type = 1   ⇝   Burner(X),

SELECT mnted_part AS X, s_id AS Y
FROM tb_sensors   ⇝   isMonitoredBy(X, Y).
```

To explain how the mapping assertions work, consider the database from Example 5 and the *SQL* query on the left-hand side of the first assertion. The result of executing this query over the database is a table with a single column named `X` and containing a tuple `mf01`. The right-hand side of the mapping indicates that, according to the database, the fact TemperatureSensor(`mf01`) holds true (as well as other such facts if the query returns more answers). ♦

By applying these mapping assertions to the database from Example 5, we extract the following facts (ground atoms):

$$\mathsf{Burner}(\mathtt{b01}),\ \mathsf{TemperatureSensor}(\mathtt{mf01}),\ \mathsf{isMonitoredBy}(\mathtt{pt01},\mathtt{rs01}),\ \mathsf{isMonitoredBy}(\mathtt{b01},\mathtt{mf01}).$$

We call the mapping assertions that extract ground atoms for predicates from Σ_s *static*, and use $\mathcal{M}_s$ to denote their sets.

On the other hand, to deal with temporal predicates from Σ_t, we also require *temporal mapping assertions* of the form

$$\varphi(\vec{x},\mathtt{begin},\mathtt{end}) \ \rightsquigarrow\ \psi(\vec{x})@\langle t_{\mathtt{begin}}, t_{\mathtt{end}}\rangle,$$

where $\varphi(\vec{x},\mathtt{begin},\mathtt{end})$ is a query over $\mathcal{S}$ such that the variables begin and end are mapped to values of the date/time format, ψ is a predicate from Σ_t, $t_{\mathtt{begin}}$ is either the variable begin or a constant temporal value (timestamp) including ∞, $-\infty$ (and similarly for $t_{\mathtt{end}}$), '$\langle$' is either '(' or '[', and '$\rangle$' is either ')' or ']'. For example, $\psi(\vec{x})@[\mathtt{begin},\infty)$ means that $\psi(\vec{x})$ holds at every time instant in the interval $[\mathtt{begin},\infty)$, and the variables $\vec{x}$ and begin are instantiated by the query on the left-hand side of the mapping assertion. Temporal mapping assertions are required to define predicates from Σ_t only. We denote by $\mathcal{M}_t$ sets of such mapping assertions.

Example 7. Given Σ_t and $\mathcal{T}$ from Example 4 and the schema $\mathcal{S}$ from Example 5, the following is an example of a temporal mapping assertion:

```
SELECT s_id, value,
       tstmp AS begin,
       LEAD(tstmp, 1) OVER W AS end
FROM tb_measurement, tb_sensors
WHERE tb_measurement.s_id = tb_sensors.s_id
      AND s_type = 1
WINDOW W AS (PARTITION BY s_id ORDER BY tstmp)
⇝    mainFlame(s_id, value)@[begin, end).
```

This mapping assertion extracts from the database in Example 5 the following temporal facts:

$$\mathsf{mainFlame}(\mathtt{mf01}, 2.3)@[12{:}32{:}30, 12{:}32{:}37),\ \mathsf{mainFlame}(\mathtt{mf01}, 1.8)@[12{:}32{:}37, 12{:}32{:}45).$$

For instance, the first of them states that the main flame sensor mf01 was registering the value 2.3 in the interval [12:32:30, 12:32:37). Note that the interval is left-closed and right-open, which reflects the logic of how turbine sensor outputs are produced: namely, a sensor outputs a value when the result of a measurement differs from the previously returned value by a fixed threshold. Similarly, by means of an appropriate mapping, we shall extract the temporal fact stating that the rotation sensor rs01 was registering the speed 570 in the interval [12:20:00, 12:21:00). ♦

Table 2. Languages of the components in temporal OBDA.

component	defines predicates in	in terms of predicates in	language
$\mathcal{M}_s$	Σ_s	$\mathcal{S}$	*R2RML*
$\mathcal{M}_t$	Σ_t	$\mathcal{S}$	*R2RML*
$\mathcal{O}$	Σ_s	Σ_s	*OWL 2 QL*
$\mathcal{R}$	Σ_s	Σ_s	non-recursive datalog
$\mathcal{T}$	Σ_t	$\Sigma_s \cup \Sigma_t$	*datalog$_{nr}$MTL*

2.1.5. Temporal OBDA specification and instance. An *OBDA specification* in the temporal OBDA framework is the tuple

$$\mathfrak{S} = \langle \Sigma_s, \Sigma_t, \mathcal{M}_s, \mathcal{M}_t, \mathcal{O}, \mathcal{R}, \mathcal{T}, \mathcal{S}\rangle,$$

where Σ_s is a static signature, Σ_t a temporal signature, $\mathcal{M}_s$ a set of static and $\mathcal{M}_t$ a set of temporal mapping assertions, $\mathcal{O}$ is a standard OBDA ontology in Σ_s, $\mathcal{R}$ a set of static rules in Σ_s, $\mathcal{T}$ a set of temporal rules in Σ_t, and $\mathcal{S}$ a database schema. In Table 2, we clarify the intuition behind the different components of $\mathfrak{S}$ and the associated specification languages. A *temporal OBDA instance* $\mathfrak{I}$ is a pair $\langle \mathfrak{S}, D\rangle$, where $\mathfrak{S}$ is a temporal OBDA specification and D a database instance compliant with the database schema $\mathcal{S}$ in $\mathfrak{S}$. We next discuss languages for querying temporal OBDA instances.

2.2. Ontology-mediated query answering. A popular query language in standard (atemporal) OBDA is the language of conjunctive queries (Calvanese *et al.*, 2007). In our temporal setting, a *conjunctive query* is a first-order formula of the form

$$\boldsymbol{q}(\vec{x},\vec{\iota}) \ = \ \exists \vec{y}, \vec{\tau}\ \Phi(\vec{x},\vec{y},\vec{\iota},\vec{\tau}),$$

where Φ is a conjunction of atoms of the form $P(\vec{z})$ with P from Σ_s, atoms of the form $Q(\vec{z})@\varrho$ with Q from Σ_t, $\vec{z} \subseteq \vec{x} \cup \vec{y}$ and $\varrho \in \vec{\iota} \cup \vec{\tau}$, and possibly built-in numerical predicates over the timeline. Here, ϱ is a variable over *temporal intervals* (and $\vec{\iota}$, $\vec{\tau}$ are lists of such variables). A *certain answer* to $\boldsymbol{q}(\vec{x},\vec{\iota})$ over a temporal OBDA instance $\mathfrak{I}$ is a tuple $\vec{a}$ of constants from D and a tuple $\vec{\alpha}$ of temporal intervals (of the form defined above, say, [12:20:00, 15:00:00) or [12:20:00, ∞)) such that $\boldsymbol{q}(\vec{a},\vec{\alpha})$ holds true in every temporal model of the ontology $\mathcal{O} \cup \mathcal{R} \cup \mathcal{T}$ and the sets of ground atoms extracted from D by $\mathcal{M}_s$ and $\mathcal{M}_t$. One could also require that the intervals in $\vec{\alpha}$ only use those time instants that are explicitly mentioned in D. A more expressive query language can be obtained by extending conjunctive queries with Allen's interval relations such as 'ι is after τ' (Allen, 1983).

Example 8. The conjunctive query

$$\boldsymbol{q}_1(\varrho) = \mathsf{MainFlameOn}(x)@\varrho$$

can be used to find the periods of time when the main flame was on for some sensor, and the conjunctive query

$$\boldsymbol{q}_2(x,\varrho) = \mathsf{GasTurbine}(x) \wedge \mathsf{isDeployedIn}(x, \mathsf{tr05}) \wedge \mathsf{PurgingIsOver}(x)@\varrho$$

can be used to find the gas turbines that were deployed in the train with the ID tr05 and the time periods of their accomplished purgings. Observe that, for the temporal facts shown in Example 7, the program $\mathcal{T}$ from Example 4 and $\mathcal{R}$, $\mathcal{O}$ as above, the certain answer to the query $\boldsymbol{q}_1$ will be the interval [12:32:40, 12:32:45). This is because $\mathsf{MainFlameUpTH}(\mathtt{mf01})@$[12:32:30, 12:32:37) and $\mathsf{MainFlameUpTH}(\mathtt{mf01})@$[12:32:37, 12:32:45) (and so also $\mathsf{MainFlameUpTH}(\mathtt{mf01})@$[12:32:30, 12:32:45)) hold true in every temporal model of the ontology $\mathcal{O} \cup \mathcal{R} \cup \mathcal{T}$ and the sets of ground atoms extracted from the data instance D by means of the mappings $\mathcal{M}_s$ and $\mathcal{M}_t$, and so the same holds also for $\mathsf{MainFlameOn}(\mathtt{mf01})@$[12:32:40, 12:32:45). ♦

Answering conjunctive queries in temporal OBDA turns out to be a difficult problem, in both theory and practice; see, e.g., the work by Artale *et al.* (2013; 2015a), who considered extensions of *OWL 2 QL* and conjunctive queries with the operators of linear temporal logic *LTL*. When datalog (instead of *OWL 2 QL*) is used as a static ontology language, answering temporal conjunctive queries becomes easier (in terms of query rewriting algorithms); we refer to Section 5 for practical applications and further discussion.

2.2.1. Temporal *SPARQL*.

In the temporal OBDA framework we suggest in this paper, our aim is to employ as the query answering engine the OBDA platform Ontop that was designed for classical OBDA with *OWL 2 QL* (Rodriguez-Muro *et al.*, 2013; Calvanese *et al.*, 2017). In the non-temporal setting, Ontop essentially supports answering *SPARQL* queries under the *OWL 2 QL* direct semantics entailment regime over virtual RDF graphs populated by *R2RML* mappings and data instances stored in relational databases (Kontchakov *et al.*, 2014).

In the temporal setting, to be compatible with the available functionalities of Ontop, we suggest a query language that is an extension of *SPARQL* similar to *τ-SPARQL* proposed by Tappolet and Bernstein (2009). We remind the reader that variables in *SPARQL* are prefixed by '?' and that atoms take the form $?x$ a : GasTurbine (which stands for $\mathsf{GasTurbine}(x)$) and $?x$: isDeployedIn $?y$ (which stands for $\mathsf{isDeployedIn}(x, y)$). (Relations of arity higher than 2, such as ColocTempRotSensors mentioned above, are not supported directly in our language and have to be handled via *reification*; see, e.g., the work by Calvanese and De Giacomo (2003) and the references therein. Atoms like these are used for the static predicates from Σ_s (such as GasTurbine, isDeployedIn). Temporal predicates from Σ_t (such as PurgingIsOver) have to be followed by a suffix $@\langle ?e_1, ?v_1, ?v_2, ?e_2\rangle$, where $?e_1$ is a Boolean variable evaluating to either 'true' or 'false', depending on whether the interval where the predicate holds is left-closed or left-open (and similarly for $?e_2$ indicating right-closedness or right-openness), while $?v_1$ and $?v_2$ are variables over date/time whose values respectively indicate from when and until when the predicate holds.

Example 9. The conjunctive query $\boldsymbol{q}_1$ from Example 8 can be represented as the following temporal *SPARQL* query:

```
SELECT ?l ?begin ?end ?r
WHERE {
  {?ts a :MainFlameOn}@⟨?l, ?begin, ?end, ?r⟩
}
```

whereas the conjunctive query $\boldsymbol{q}_2$ is represented as

```
SELECT ?tb ?l ?begin ?end ?r
WHERE {
  ?tb a :GasTurbine.
  ?tb :isDeployedIn ss:train_tr05.
  {?tb a :PurgingIsOver}@⟨?l, ?begin, ?end, ?r⟩
}
```

As explained above, the certain answer to the query $\boldsymbol{q}_1$ with ontology $\mathcal{O} \cup \mathcal{R} \cup \mathcal{T}$, data instance D, as well as mappings $\mathcal{M}_s$ and $\mathcal{M}_t$, contains the tuple ($\mathtt{true}$, 12:32:40, 12:32:45, $\mathtt{false}$). ♦

2.2.2. On temporal RDF graphs.

There does not seem to exist a standardized way of representing temporal data as RDF graphs; we refer to a few relevant proposals by Gutiérrez *et al.* (2005), Tappolet and Bernstein (2009), and Grandi (2010). Although our temporal OBDA framework does not presuppose any materialization of relational data in the form of an RDF graph (which will be discussed in Section 3), we advocate the use of RDF datasets[7] comprising a distinguished graph and a set of named RDF graphs, following the model of RDF stream proposed by the W3C RDF Stream Processing Community Group.[8]

More specifically, to model temporal facts, for each relevant temporal interval we introduce a graph identifier and collect the triples that hold within this interval into the respective graph. The details of the interval (namely,

[7] https://www.w3.org/TR/rdf11-datasets/.

[8] http://streamreasoning.github.io/RSP-QL/RSP_Requirements_Design_Document/.

the beginning and the end) of the graph identifier are described in the distinguished graph using the vocabulary from the W3C TIME ontology (Cox and Little, 2017). The static triples also reside in the distinguished graph.

Example 10. The temporal fact

$$\mathsf{mainFlame}(\mathsf{mf01}, 2.3)@[12{:}32{:}30, 12{:}32{:}37)$$

from Example 7 can be represented as the named graph

$$\text{GRAPH } g_0 \ \{(\mathsf{mf01}, \mathsf{mainFlame}, 2.3)\}$$

and the distinguished graph containing the following triples [9]:

$(g_0, \mathsf{time{:}hasTime}, i_0)$,
$(i_0, \mathsf{a}, \mathsf{time{:}Interval})$,
$(i_0, \mathsf{time{:}isBeginningInclusive}, \mathtt{true})$,
$(i_0, \mathsf{time{:}isEndInclusive}, \mathtt{false})$,
$(i_0, \mathsf{time{:}hasBeginning}, b_0)$,
$(b_0, \mathsf{time{:}inXSDDateTimeStamp}$, '2017-06-06 12:32:30'$)$,
$(i_0, \mathsf{time{:}hasEnd}, e_0)$,
$(e_0, \mathsf{time{:}inXSDDateTimeStamp}$, '2017-06-06 12:32:37'$)$.

In the same way, the temporal fact

$$\mathsf{mainFlame}(\mathtt{mf01}, 2.3)@[12{:}32{:}37, 12{:}32{:}45)$$

can be modelled by a named graph g_1 and an interval i_1. Using the third rule of the program $\mathcal{T}$ from Example 4, we obtain the fact represented by the named graph

$$\text{GRAPH } g_2 \ \{(\mathsf{mf01}, \mathsf{a}, \mathsf{MainFlameUpTH})\}$$

and the distinguished graph

$(g_2, \mathsf{time{:}hasTime}, i_2)$,
$(i_2, \mathsf{a}, \mathsf{time{:}Interval})$,
$(i_2, \mathsf{time{:}isBeginningInclusive}, \mathtt{true})$,
$(i_2, \mathsf{time{:}isEndInclusive}, \mathtt{false})$,
$(i_2, \mathsf{time{:}hasBeginning}, b_2)$,
$(b_2, \mathsf{time{:}inXSDDateTimeStamp}$, '2017-06-06 12:32:30'$)$,
$(i_2, \mathsf{time{:}hasEnd}, e_2)$,
$(e_2, \mathsf{time{:}inXSDDateTimeStamp}$, '2017-06-06 12:32:45'$)$.

Further, the second rule in $\mathcal{T}$ gives us the named graph

$$\text{GRAPH } g_3 \ \{(\mathtt{mf01}, \mathsf{a}, \mathsf{MainFlameOn})\}$$

and the distinguished graph

$(g_3, \mathsf{time{:}hasTime}, i_3)$,
$(i_3, \mathsf{a}, \mathsf{time{:}Interval})$,
$(i_3, \mathsf{time{:}isBeginningInclusive}, \mathtt{true})$,
$(i_3, \mathsf{time{:}isEndInclusive}, \mathtt{false})$,
$(i_3, \mathsf{time{:}hasBeginning}, b_3)$,
$(b_3, \mathsf{time{:}inXSDDateTimeStamp}$, '2017-06-06 12:32:40'$)$,
$(i_2, \mathsf{time{:}hasEnd}, e_3)$,
$(e_3, \mathsf{time{:}inXSDDateTimeStamp}$, '2017-06-06 12:32:45'$)$.

[9]Note that we extended the Time ontology with the isBeginInclusive and isEndInclusive data properties, which are not currently supported.

In order to match this data, our query $\boldsymbol{q}_1$ from Example 9 has to be rewritten into the following *SPARQL* query with named graph variables:

```
SELECT ?l ?begin ?end ?r
WHERE {
  GRAPH ?g {?tb a :MainFlameOn.}
  ?g time:hasTime ?i.
  ?i a time:Interval;
    time:isBeginningInclusive ?l;
    time:hasBeginning
      [time:inXSDDateTimeStamp ?begin];
    time:hasEnd [time:inXSDDateTimeStamp ?end];
    time:isEndInclusive ?r.
}
```

This query returns the certain answer

$$(\mathtt{true}, 12{:}32{:}40, 12{:}32{:}45, \mathtt{false}).$$

In fact, this rewriting is similar to the one proposed by Tappolet and Bernstein (2009). ♦

3. System architecture and implementation in Ontop

In this section, we propose a system architecture of temporal OBDA by extending the OBDA platform Ontop. We first briefly describe the workflow of Ontop in the case of classical OBDA, and then discuss how to extend it to temporal OBDA.

3.1. Classical OBDA with Ontop. Ontop is a state-of-the-art OBDA system developed at the Free University of Bozen-Bolzano (Calvanese *et al.*, 2017). Ontop supports the standard W3C recommendations related to OBDA (such as *OWL 2 QL*, *R2RML*, *SPARQL*, and the *OWL 2 QL* entailment regime in *SPARQL*). The system is available as a Protégé plugin, and an extensible open-source Java library supporting OWL API and RDF4J.

The core of an OBDA system is the query answering algorithm. Ontop uses an optimized query rewriting algorithm (Rodriguez-Muro *et al.*, 2013) whose workflow is outlined as Algorithm 1. The algorithm takes as inputs an OBDA instance $\langle \mathcal{P}, D\rangle$ with $\mathcal{P} = \langle \mathcal{O}, \mathcal{M}_s, \mathcal{S}\rangle$ and a *SPARQL* query $\boldsymbol{q}$, and returns the certain answers to $\boldsymbol{q}$ over $\langle \mathcal{P}, D\rangle$. The workflow can be divided into the offline and online stages. During start-up (the offline stage), Ontop *(a)* classifies the static ontology $\mathcal{O}$ and *(b)* compiles the classified ontology into the input mapping $\mathcal{M}_s$, thus obtaining the saturated mapping $\mathcal{M}_s^{\mathcal{O}}$ known as the T-mapping (Rodriguez-Muro *et al.*, 2013).

During query execution (the online stage), Ontop transforms an input *SPARQL* query $\boldsymbol{q}$ into an optimized *SQL* query $\boldsymbol{q}_{opt}^{\mathcal{M},\mathcal{O}}$ exploiting the T-mapping $\mathcal{M}_s^{\mathcal{O}}$ and the database integrity constraints $\mathcal{S}$, and evaluates the generated *SQL* query over the database instance D.

Algorithm 1. Ontop workflow.

function *ontop*($\langle \mathcal{P}, D\rangle, \boldsymbol{q}$)
  let $\langle \mathcal{O}, \mathcal{M}_s, \mathcal{S}\rangle = \mathcal{P}$
  // offline
  $\mathcal{O}'$ = *classify*($\mathcal{O}$)
  $\mathcal{M}_s^{\mathcal{O}}$ = *saturate*($\mathcal{M}_s, \mathcal{O}'$)
  // online
  $\boldsymbol{q}^{\mathcal{M},\mathcal{O}}$ = *unfold*($\boldsymbol{q}, \mathcal{M}_s^{\mathcal{O}}$)
  $\boldsymbol{q}_{opt}^{\mathcal{M},\mathcal{O}}$ = *optimize*($\boldsymbol{q}^{\mathcal{M},\mathcal{O}}, S$)
  return *eval*($\boldsymbol{q}_{opt}^{\mathcal{M},\mathcal{O}}, D$)

3.2. Ontop-temporal. Now we present our proposal for the temporal extension of Ontop, which we call *Ontop-temporal*. Specifically, we discuss the choice of concrete languages for the additional input components and how to adapt the query rewriting algorithm.

3.2.1. Concrete languages. Our principle when choosing concrete languages for the inputs is to be compliant with the relevant existing standards whenever possible; we only extend and create new syntax/languages when it is absolutely necessary.

In Ontop-temporal, we continue to use *OWL 2 QL* for static ontologies and also allow the use of non-recursive datalog rules satisfying *(i)* and *(ii)* in Section 2.1.2. For temporal rules, there are no standard languages. The proposed concrete syntax for *datalog*$_{nr}$*MTL* is inspired by datalog, *SWRL*, and *SPARQL*.

We continue to use *R2RML* as a mapping language. Intuitively, an *R2RML* mapping produces named graphs (see Section 10) to represent temporal information. Named graphs are supported through the *R2RML* construct `rr:GraphMap`. Alternatively, considering that it is rather verbose to map all the temporal information in *R2RML*, we also extend the Ontop mapping language (Calvanese *et al*., 2017) to provide an alternative compact syntax close to the one used in Example 7.

As for the query language, we support both a τ-*SPARQL*-based language and plain *SPARQL* as discussed in Examples 9 and 10. Internally, the τ-*SPARQL*-based language is treated as syntactic sugar and handled by compiling it into the corresponding plain *SPARQL* query language.

3.2.2. Query answering algorithm. The algorithm of Ontop-temporal, outlined as Algorithm 2, takes as inputs a temporal OBDA instance $\langle \mathfrak{S}, D\rangle$ and a τ-*SPARQL* or *SPARQL* query $\boldsymbol{q}$, and returns the answers of $\boldsymbol{q}$ over $\langle \mathfrak{S}, D\rangle$.

Similarly to Ontop, the workflow of Ontop-temporal also consists of an offline stage, compiling $\mathfrak{S}$ into a set of mapping assertions $\mathcal{M}^{\mathfrak{S}}$, and an online stage, evaluating $\boldsymbol{q}$ over D by query rewriting. The offline stage has two more steps to process the temporal components in $\mathfrak{S}$:

(a) it saturates $\mathcal{M}_s^{\mathcal{O}}$ with static rules $\mathcal{R}$ as proposed by Xiao *et al*. (2014) and obtains the mapping $\mathcal{M}_s^{\mathcal{O},\mathcal{R}}$;

(b) it saturates $\mathcal{M}_t$ and $\mathcal{M}_s^{\mathcal{O},\mathcal{R}}$ with $\mathcal{T}$ using the algorithm by Brandt *et al.* (2017a; 2018), and obtains the final saturated mapping $\mathcal{M}^{\mathfrak{S}}$. In a nutshell, the algorithm computes a view for each predicate in a bottom-up fashion. The view definitions exploit *SQL* functions simulating the temporal operators in *MTL* and often result in complex *SQL* queries.

The online stage first converts the input query into *SPARQL* when it is expressed in τ-*SPARQL*. The optimization step also needs to be extended to handle temporal-specific constructs in *SQL* queries. We now present the *SQL* queries that we expect to be generated for the running example.

Algorithm 2. Proposed workflow for Ontop-temporal.

function *ontop_temporal*($\langle \mathfrak{S}, D\rangle, \boldsymbol{q}$)
  let $\langle \Sigma_s, \Sigma_t, \mathcal{M}_s, \mathcal{M}_t, \mathcal{O}, \mathcal{R}, \mathcal{T}, \mathcal{S}\rangle = \mathfrak{S}$
  // offline
  $\mathcal{O}'$ = *classify*($\mathcal{O}$)
  $\mathcal{M}_s^{\mathcal{O}}$ = *saturate*($\mathcal{M}_s, \mathcal{O}'$)
  $\mathcal{M}_s^{\mathcal{O},\mathcal{R}}$ = *saturate*($\mathcal{M}_s^{\mathcal{O}}, \mathcal{R}$)
  $\mathcal{M}^{\mathfrak{S}}$ = *saturate*($\mathcal{M}_t \cup \mathcal{M}_s^{\mathcal{O},\mathcal{R}}, \mathcal{T}$)
  // online
  if $\boldsymbol{q}$ is a $\tau-$\SPARQL$-$like query
    then $\boldsymbol{q}$ = *sparql*($\boldsymbol{q}$)
  $\boldsymbol{q}^{\mathfrak{S}}$ = *unfold*($\boldsymbol{q}, \mathcal{M}^{\mathfrak{S}}$)
  $\boldsymbol{q}_{opt}^{\mathfrak{S}}$ = *optimize*($\boldsymbol{q}^{\mathfrak{S}}, \mathcal{S}$)
  return *eval*($\boldsymbol{q}_{opt}^{\mathfrak{S}}, D$)

Example 11. The expected *SQL* translation of the temporal *SPARQL* query $\boldsymbol{q}_1$ from Example 9 is as follows:

```
CREATE TEMPORARY main_flame_up_th AS
(SELECT tb_measurement.s_id, begin, end
 FROM
   (SELECT s_id, value, tstmp AS  begin,
     LEAD(tstmp, 1) OVER
     (PARTITION BY s_id ORDER BY  tstmp) AS end
    FROM tb_measurement, tb_sensors
    WHERE tb_measurement.s_id = tb_sensors.s_id
          AND s_type = 1) SUBQ
 WHERE value >= 1.0 AND end IS NOT NULL);

SELECT s_id,
       (begin + interval '10␣seconds') AS begin,
       end
FROM CoalesceIntervals('main_flame_up_th',
                       's_id', 'begin', 'end')
WHERE end - begin > interval '10␣seconds';
```

The utility function/query `CoalesceIntervals` is defined in Appendix. This query takes a table (main_flame_up_th in this example) with a main column (s_id) and two columns (begin and end) representing the ends of the validity interval of an object in the main column. The returned table only contains the maximal validity intervals merging (coalescing) the overlapping intervals (see how the query answer is computed in Example 8). Note that the implementation of `CoalesceIntervals` in this example assumes that all intervals are of the form $[x, y)$.

In more complex queries (such as $\boldsymbol{q}_2$ from Example 9), we use an *SQL* query/function `TemporalJoin` that, given two tables T_1 and T_2 with tuples of the form (object_id, begin, end), returns a table with the intersection of the validity intervals for each object_id (see Appendix).

♦

4. Experimental evaluation

In order to show the feasibility of our approach, in this section, we present an experiment based on the running example of this paper. In the experiment, we manually computed the *SQL* queries produced by applying the *datalog$_{nr}$MTL* rewriting algorithm (Brandt *et al.*, 2017a) and we evaluate these queries over Siemens turbine data.

The initial data provided by Siemens is a sample for one running turbine over 4 days. We replicated this sample to imitate the data for one turbine over 10 different periods ranging from 32 to 319 months. We ran the experiments on an HP Proliant server with 2 Intel Xeon X5690 processors (each with 12 logical cores at 3.47 GHz), 106 GB of RAM and five 1 TB 15 K RPM hard disks. The data are stored on a PostgreSQL 9.6 database server.

We evaluated four queries $\mathsf{LowRotorSpeed}(x)@t$, $\mathsf{HighRotorSpeed}(x)@t$, $\mathsf{MainFlameOn}(x)@t$ and $\mathsf{PurgingIsOverFor1Tb}(x)@t$. The first three queries are based on the definitions in Section 2. Regarding the query: $\mathsf{PurgingIsOverFor1Tb}(\mathsf{tb0})@x$, since the available data contains measurements only for one turbine, we define it by simplifying PurgingIsOver as follows:

$$\begin{aligned}\mathsf{PurgingIsOverFor1Tb}(tb) \leftarrow\ & \mathsf{MainFlameOn}(tb),\\ & \diamondminus_{(0,10m]}(\boxminus_{(0,30s]}\mathsf{HighRotorSpeed}(tb),\\ & \quad \diamondminus_{(0,2m]}\boxminus_{(0,1m]}\mathsf{LowRotorSpeed}(tb)).\end{aligned}$$

The running times of these queries are shown in Fig. 2. As can be seen, they scale linearly. In particular the running times of $\mathsf{PurgingIsOverFor1Tb}(\mathsf{tb0})@x$, which contains all the other queries as subcomponents, provide an indication that our algorithm respects modularity.

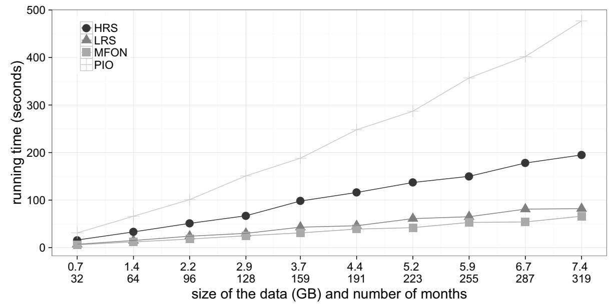

Fig. 2. Experiments with Siemens data over 32–319 months.

5. Related work

In this paper, our ontology language uses the operators of the metric temporal logic *MTL* that was originally introduced for modeling and reasoning about real-time systems (Koymans, 1990; Alur and Henzinger, 1993). Initial experimental results obtained by Brandt *et al.* (2018) demonstrated reasonably good performance and scalability on data from weather stations in the USA (of a size up to 8 GB) and turbine sensor data (of a size up to 6 GB). Another application of a similar formalism has been provided by El Raheb *et al.* (2017) in the context of querying user annotated ballet movement videos.

Practical ontology-mediated query answering with temporal ontologies based on the Halpern–Shoham interval temporal logic, $\mathcal{HS}$ (Halpern and Shoham, 1991) was investigated by Kontchakov *et al.* (2016). We remind the reader that $\mathcal{HS}$ is a classical propositional logic enriched with modal diamond operators of the form $\langle R\rangle$, where R is one of the twelve interval relations by Allen (1983): After, Begins, Ends, During, Later, Overlaps, and their inverses. One of the use-cases reported by Kontchakov *et al.* (2016) deals with historical data about the Italian public administration, and the other one with the weather data mentioned above. The proposed implementation, based on a reduction to standard datalog reasoning (with arithmetic constraints), showed feasibility of the approach with several state-of-the-art datalog engines. We note that, in the papers discussed above, datalog is used as a static ontology language and conjunctive queries are used as a query language.

Temporal relational databases have been studied intensively since the 1990s. Notably, the *TSQL2* query language was proposed by Snodgrass (1995) as a temporal extension of *SQL92*. Zimányi (2006) investigated temporal aggregations in temporal databases. Dignös *et al.* (2016) proposed a framework to implement temporal operators in a DBMS engine by extending its kernel. In this work, we do not assume that the underlying database supports additional temporal query language features (as those provided by *TSQL2*), and instead maintain compatibility with standard relational data sources. How to exploit in temporal OBDA additional features provided by temporal database query languages, when the underlying database supports them, is an interesting subject for future work.

6. Conclusions

In this paper, we have proposed a framework for practical temporal OBDA, defined its main components, and given a high level view of the system architecture.

As future work, we plan to formally present the working mechanism of the framework in detail, implement it as an extension of Ontop, and evaluate the implementation over the large scale heterogeneous Siemens use case data. Another future direction is to extend the framework in order to support semantic query answering over streaming data. There is an extensive body of work on semantic query answering over RDF streaming data (Barbieri *et al*., 2010; Calbimonte *et al*., 2012; Phuoc *et al*., 2011; Anicic *et al*., 2011; Beck *et al*., 2015; Özçep *et al*., 2014). However, none of these works follows the OBDA approach, apart from *STARQL* (Özçep *et al*., 2014), where one can define complex temporal patterns only at the query level rather than the ontology level as in our proposal. We plan to investigate how to incorporate the streaming data setting into our temporal OBDA framework.

Acknowledgment

This research has been partially supported by the Open Access Publishing Fund of the Free University of Bozen-Bolzano, by the projects OnProm, OBATS, and QUADRO, funded respectively through the 2015, 2017, and 2018 calls issued by the Research Committee of the Free University of Bozen-Bolzano, and by the Euregio IPN12 project KAOS funded through *European Region Tyrol-South Tyrol-Trentino* (EGTC) under the first call for basic research projects. The work of M. Zakharyaschev was carried out at the National Research University Higher School of Economics and supported by the Russian Science Foundation under the grant 17-11-01294.

This article is an extended and revised version of a paper that was published in the proceedings of the Workshop on *Novel Techniques for Integrating Big Data (BigNovelTI 2017)*, co-located with the 21st European Conference on *Advances in Databases and Information Systems (ADBIS 2017)* (Brandt *et al.*, 2017b).

References

Abiteboul, S., Hull, R. and Vianu, V. (1995). *Foundations of Databases*, Addison Wesley Publ. Co., Boston, MA.

Allen, J.F. (1983). Maintaining knowledge about temporal intervals, *Communications of the ACM* **26**(11): 832–843.

Alur, R. and Henzinger, T.A. (1993). Real-time logics: Complexity and expressiveness, *Information and Computation* **104**(1): 35–77.

Anicic, D., Fodor, P., Rudolph, S. and Stojanovic, N. (2011). EP-SPARQL: A unified language for event processing and stream reasoning, *Proceedings of the 20th International World Wide Web Conference (WWW), Hyderabad, India*, pp. 635–644.

Artale, A., Kontchakov, R., Kovtunova, A., Ryzhikov, V., Wolter, F. and Zakharyaschev, M. (2015a). First-order rewritability of temporal ontology-mediated queries, *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI), Buenos Aires, Argentina*, pp. 2706–2712.

Artale, A., Kontchakov, R., Ryzhikov, V. and Zakharyaschev, M. (2015b). Tractable interval temporal propositional and description logics, *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI), Austin, TX, USA*, pp. 1417–1423.

Artale, A., Kontchakov, R., Wolter, F. and Zakharyaschev, M. (2013). Temporal description logic for ontology-based data access, *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI), Beijing, China*, pp. 711–717.

Baader, F., Borgwardt, S., Koopmann, P., Ozaki, A. and Thost, V. (2017). Metric temporal description logics with interval-rigid names, *in* C. Dixon and M. Finger (Eds.), *Proceedings of the 11th International Symposium on Frontiers of Combining Systems (FroCoS)*, Lecture Notes in Computer Science, Vol. 10483, Springer, Berlin/Heidelberg, pp. 60–76.

Baader, F., Borgwardt, S. and Lippmann, M. (2013). Temporalizing ontology-based data access, *Proceedings of the 24th International Conference on Automated Deduction (CADE), Lake Placid, NY, USA*, pp. 330–344.

Baader, F., Calvanese, D., McGuinness, D., Nardi, D. and Patel-Schneider, P.F. (Eds.) (2007). *The Description Logic Handbook: Theory, Implementation and Applications*, 2nd Edn., Cambridge University Press, Cambridge.

Barbieri, D.F., Braga, D., Ceri, S., Valle, E.D. and Grossniklaus, M. (2010). C-SPARQL: A continuous query language for RDF data streams, *International Journal of Semantic Computing* **4**(1): 3–25.

Beck, H., Dao-Tran, M., Eiter, T. and Fink, M. (2015). LARS: A logic-based framework for analyzing reasoning over streams, *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI), Austin, TX, USA*, pp. 1431–1438.

Borgwardt, S., Lippmann, M. and Thost, V. (2013). Temporal query answering in the description logic DL-Lite, *in* P. Fontaine *et al.* (Eds.), *Proceedings of the 9th International Symposium on Frontiers of Combining Systems (FroCoS)*, Lecture Notes in Computer Science, Vol. 8152, Springer, Berlin/Heidelberg, pp. 165–180.

Brandt, S., Güzel Kalayci, E., Kontchakov, R., Ryzhikov, V., Xiao, G. and Zakharyaschev, M. (2017a). Ontology-based data access with a Horn fragment of metric temporal logic, *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI), San Francisco, CA, USA*, pp. 1070–1076.

Brandt, S., Güzel Kalayci, E., Ryzhikov, V., Xiao, G. and Zakharyaschev, M. (2017b). A framework for temporal ontology-based data access: A proposal, *in* M. Kirikova *et al.* (Eds.), *New Trends in Databases and Information Systems*, Communications in Computer and Information Science, Vol. 767, Springer, Berlin/Heidelberg, pp. 161–173.

Brandt, S., Güzel Kalayci, E., Ryzhikov, V., Xiao, G. and Zakharyaschev, M. (2018). Querying log data with metric temporal logic, *Journal of Artificial Intelligence Research* **62**: 829–877.

Calbimonte, J.-P., Jeung, H., Corcho, Ó. and Aberer, K. (2012). Enabling query technologies for the semantic sensor web, *International Journal on Semantic Web and Information Systems* **8**(1): 43–63.

Calvanese, D., Cogrel, B., Komla-Ebri, S., Kontchakov, R., Lanti, D., Rezk, M., Rodriguez-Muro, M. and Xiao, G. (2017). Ontop: Answering SPARQL queries over relational databases, *Semantic Web Journal* **8**(3): 471–487.

Calvanese, D. and De Giacomo, G. (2003). Expressive description logics, *in* F. Baader *et al.* (Eds.), *The Description Logic Handbook: Theory, Implementation, and Applications*, Cambridge University Press, Cambridge, pp. 178–218.

Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M. and Rosati, R. (2007). Tractable reasoning and efficient query answering in description logics: The DL-Lite family, *Journal of Automated Reasoning* **39**(3): 385–429.

Cox, S. and Little, C. (2017). Time ontology in OWL, *W3C Recommendation*, W3C, `https://www.w3.org/TR/owl-time/`.

Das, S., Sundara, S. and Cyganiak, R. (2012). R2RML: RDB to RDF mapping language, *W3C Recommendation*, W3C, `http://www.w3.org/TR/r2rml/`.

Dignös, A., Böhlen, M.H., Gamper, J. and Jensen, C.S. (2016). Extending the kernel of a relational DBMS with comprehensive support for sequenced temporal queries, *ACM Transactions on Database Systems* **41**(4): 26:1–26:46.

El Raheb, K., Mailis, T., Ryzhikov, V., Papapetrou, N. and Ioannidis, Y. E. (2017). BalOnSe: Temporal aspects of dance movement and its ontological representation, *in* E. Blomqvist *et al.*(Eds.), *Proceedings of the 14th Extended Semantic Web Conference (ESWC)*, Vol. 10250, Springer, Berlin/Heidelberg, pp. 49–64.

Grandi, F. (2010). T-SPARQL: A SQL2-like temporal query language for RDF, *Local Proceedings of the 14th East-European Conference on Advances in Databases and Information Systems (ADBIS), Novi Sad, Serbia*, pp. 21–30.

Gutiérrez-Basulto, V., Jung, J.C. and Kontchakov, R. (2016a). Temporalized EL ontologies for accessing temporal data: Complexity of atomic queries, *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI), New York, NY, USA*.

Gutiérrez-Basulto, V., Jung, J.C. and Ozaki, A. (2016b). On metric temporal description logics, *in* G.A. Kaminka *et al.* (Eds.), *Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI)*, Frontiers in Artificial Intelligence and Applications, Vol. 285, IOS Press, Amsterdam, pp. 837–845.

Gutiérrez-Basulto, V. and Klarman, S. (2012). Towards a unifying approach to representing and querying temporal data in description logics, *in* M. Krötzsch and U. Straccia (Eds.), *Proceedings of the 6th International Conference on Web Reasoning and Rule Systems (RR)*, Lecture Notes in Computer Science, Vol. 7497 Springer, Berlin/Heidelberg, pp. 90–105.

Gutiérrez, C., Hurtado, C.A. and Vaisman, A.A. (2005). Temporal RDF, *in* A. Gómez Pérez and J. Euzenat (Eds.), *Proceedings of the 2nd European Semantic Web Conference (ESWC)*, Lecture Notes in Computer Science, Vol. 3532, Springer, Berlin/ Heidelberg, pp. 93–107.

Halpern, J.Y. and Shoham, Y. (1991). A propositional modal logic of time intervals, *Journal of the ACM* **38**(4): 935–962.

Kharlamov, E., Brandt, S., Jiménez-Ruiz, E., Kotidis, Y., Lamparter, S., Mailis, T., Neuenstadt, C., Özçep, Ö. L., Pinkel, C., Svingos, C., Zheleznyakov, D., Horrocks, I., Ioannidis, Y.E. and Möller, R. (2016). Ontology-based integration of streaming and static relational data with Optique, *Proceedings of the 2016 International Conference on Management of Data (SIGMOD), San Francisco, CA, USA*, pp. 2109–2112.

Kharlamov, E., Mailis, T., Mehdi, G., Neuenstadt, C., Özçep, Ö. L., Roshchin, M., Solomakhina, N., Soylu, A., Svingos, C., Brandt, S., Giese, M., Ioannidis, Y.E., Lamparter, S., Möller, R., Kotidis, Y. and Waaler, A. (2017). Semantic access to streaming and static data at Siemens, *Journal of Web Semantics* **44**: 54–74.

Klarman, S. and Meyer, T. (2014). Querying temporal databases via OWL 2 QL, *in* R. Kontchakov and M.-L. Mugnier (Eds.), *Proceedings of the 8th International Conference on Web Reasoning and Rule Systems*, Lecture Notes in Computer Science, Vol. 8741, Springer, Berlin/Heidelberg, pp. 92–107.

Kontchakov, R., Pandolfo, L., Pulina, L., Ryzhikov, V. and Zakharyaschev, M. (2016). Temporal and spatial OBDA with many-dimensional Halpern–Shoham logic, *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI), New York, NY, USA*, pp. 1160–1166.

Kontchakov, R., Rezk, M., Rodriguez-Muro, M., Xiao, G. and Zakharyaschev, M. (2014). Answering SPARQL queries over databases under OWL 2 QL entailment regime, *in* P. Mika *et al.* (Eds.), *Proceedings of the 13th International Semantic Web Conference (ISWC)*, Lecture Notes in Computer Science, Vol. 8796, Springer, Berlin/Heidelberg, pp. 552–567.

Koymans, R. (1990). Specifying real-time properties with Metric Temporal Logic, *Real-Time Systems* **2**(4): 255–299.

Möller, R., Özçep, Ö. L., Neuenstadt, C., Zheleznyakov, D. and Kharlamov, E. (2013). A semantics for temporal and stream-based query answering in an OBDA context, *Optique Project Deliverable Deliverable D5.1*, FP7-318338.

Özçep, Ö.L., Möller, R. and Neuenstadt, C. (2014). A stream-temporal query language for ontology based data access, *in* C. Lutz and M. Thielscher (Eds.), *Proceedings of the 37th Annual German Conference on AI (KI)*, Lecture Notes in Computer Science, Vol. 8736, Springer, Berlin/Heidelberg, pp. 183–194.

Phuoc, D.L., Dao-Tran, M., Parreira, J.X. and Hauswirth, M. (2011). A native and adaptive approach for unified processing of linked streams and linked data, Lecture Notes in Computer Science, *in* L. Aroyo *et al.* (Eds.), *Proceedings of the 10th International Semantic Web Conference (ISWC)*, Springer, Berlin/Heidelberg, pp. 370–388.

Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M. and Rosati, R. (2008). Linking data to ontologies, *Journal on Data Semantics* **4900**: 133–173.

Rodriguez-Muro, M., Kontchakov, R. and Zakharyaschev, M. (2013). Ontology-based data access: Ontop of databases, *in* H. Alani *et al.* (Eds.), *Proceedings of the 12th International Semantic Web Conference (ISWC)*, Lecture Notes in Computer Science, Vol. 8218, Springer, Berlin/Heidelberg, pp. 558–573.

Snodgrass, R.T. (Ed.) (1995). *The TSQL2 Temporal Query Language*, Kluwer, Tucson, AZ.

Tappolet, J. and Bernstein, A. (2009). Applied temporal RDF: Efficient temporal querying of RDF data with SPARQL, *in* L. Aroyo *et al.* (Eds.), *ESWC 2009*, Lecture Notes in Computer Science, Vol. 5554, Springer, Berlin/Heidelberg, pp. 308–322.

Xiao, G., Calvanese, D., Kontchakov, R., Lembo, D., Poggi, A., Rosati, R. and Zakharyaschev, M. (2018). Ontology-based data access: A survey, *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI), Stockholm, Sweden*, pp. 5511–5519.

Xiao, G., Rezk, M., Rodriguez-Muro, M. and Calvanese, D. (2014). Rules and ontology based data access, *in* R. Kontchakov and M.-L. Mugnier (Eds.), *Proceedings of the 8th International Conference on Web Reasoning and Rules Systems (RR)*, Lecture Notes in Computer Science, Vol. 8741, Springer, Berlin/Heidelberg, pp. 157–172.

Zimányi, E. (2006). Temporal aggregates and temporal universal quantification in standard SQL, *SIGMOD Record* **35**(2): 16–21.

Elem Güzel Kalaycı is currently a PhD student of computer science at the KRDB Research Centre for Knowledge and Data, Free University of Bozen-Bolzano, Italy. Her main research interests are knowledge representation, temporal and stream reasoning, semantic web, and ontology-based data access.

Sebastian Brandt received his PhD in computer science from the Technical University of Dresden, Germany, in 2006. He is a senior key expert at Siemens Corporate Technology. He is an expert in semantic technologies, reasoning, big data analytics, and knowledge representation formalisms, with the emphasis on lightweight ontology languages. His work has enabled the OWL 2 language profile OWL 2 EL, a W3C standard.

Diego Calvanese is a full professor at the KRDB Research Centre for Knowledge and Data, Free University of Bozen-Bolzano, Italy. His research interests include formalisms for knowledge representation and reasoning, ontology languages, description logics, conceptual data modeling, data integration, graph data management, data-aware process verification, and service modeling and synthesis. He is an editor of *Description Logic Handbook*. He has been a fellow of the European Association for Artificial Intelligence (EurAI) since 2015.

Vladislav Ryzhikov is a lecturer in computer science at Birkbeck, University of London. He had received his BSc and MSc degrees at Ufa Aviation State University, and was awarded a PhD in Computer Science at the Free University of Bozen-Bolzano, Italy, in 2010. His major research areas are artificial intelligence and knowledge representation. He has worked in a number of projects related to ontology-based data access.

Guohui Xiao is an assistant professor at the KRDB Research Center for Knowledge and Data, Free University of Bozen-Bolzano, Italy. He received his PhD degree in computer science from the Vienna University of Technology, Austria, in 2014. He is currently leading the development within the Ontop team. His main research interests are knowledge representation, description logics, semantic web, database theory, ontology-based data access, and optimization and implementation of reasoning engines.

Michael Zakharyaschev is a professor of computer science at Birkbeck, University of London. He had received his BSc and MSc degrees at Moscow State University, and was then awarded a PhD and habilitation in mathematics at Novosibirsk State University. He has co-authored two major research monographs in modal logic. His current research areas are computational logic and its applications in knowledge representation, semantic technologies and, in particular, theory and practice of ontology-based data access.

Appendix
SQL functions (in plpgsql syntax) for temporal join and temporal interval coalesce

```
-- a temporal interval consists of an object identifier SID and two timestamps dFrom and dTo
CREATE TYPE TEMP_INTERVAL AS (SID VARCHAR(20), dFrom TIMESTAMP, dTo TIMESTAMP);

CREATE OR REPLACE FUNCTION TemporalJoin(t1 TEXT, t2 TEXT)
  RETURNS SETOF TEMP_INTERVAL AS
$BODY$
BEGIN
  SELECT
    t1.SID AS SID,
    CASE
      WHEN t1.dFrom > t2.dFrom AND t2.dTo > t1.dFrom      THEN t1.dFrom
      WHEN t2.dFrom > t1.dFrom AND t1.dTo > t2.dFrom      THEN t2.dFrom
      WHEN t1.dFrom = t2.dFrom   THEN t1.dFrom
    END AS dFrom,
    CASE
      WHEN t1.dTo < t2.dTo AND t1.dTo > t2.dFrom          THEN t1.dTo
      WHEN t2.dTo < t1.dTo AND t2.dTo > t1.dFrom          THEN t2.dTo
      WHEN t1.dTo = t2.dTo                                THEN t1.dTo
    END   AS dTo
  FROM t1, t2
  WHERE
    t1.SID = t2.SID AND
        ((t1.dFrom > t2.dFrom AND t2.dTo > t1.dFrom) OR
                             (t2.dFrom > t1.dFrom AND t1.dTo > t2.dFrom) OR
                             (t1.dFrom = t2.dFrom))
        AND ((t1.dTo < t2.dTo AND t1.dTo > t2.dFrom) OR
             (t2.dTo < t1.dTo AND t2.dTo > t1.dFrom) OR (t1.dTo = t2.dTo));
  RETURN;
END
$BODY$ LANGUAGE 'plpgsql';

-- we assume that the table is already ordered by SID, and then begin and end columns
CREATE OR REPLACE FUNCTION CoalesceInterval(
    tableName TEXT, objectColumn TEXT, beginColumn TEXT, endColumn TEXT)
  RETURNS SETOF TEMP_INTERVAL AS $BODY$
DECLARE
  r      TEMP_INTERVAL%ROWTYPE;
  SID    VARCHAR(20) :='';
  dFrom  TIMESTAMP := NULL;
  dTo    TIMESTAMP := NULL;

BEGIN
  FOR r IN EXECUTE 'SELECT␣' || objectColumn || '␣AS␣SID,␣' || beginColumn || '␣AS␣dFrom,␣'
                  || endColumn || '␣AS␣dTo' || '␣FROM␣' || tableName
  LOOP
    IF (SID = '' AND dFrom IS NULL AND dTo IS NULL) THEN -- initialization
      SID := r.SID; dFrom := r.dFrom; dTo := r.dTo;
    ELSIF (r.SID <> SID AND SID <> '' AND dFrom IS NOT NULL AND dTo IS NOT NULL) THEN
      RETURN NEXT (SID, dFrom, dTo);
      SID := r.SID; dFrom := r.dFrom; dTo := r.dTo;
    ELSIF r.SID = SID THEN
      IF r.dFrom >= dFrom AND r.dFrom <= dTo THEN
        IF r.dTo >= dTo THEN dTo := r.dTo; END IF;
      ELSIF r.dFrom > dTo THEN
        RETURN NEXT (SID, dFrom, dTo);
        SID = r.SID; dFrom = r.dFrom; dTo = r.dTo;
      END IF;
    END IF;
  END LOOP;
  IF (SID <> '' AND dFrom IS NOT NULL AND dTo IS NOT NULL) THEN
    RETURN NEXT (SID, dFrom, dTo);
  END IF;
  RETURN;
END
$BODY$ LANGUAGE 'plpgsql';
```

Received: 5 February 2018
Revised: 17 June 2018
Accepted: 22 September 2018