Fakultät für Informatik
Facoltà di Scienze e Tecnologie informatiche
Faculty of Computer Science

A Framework for Temporal Ontology-based Data Access: a Proposal

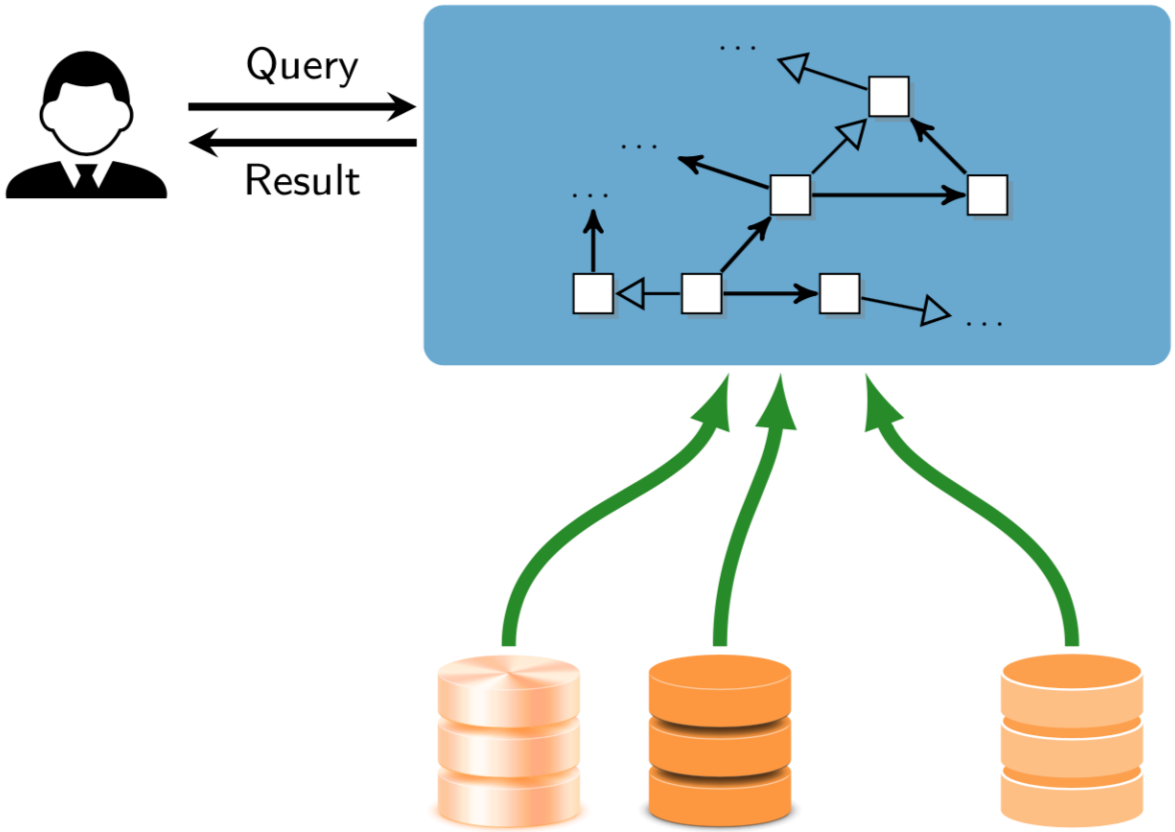
Sebastian Brandt, Elem Güzel Kalaycı, Vladislav Ryzhikov, Guohui Xiao, and Michael Zakharyaschev

Faculty of Computer Science, Free University of Bozen-Bolzano, Italy

September 24, 2017

- Ontology-based Data Access
- A Framework of Temporal OBDA
- Practical Considerations
- Conclusion and Future Work

Outline



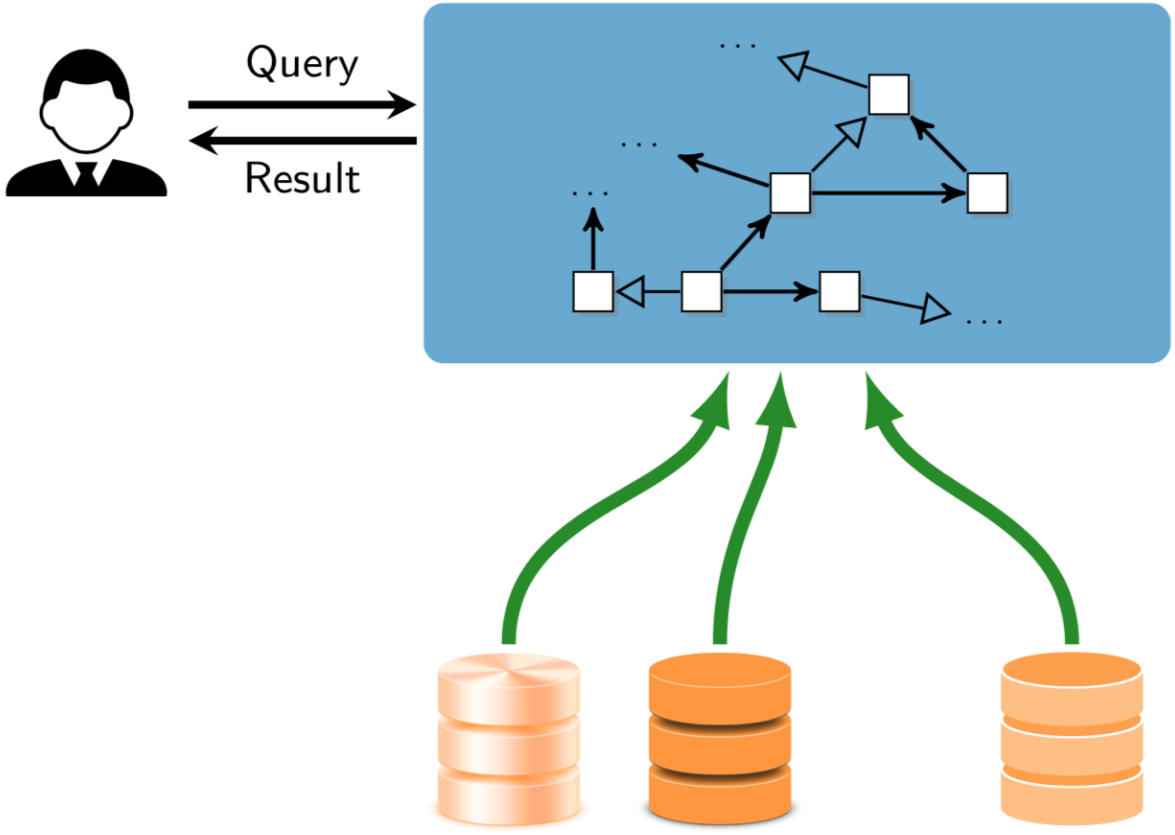
Ontology/Knowledge Graph
provides global vocabulary and conceptual view

Mappings
how to populate the ontology from the data

Data Sources
external and heterogeneous

Logical transparency in accessing data:

- does not know where and how the data is stored.
- can only see a conceptual view of the data.



Ontology/Knowledge Graph
provides global vocabulary and conceptual view

Mappings
how to populate the ontology from the data

Data Sources
external and heterogeneous

Logical transparency in accessing data:

- does not know where and how the data is stored.
- can only see a conceptual view of the data.

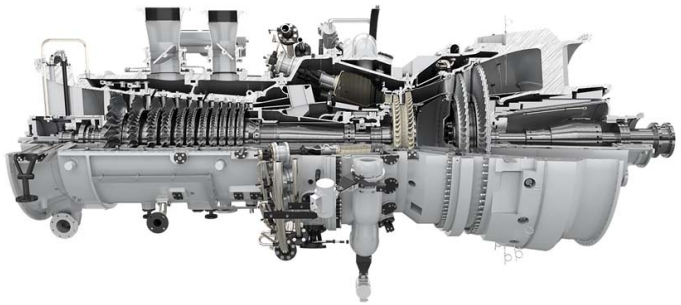
Most of the existing works in OBDA uses only on static data, or only supported limited temporal reasoning capabilities.

Siemens Use Case

Monitoring gas and steam turbines

Collects data from 50 remote diagnostic centres

Centres are linked to a **common DB centre**

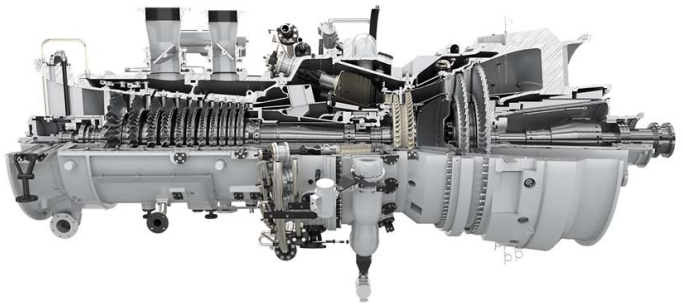


Siemens Use Case

Monitoring gas and steam turbines

Collects data from 50 remote diagnostic centres

Centres are linked to a **common DB centre**



Retrospective diagnostics (detect **abnormal** or **potentially dangerous** events)

Events involve a number of **measurements from sensors**,

have a certain **temporal duration**,

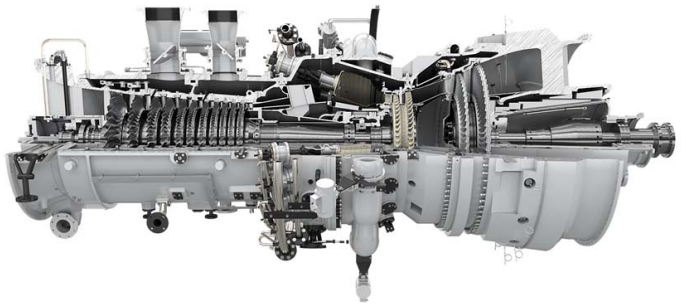
occurring in a certain **temporal sequence**.

Siemens Use Case

Monitoring gas and steam turbines

Collects data from 50 remote diagnostic centres

Centres are linked to a **common DB centre**



Retrospective diagnostics (detect **abnormal** or **potentially dangerous** events)

Events involve a number of **measurements from sensors**,

have a certain **temporal duration**,

occurring in a certain **temporal sequence**.

'find the **gas turbines** deployed in the train with the ID *T001* and **time periods** of their **accomplished purgings**'.

Outline

Proposed Temporal OBDA Framework

A traditional *OBDA* ***specification*** is a **triple** $\mathfrak{S} = \langle \mathcal{O}, \mathfrak{M}, \mathcal{S} \rangle$.

- $\mathcal{O}$ is an ontology,
- $\mathfrak{M}$ a set of mapping assertions between ontological entities and data sources,
- $\mathcal{S}$ a database schema.

Proposed Temporal OBDA Framework

A traditional *OBDA* ***specification*** is a **triple** $\mathfrak{S} = \langle \mathcal{O}, \mathfrak{M}, \mathcal{S} \rangle$.

- $\mathcal{O}$ is an ontology,
- $\mathfrak{M}$ a set of mapping assertions between ontological entities and data sources,
- $\mathcal{S}$ a database schema.

Temporal OBDA **on top of** the **traditional** OBDA

Proposed Temporal OBDA Framework

A traditional *OBDA* ***specification*** is a **triple** $\mathfrak{S} = \langle \mathcal{O}, \mathfrak{M}, \mathcal{S} \rangle$.

- $\mathcal{O}$ is an ontology,
- $\mathfrak{M}$ a set of mapping assertions between ontological entities and data sources,
- $\mathcal{S}$ a database schema.

Temporal OBDA **on top of** the **traditional** OBDA

Proposed Temporal OBDA Framework

A traditional *OBDA* ***specification*** is a **triple** $\mathfrak{S} = \langle \mathcal{O}, \mathfrak{M}, \mathcal{S} \rangle$.

- $\mathcal{O}$ is an ontology,
- $\mathfrak{M}$ a set of mapping assertions between ontological entities and data sources,
- $\mathcal{S}$ a database schema.

Temporal OBDA **on top of** the **traditional** OBDA

Our temporal OBDA ***specification*** is a **tuple**

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

- where Σ_s (Σ_t) is a **static** (respectively, **temporal**) vocabulary,
- $\mathfrak{M}_s$ ($\mathfrak{M}_t$) a set of **static** (respectively, **temporal**) mapping assertions,
- $\mathcal{O}$ an **ontology**,
- $\mathcal{R}$ ($\mathcal{T}$) a set of **static** (respectively, **temporal**) rules,
- $\mathcal{S}$ is a database **schema**.

Static Ontology

At **Siemens**, devices are monitored by many different kinds of **sensors** (temperature, pressure, vibration etc.).

Static Ontology

At **Siemens**, devices are monitored by many different kinds of **sensors** (temperature, pressure, vibration etc.).

An **ontology** for modeling the **static knowledge** about

- machines and their deployment profiles, sensor configurations,
- component hierarchies, and functional profiles.

Static Ontology

At **Siemens**, devices are monitored by many different kinds of **sensors** (temperature, pressure, vibration etc.).

An **ontology** for modeling the **static knowledge** about

- machines and their deployment profiles, sensor configurations,
- component hierarchies, and functional profiles.

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

We use *DL-Lite*$_{\mathcal{R}}$ (OWL 2 QL) as our static ontology language.

Static Rules

However, ***DL-Lite***$_{\mathcal{R}}$ is not able to capture all the **static knowledge** required in the **Siemens** use case.
We complement this ontology with **nonrecursive datalog** *static rules*.

Static Rules

However, ***DL-Lite***$_{\mathcal{R}}$ is not able to capture all the **static knowledge** required in the **Siemens** use case.
We complement this ontology with **nonrecursive datalog** *static rules*.

Turbine parts monitored by different co-located sensors (e.g. temperature, rotation speed).

$$\begin{aligned}\text{ColocSensors}(tb, ts, rs) \leftarrow\ & \text{isMonitoredBy}(pt, ts), \text{TemperatureSensor}(ts),\\ & \text{isMonitoredBy}(pt, rs), \text{RotationSpeedSensor}(rs),\\ & \text{isPartOf}(pt, tb), \text{Turbine}(tb).\end{aligned}$$

Temporal Rules

Siemens is interested in,

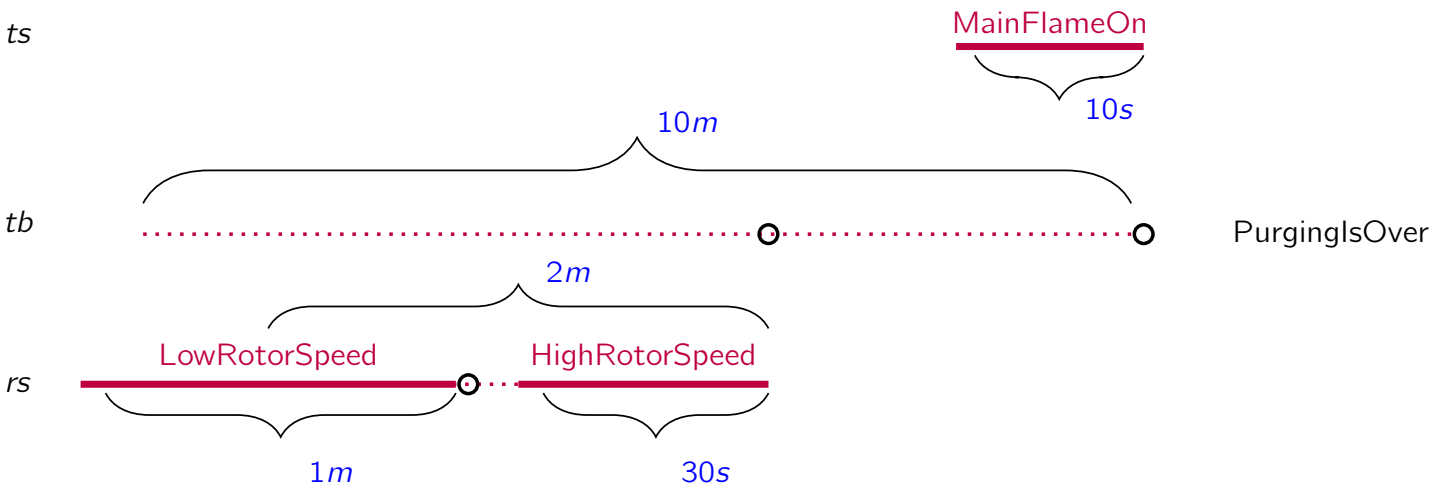
- detecting **abnormal situations**
- monitoring **running tasks**.

Temporal Rules

Siemens is interested in,

- detecting **abnormal situations**
- monitoring **running tasks**.

Purging is Over is a complex event of a certain *turbine*.

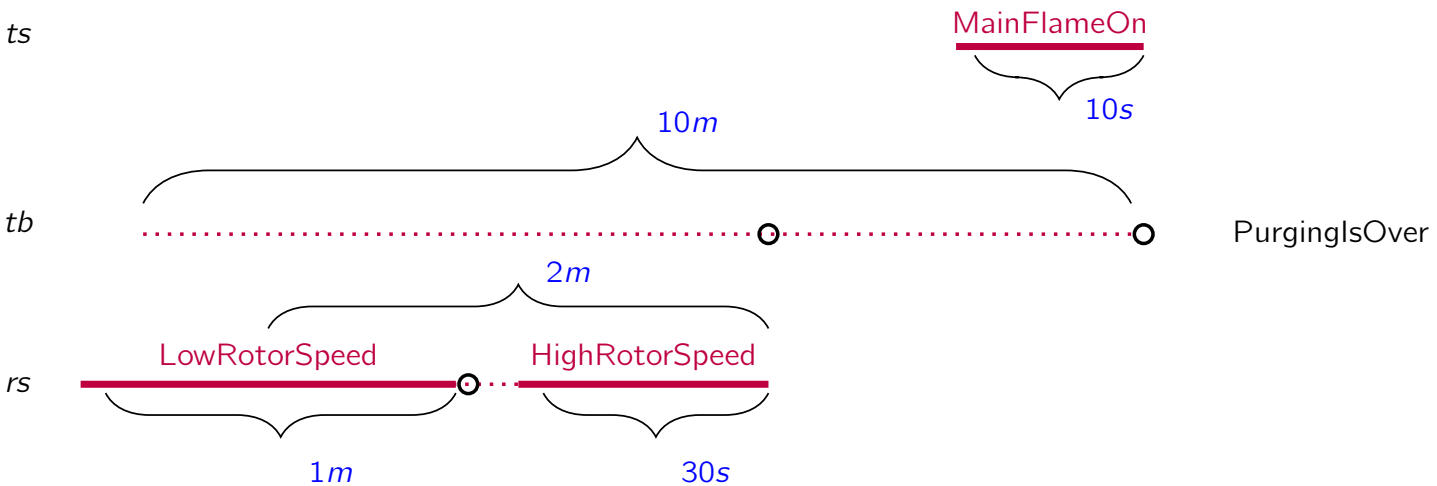


Temporal Rules

Siemens is interested in,

- detecting **abnormal situations**
- monitoring **running tasks**.

Purging is Over is a complex event of a certain *turbine*.



$$\text{PurgingIsOver}(tb) \leftarrow \boxminus_{[0s,10s]} \text{MainFlameOn}(ts),$$
$$\diamondminus_{(0,10m]}\big[\boxminus_{(0,30s]} \text{HighRotorSpeed}(rs), \diamondminus_{(0,2m]} \boxminus_{(0,1m]} \text{LowRotorSpeed}(rs)\big],$$
$$\text{ColocTempRotSensors}(tb, ts, rs).$$
$$\text{HighRotorSpeed}(tb) \leftarrow \text{rotorSpeed}(tb, v), v > 1260.$$
$$\text{LowRotorSpeed}(tb) \leftarrow \text{rotorSpeed}(tb, v), v < 1000.$$

A Quick Recap of datalogMTL

datalogMTL is a Horn fragment of **Metric Temporal Logic (MTL)**.

A **datalogMTL** *program* is a finite set of ***rules*** of the form

$$A^+ \leftarrow A_1 \wedge \cdots \wedge A_k \quad \text{or} \quad \bot \leftarrow A_1 \wedge \cdots \wedge A_k,$$

where $k \geq 1$, each A_i is either a comparison expression (e.g. $\tau \neq \tau'$, $\tau \leq \tau'$) or defined by the grammar

$$A \ ::= \ P(\tau_1, \ldots, \tau_m) \ \mid \ \boxplus_\varrho A \ \mid \ \boxminus_\varrho A \ \mid \ \diamondplus_\varrho A \ \mid \ \diamondminus_\varrho A$$

A^+ does not contain $\diamondplus_\varrho$ or $\diamondminus_\varrho$.

A Quick Recap of datalogMTL

datalogMTL is a Horn fragment of **Metric Temporal Logic (MTL)**.

A **datalogMTL** *program* is a finite set of ***rules*** of the form

$$A^+ \leftarrow A_1 \wedge \cdots \wedge A_k \quad \text{or} \quad \bot \leftarrow A_1 \wedge \cdots \wedge A_k,$$

where $k \geq 1$, each A_i is either a comparison expression (e.g. $\tau \neq \tau'$, $\tau \leq \tau'$) or defined by the grammar

$$A \ ::= \ P(\tau_1, \ldots, \tau_m) \ \mid \ \boxplus_\varrho A \ \mid \ \boxminus_\varrho A \ \mid \ \diamondplus_\varrho A \ \mid \ \diamondminus_\varrho A$$

A^+ does not contain $\diamondplus_\varrho$ or $\diamondminus_\varrho$.

$$\boxminus_{[0,3h]}\mathsf{Blizzard}(v) \leftarrow \boxminus_{[0,3h]}\mathsf{StrongWind}(v) \wedge \boxminus_{[0,3h]}\mathsf{LowVisibility}(v) \wedge \boxminus_{[0,3h]}\mathsf{Snow}(v)$$

A Quick Recap of datalogMTL

datalogMTL is a Horn fragment of **Metric Temporal Logic (MTL)**.

A **datalogMTL** *program* is a finite set of ***rules*** of the form

$$A^+ \leftarrow A_1 \wedge \cdots \wedge A_k \quad \text{or} \quad \bot \leftarrow A_1 \wedge \cdots \wedge A_k,$$

where $k \geq 1$, each A_i is either a comparison expression (e.g. $\tau \neq \tau'$, $\tau \leq \tau'$) or defined by the grammar

$$A \ ::= \ P(\tau_1, \ldots, \tau_m) \ \mid \ \boxplus_\varrho A \ \mid \ \boxminus_\varrho A \ \mid \ \diamondplus_\varrho A \ \mid \ \diamondminus_\varrho A$$

A^+ does not contain $\diamondplus_\varrho$ or $\diamondminus_\varrho$.

$$\boxminus_{[0,3h]}\text{Blizzard}(v) \leftarrow \boxminus_{[0,3h]}\text{StrongWind}(v) \wedge \boxminus_{[0,3h]}\text{LowVisibility}(v) \wedge \boxminus_{[0,3h]}\text{Snow}(v)$$

- **Data complexity** of query answering with **datalog$_{nr}$MTL** is AC^0.
- query answering with **datalog$_{nr}$MTL** can be reduced to SQL
- Suitable as a temporal rule language for OBDA

Schema and Data

An example data schema $\mathcal{S}$ for the Siemens data, including sensor measurements and deployment details:

$$\text{tb\_measurement}(\underline{\text{timestamp}}, \underline{\text{sensor\_id}}, \text{value}),$$
$$\text{tb\_sensors}(\underline{\text{sensor\_id}}, \text{sensor\_type}, \text{mnted\_part}, \text{mnted\_tb}),$$
$$\text{tb\_deployment}(\underline{\text{turbine\_id}}, \text{turbine\_type}, \text{deployed\_in}),$$
$$\text{tb\_components}(\underline{\text{turbine\_id}}, \underline{\text{component\_id}}, \text{component\_type}).$$

tb_measurement		
timestamp	sensor_id	value
2017-06-06 12:20:00	rs01	570
2017-06-06 12:21:00	rs01	680
2017-06-06 12:21:30	rs01	920
2017-06-06 12:22:50	rs01	1278
2017-06-06 12:23:40	rs01	1310
	...	
2017-06-06 12:32:30	mf01	2.3
2017-06-06 12:32:50	mf01	1.8
2017-06-06 12:33:40	mf01	0.9
	...	

tb_sensors			
sensor_id	sensor_type	mnted_part	mnted_tb
rs01	0	pt01	tb01
mf01	1	b01	tb01
	...	...	

tb_components		
turbine_id	component_id	component_type
tb01	pt01	0
tb01	b01	1
	...	

Static Mapping Assertions

Static mapping assertions: $\phi(\vec{x}) \rightsquigarrow \psi(\vec{x})$

where $\phi(\vec{x})$ is a **query** over the **schema** $\mathcal{S}$,

$\psi(\vec{x})$ is an atom in Σ_s and **variables** $\vec{x}$

Static Mapping Assertions

Static mapping assertions: $\phi(\vec{x}) \rightsquigarrow \psi(\vec{x})$

where $\phi(\vec{x})$ is a **query** over the **schema** $\mathcal{S}$,

$\psi(\vec{x})$ is an atom in Σ_s and **variables** $\vec{x}$

$$\texttt{SELECT sensor\_id AS X FROM tb\_sensors WHERE sensor\_type} = 1 \rightsquigarrow \text{TemperatureSensor}(\texttt{X})$$

$$\texttt{SELECT component\_id AS X FROM tb\_components WHERE component\_type} = 1 \rightsquigarrow \text{Burner}(\texttt{X})$$

$$\texttt{SELECT mnted\_part AS X, sensor\_id AS Y FROM tb\_sensors} \rightsquigarrow \text{isMonitoredBy}(\texttt{X}, \texttt{Y})$$

Static Mapping Assertions

Static mapping assertions: $\phi(\vec{x}) \rightsquigarrow \psi(\vec{x})$

where $\phi(\vec{x})$ is a **query** over the **schema** $\mathcal{S}$,

$\psi(\vec{x})$ is an atom in Σ_s and **variables** $\vec{x}$

$$\texttt{SELECT sensor\_id AS X FROM tb\_sensors WHERE sensor\_type} = 1 \rightsquigarrow \text{TemperatureSensor}(\texttt{X})$$

$$\texttt{SELECT component\_id AS X FROM tb\_components WHERE component\_type} = 1 \rightsquigarrow \text{Burner}(\texttt{X})$$

$$\texttt{SELECT mnted\_part AS X, sensor\_id AS Y FROM tb\_sensors} \rightsquigarrow \text{isMonitoredBy}(\texttt{X}, \texttt{Y})$$

We retrieve the following **facts**:

Burner(b01), TemperatureSensor(mf01),

isMonitoredBy(pt01, rs01), isMonitoredBy(b01, mf01).

Static Mapping Assertions

Static mapping assertions: $\phi(\vec{x}) \rightsquigarrow \psi(\vec{x})$

where $\phi(\vec{x})$ is a **query** over the **schema** $\mathcal{S}$,

$\psi(\vec{x})$ is an atom in Σ_s and **variables** $\vec{x}$

$$\texttt{SELECT sensor\_id AS X FROM tb\_sensors WHERE sensor\_type} = 1 \rightsquigarrow \text{TemperatureSensor}(\texttt{X})$$

$$\texttt{SELECT component\_id AS X FROM tb\_components WHERE component\_type} = 1 \rightsquigarrow \text{Burner}(\texttt{X})$$

$$\texttt{SELECT mnted\_part AS X, sensor\_id AS Y FROM tb\_sensors} \rightsquigarrow \text{isMonitoredBy}(\texttt{X}, \texttt{Y})$$

We retrieve the following **facts**:

Burner(b01), TemperatureSensor(mf01),

isMonitoredBy(pt01, rs01), isMonitoredBy(b01, mf01).

$\mathfrak{M}_s$ is the set of ***static*** mapping assertions.

Temporal Mapping Assertions

Temporal mapping assertions:

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

`begin` and `end` are mapped to values of the **date/time** format

'$\langle$' is either '(' or '[', (and similarly for '$\rangle$').

$\psi(\vec{x})$ is an atom in Σ_t and **variables** $\vec{x}$

Temporal Mapping Assertions

Temporal mapping assertions:

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

`begin` and `end` are mapped to values of the **date/time** format
'$\langle$' is either '(' or '[', (and similarly for '$\rangle$').
$\psi(\vec{x})$ is an atom in Σ_t and **variables** $\vec{x}$

```
SELECT * FROM (
    SELECT sensor_id, value,timestamp AS begin, LEAD(timestamp, 1) OVER W AS end
    FROM tb_measurement, tb_sensors
    WINDOW W AS (PARTITION BY sensor_id ORDER BY timestamp)
    WHERE tb_measurement.sensor_id = tb_sensors.sensor_id AND sensor_type = 0
) SUBQ WHERE value > 1260
```

$\rightsquigarrow$ HighRotorSpeed(sensor_id)@[begin, end)

Temporal Mapping Assertions

Temporal mapping assertions:

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

`begin` and `end` are mapped to values of the **date/time** format
'$\langle$' is either '(' or '[', (and similarly for '$\rangle$').
$\psi(\vec{x})$ is an atom in Σ_t and **variables** $\vec{x}$

```
SELECT * FROM (
    SELECT sensor_id, value,timestamp AS begin, LEAD(timestamp, 1) OVER W AS end
    FROM tb_measurement, tb_sensors
    WINDOW W AS (PARTITION BY sensor_id ORDER BY timestamp)
    WHERE tb_measurement.sensor_id = tb_sensors.sensor_id AND sensor_type = 0
) SUBQ WHERE value > 1260
```

$\rightsquigarrow$ HighRotorSpeed(`sensor_id`)@[`begin`, `end`)

We retrieve **temporal facts** from the database:

HighRotorSpeed(`rs01`)@[2017-06-06 12:22:50, 2017-06-06 12:23:40)

Temporal Mapping Assertions

Temporal mapping assertions:

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

`begin` and `end` are mapped to values of the **date/time** format
'$\langle$' is either '(' or '[', (and similarly for '$\rangle$').
$\psi(\vec{x})$ is an atom in Σ_t and **variables** $\vec{x}$

```
SELECT * FROM (
    SELECT sensor_id, value,timestamp AS begin, LEAD(timestamp, 1) OVER W AS end
    FROM tb_measurement, tb_sensors
    WINDOW W AS (PARTITION BY sensor_id ORDER BY timestamp)
    WHERE tb_measurement.sensor_id = tb_sensors.sensor_id AND sensor_type = 0
) SUBQ WHERE value > 1260
```

$\rightsquigarrow$ HighRotorSpeed(`sensor_id`)@[`begin`, `end`)

We retrieve **temporal facts** from the database:

HighRotorSpeed(`rs01`)@[2017-06-06 12:22:50, 2017-06-06 12:23:40)

We denote **sets** of such mapping assertions by $\mathfrak{M}_t$.

Temporal OBDA Specification (Recap)

A Temporal OBDA ***specification*** is the tuple

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

- where Σ_s (Σ_t) is a **static** (respectively, **temporal**) vocabulary,
- $\mathfrak{M}_s$ ($\mathfrak{M}_t$) a set of **static** (respectively, **temporal**) mapping assertions,
- $\mathcal{O}$ an **ontology**,
- $\mathcal{R}$ ($\mathcal{T}$) a set of **static** (respectively, **temporal**) rules,
- $\mathcal{S}$ is a database **schema**.

Temporal OBDA Instance

A temporal OBDA ***instance*** $\mathfrak{I}$ is a pair $\langle \mathfrak{S}, D \rangle$

- $\mathfrak{S}$ is a temporal OBDA **specification**,
- D is a **database instance** compliant with the **database schema** $\mathcal{S}$ in $\mathfrak{S}$.

Temporal OBDA Instance

A temporal OBDA ***instance*** $\mathfrak{I}$ is a pair $\langle \mathfrak{S}, D \rangle$

- $\mathfrak{S}$ is a temporal OBDA **specification**,
- D is a **database instance** compliant with the **database schema** $\mathcal{S}$ in $\mathfrak{S}$.

We advocate the use **RDF Datasets**, following the model proposed by the W3C **RSP Community Group**.

Temporal OBDA Instance

A temporal OBDA ***instance*** $\mathfrak{I}$ is a pair $\langle \mathfrak{S}, D \rangle$

- $\mathfrak{S}$ is a temporal OBDA **specification**,
- D is a **database instance** compliant with the **database schema** $\mathcal{S}$ in $\mathfrak{S}$.

We advocate the use **RDF Datasets**, following the model proposed by the W3C **RSP Community Group**.

The **temporal fact**

$$\text{HighRotorSpeed}(\texttt{rs01})@[\text{2017-06-06 12:22:50, 2017-06-06 12:23:40})$$

is modeled as the **named graph**

$$\text{GRAPH } g_0 \ \{(\texttt{rs01}, a, \text{HighRotorSpeed})\}$$

and the following set of triples in the **default graph**:

$(g_0, a, \text{time:Interval})$,
$(g_0, \text{time:isBeginningInclusive}, \text{true}), (g_0, \text{time:isEndInclusive}, \text{false})$,
$(g_0, \text{time:hasBeginning}, b_0), (b_0, \text{time:inXSDDateTimeStamp}, \text{'2017-06-06 12:22:50'})$,
$(g_0, \text{time:hasEnd}, e_0), (e_0, \text{time:inXSDDateTimeStamp}, \text{'2017-06-06 12:23:40'})$.

Query Answering

'find the **gas turbines** deployed in the **train** with the ID *T001* and **time periods** of their **accomplished purgings**'.

Query Answering

'find the **gas turbines** deployed in the **train** with the ID ***T001*** and **time periods** of their **accomplished purgings**'.

We ask for the query $Q(tb)@\iota$, where $Q(tb)$ is defined as

$$\begin{aligned} \mathsf{Q}(tb) \leftarrow\ & \mathsf{Train}(\text{“}T001\text{”}), \mathsf{GasTurbine}(tb), \\ & \mathsf{isDeployedIn}(\mathsf{tb}, \text{“}T001\text{”}), \mathsf{PurgingIsOver}(tb). \end{aligned}$$

Outline

Concrete Syntax

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

Ontop Mapping Syntax

```
[PrefixDeclaration]
st:             http://siemens.com/temporal/ns#
xsd:    http://www.w3.org/2001/XMLSchema#
obda:   https://w3id.org/obda/vocabulary#

[MappingDeclaration] @collection [[
mappingId   mappingRSA1260
target      st:HighRotorSpeed/{sensor_id} obda:hasStatus st:HighRotorSpeed.
interval    [ {begin}^^xsd:dateTimeStamp, {end}^^xsd:dateTimeStamp )
source      SELECT sensor_id, begin, end FROM (
    SELECT sensor_id, value, timestamp AS begin,
    LEAD(timestamp, 1) OVER W AS end
    FROM MEASUREMENT
    WINDOW W AS (PARTITION BY sensor_id ORDER BY timestamp)) SUBQ
    WHERE value > 1260
]]
```

Query Language

'find the **gas turbines** deployed in the **train** with the ID ***T001*** and **time periods** of their **accomplished purgings**'.

```
PREFIX ss: <http://siemens.com/ns#>
PREFIX st: <http://siemens.com/temporal/ns#>
PREFIX obda: <https://w3id.org/obda/vocabulary#>

SELECT ?tb ?left_edge ?begin ?end ?right_edge WHERE {
?tb a ss:GasTurbine ; ss:isDeployedIn ss:train_T001 .

{?tb a st:PurgingIsOver}@<?left_edge,?begin,?end,?right_edge>
}
```

Equivalent SPARQL Query

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX time: <http://www.w3.org/2006/time#>
PREFIX ss: <http://siemens.com/ns#>
PREFIX st: <http://siemens.com/temporal/ns#>
PREFIX obda: <https://w3id.org/obda/vocabulary#>

SELECT ?tb ?left_edge ?begin ?end ?right_edge WHERE {
  ?tb a ss:GasTurbine ; ss:isDeployedIn ss:train_T001 .

  GRAPH ?g { ?tb a st:PurgingIsOver.}

  ?g a time:Interval ;
     time:isBeginningInclusive ?left_edge  ;
     time:hasBeginning [ time:inXSDDateTimeStamp ?begin ] ;
     time:hasEnd [ time:inXSDDateTimeStamp ?end ] ;
     time:isEndInclusive ?right_edge .
}
```

Outline

Future Work

- Query rewriting algorithm
- Implementation as an extension of Ontop
- Experiments with **real world** data (hopefully in Siemens)
- Extend the framework for **streaming** data

THANKS