Ontology-based Access to Temporal Data

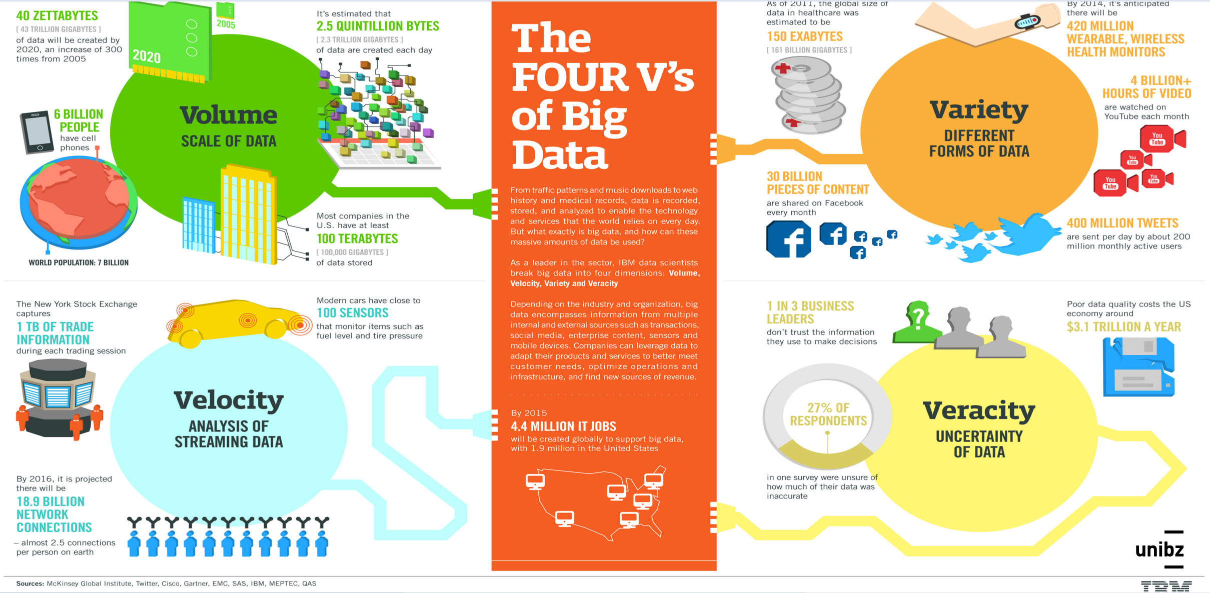
Diego Calvanese

KRDB Research Centre for Knowledge and Data
Free University of Bozen-Bolzano, Italy

RuleML Webinar

Bolzano, Italy, 22 February 2019

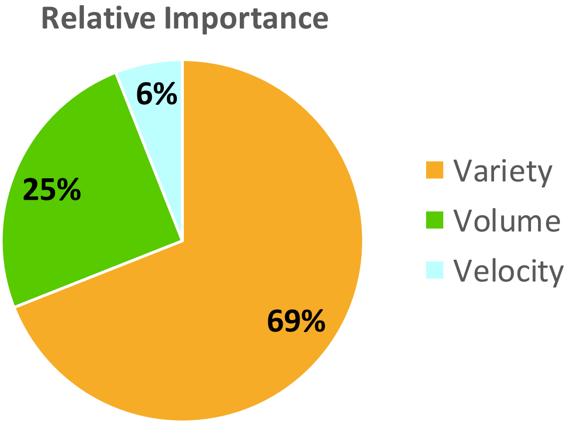
Challenges in the Big Data era



unibz

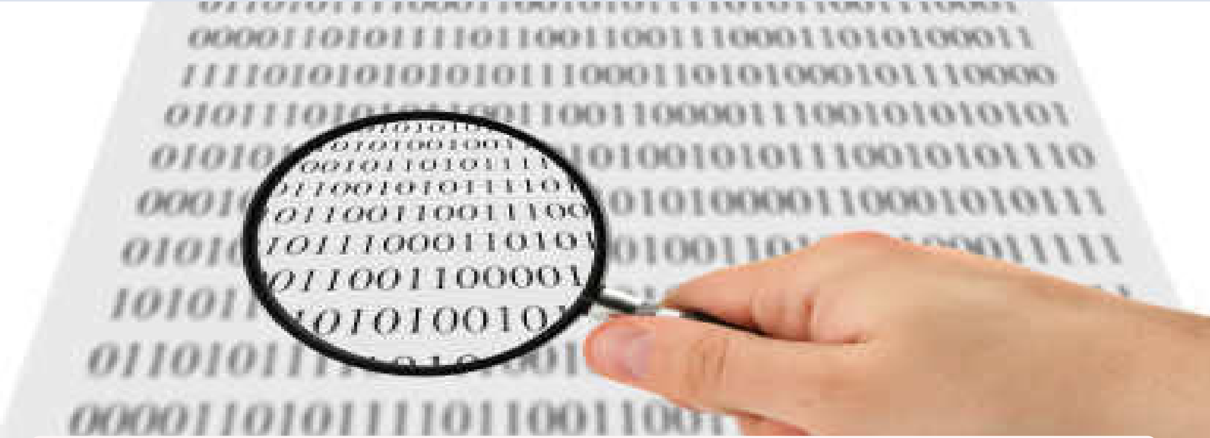
Variety, not volume, is driving Big Data initiatives

MIT Sloan Management Review (28 March 2016)



http://sloanreview.mit.edu/article/variety-not-volume-is-driving-big-data-initiatives/

How much time is spent searching for the right data?



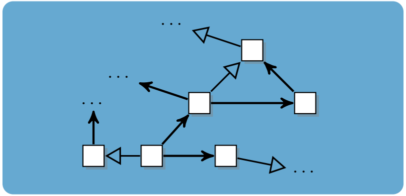
Important problem: searching for data and establishing its quality

Example: in oil&gas, engineers spend 30–70% of their time on this (Crompton, 2008)

Solution: Ontology-based data access (OBDA)

Data Sources $\mathcal{S}$
autonomous and heterogeneous

Solution: Ontology-based data access (OBDA)



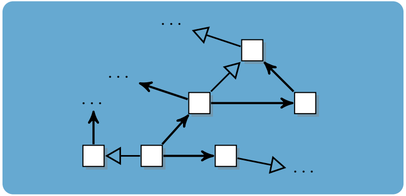
Ontology $\mathcal{O}$
conceptual view of data, convenient vocabulary

Data Sources $\mathcal{S}$
autonomous and heterogeneous

unibz

Solution: Ontology-based data access (OBDA)

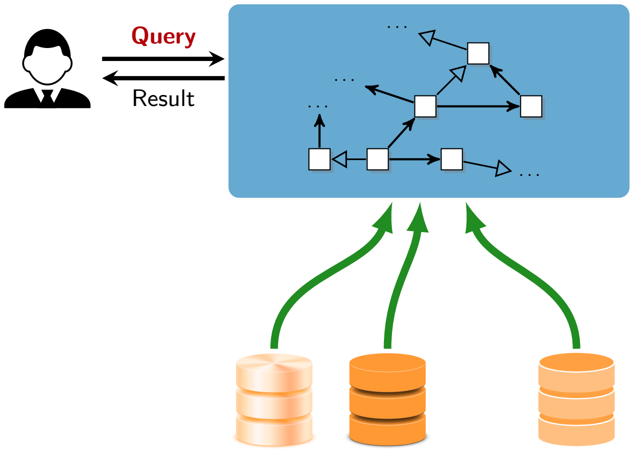
Query



Ontology $\mathcal{O}$
conceptual view of data,
convenient vocabulary

Data Sources $\mathcal{S}$
autonomous and
heterogeneous

Solution: Ontology-based data access (OBDA)



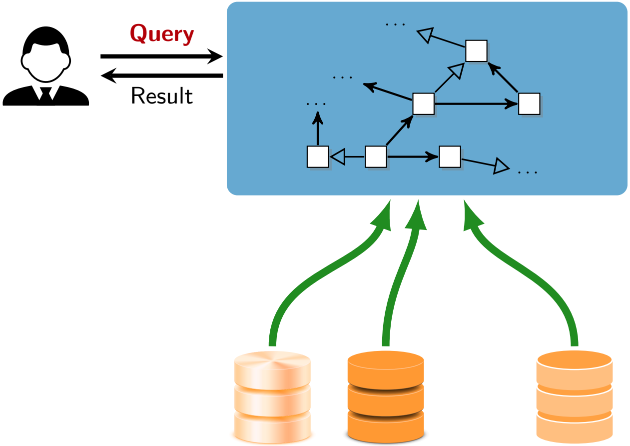
Ontology $\mathcal{O}$
conceptual view of data, convenient vocabulary

Mapping $\mathcal{M}$
how to populate the ontology from the data

Data Sources $\mathcal{S}$
autonomous and heterogeneous

unibz

Solution: Ontology-based data access (OBDA)



Ontology $\mathcal{O}$
conceptual view of data, convenient vocabulary

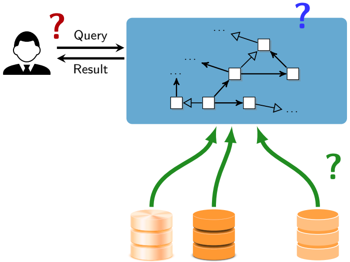
Mapping $\mathcal{M}$
how to populate the ontology from the data

Data Sources $\mathcal{S}$
autonomous and heterogeneous

Reduces the time for translating information needs into queries from days to minutes.

OBDA framework – Which languages to use?

The choice of the right languages needs to take into account the tradeoff between expressive power and efficiency of query answering.



Note: We are in a setting where data plays a prominent role, so **efficiency with respect to the data** is the key factor.

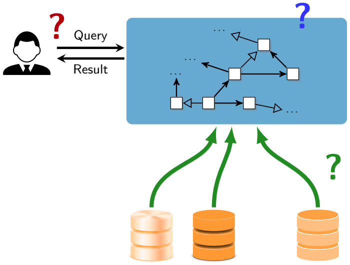
The W3C has standardized languages that are suitable for OBDA:

1. Ontology $\mathcal{O}$: expressed in **OWL 2 QL** [W3C Rec. 2012]
2. Query: expressed in **SPARQL** [W3C Rec. 2013] (v1.1)
3. Mapping $\mathcal{M}$: expressed in **R2RML** [W3C Rec. 2012]

OBDA framework – Which languages to use?

The choice of the right languages needs to take into account the tradeoff between expressive power and efficiency of query answering.

Note: We are in a setting where data plays a prominent role, so **efficiency with respect to the data** is the key factor.



The W3C has standardized languages that are suitable for OBDA:

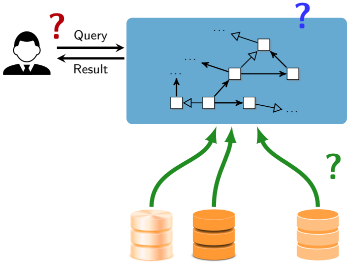
1. Ontology $\mathcal{O}$: expressed in **OWL 2 QL** [W3C Rec. 2012]
2. Query: expressed in **SPARQL** [W3C Rec. 2013] (v1.1)
3. Mapping $\mathcal{M}$: expressed in **R2RML** [W3C Rec. 2012]

unibz

OBDA framework – Which languages to use?

The choice of the right languages needs to take into account the tradeoff between expressive power and efficiency of query answering.

Note: We are in a setting where data plays a prominent role, so **efficiency with respect to the data** is the key factor.

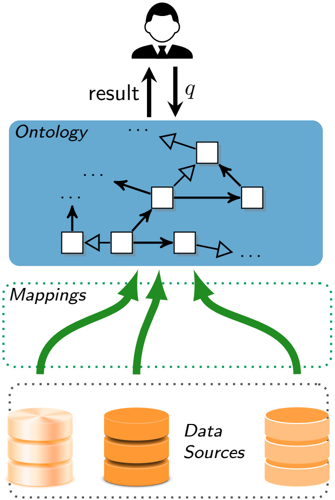


The W3C has standardized languages that are suitable for OBDA:

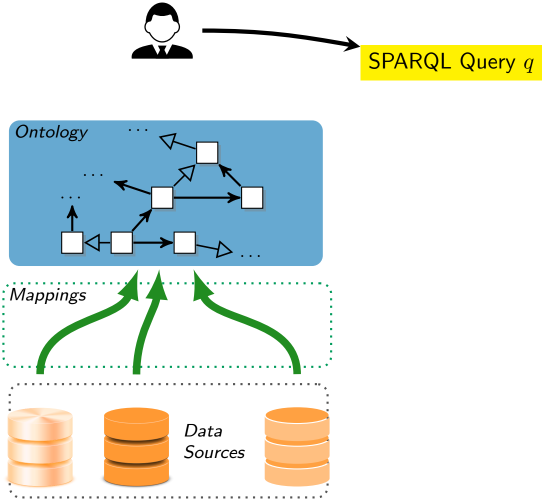
1. Ontology $\mathcal{O}$: expressed in **OWL 2 QL** [W3C Rec. 2012]
2. Query: expressed in **SPARQL** [W3C Rec. 2013] (v1.1)
3. Mapping $\mathcal{M}$: expressed in **R2RML** [W3C Rec. 2012]

unibz

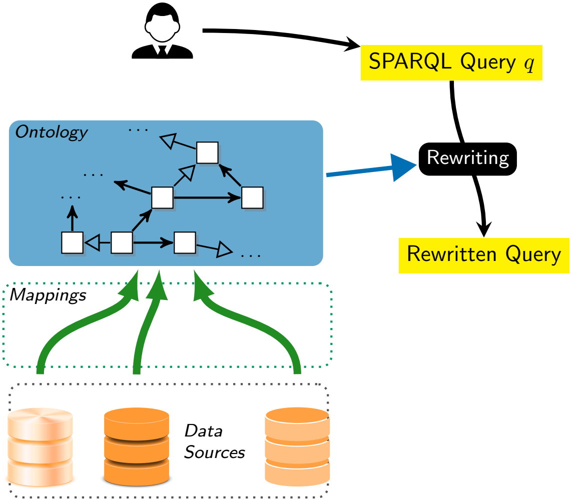
Query answering by query rewriting



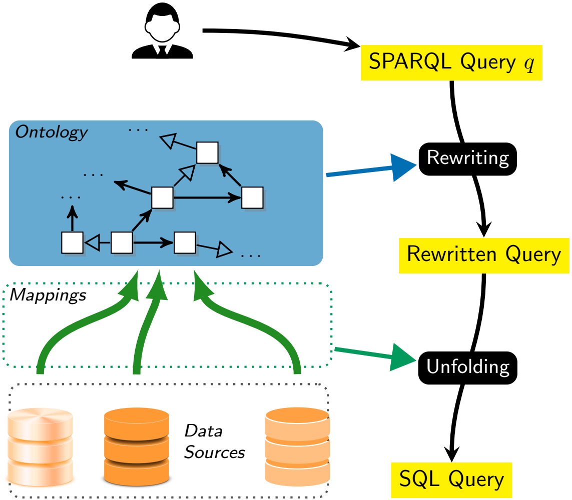
Query answering by query rewriting



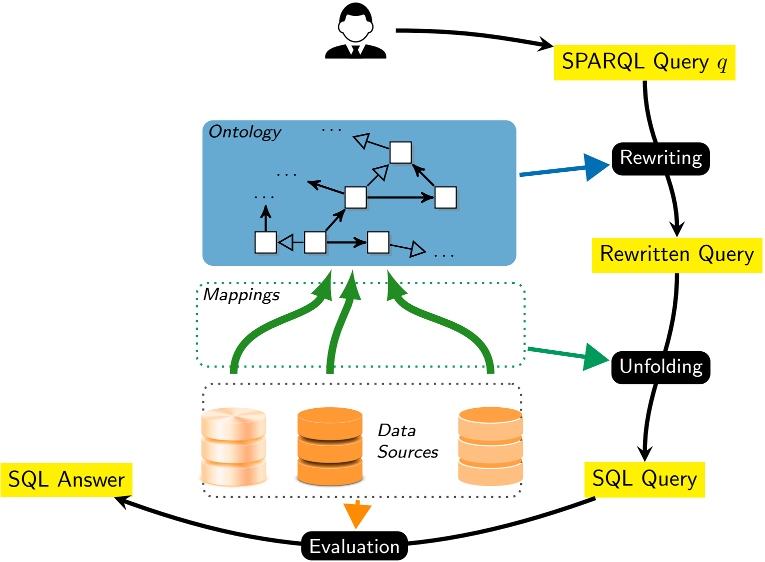
Query answering by query rewriting



Query answering by query rewriting

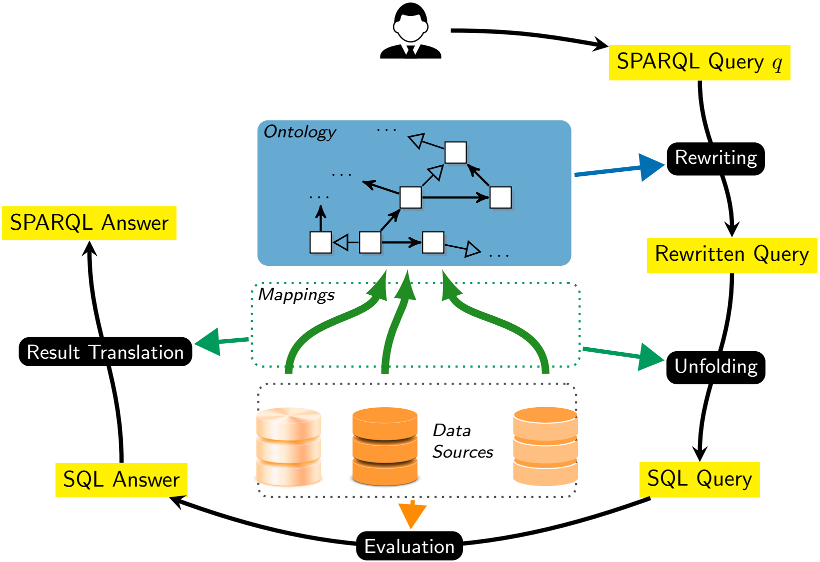


Query answering by query rewriting

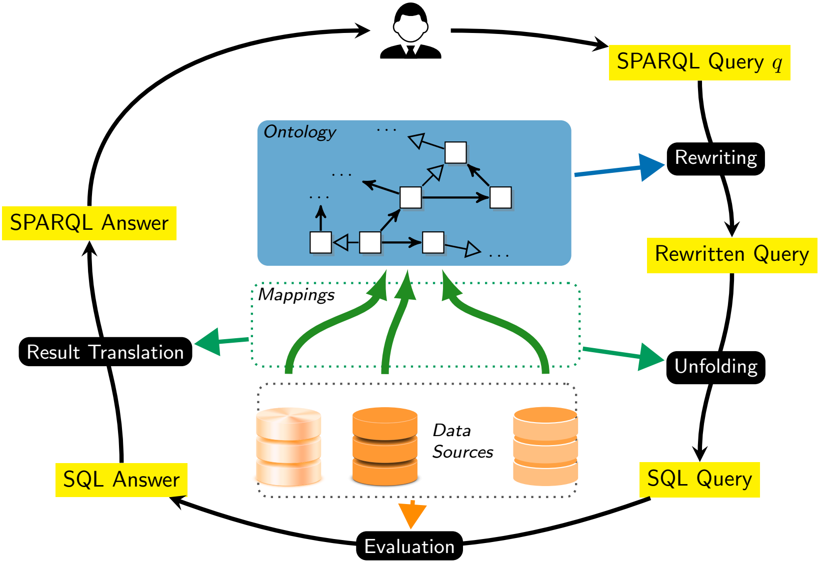


unibz

Query answering by query rewriting



Query answering by query rewriting

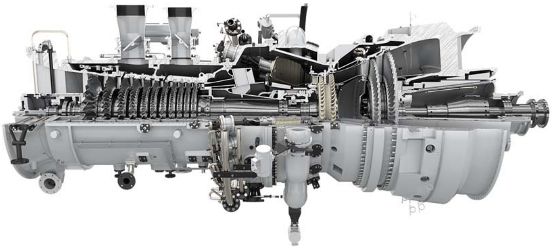


Outline

1. Motivation
2. Temporal OBDA Framework
3. Ontology Layer
4. Mapping Layer
5. Query Answering for Temporal OBDA
6. Summary and Future Work

Siemens Energy Services

- Monitor gas and steam turbines.
- Collect data from 50 remote diagnostic centers around the world.
- Centers linked to a common central DB.
- Turbines are highly complex, with 5 000–50 000 sensors each.
- Engineers compute KPIs, and extract data from maintenance reports using ETL tools.



Objective: retrospective diagnostics

i.e., detect abnormal or potentially dangerous events.

Events

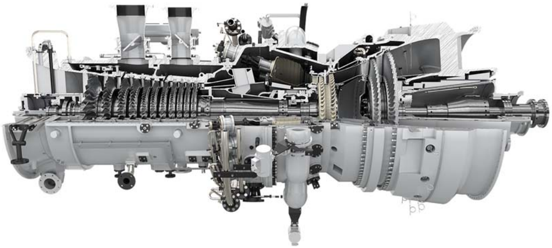
- Involve a number of sensor measurements.
- Have a certain temporal duration.
- Occur in a certain temporal sequence.

Example request

Find the gas turbines deployed in the train with ID T001, and the time periods of their accomplished purgings.

Siemens Energy Services

- Monitor gas and steam turbines.
- Collect data from 50 remote diagnostic centers around the world.
- Centers linked to a common central DB.
- Turbines are highly complex, with 5 000–50 000 sensors each.
- Engineers compute KPIs, and extract data from maintenance reports using ETL tools.



Objective: retrospective diagnostics

i.e., detect abnormal or potentially dangerous events.

Events

- Involve a number of sensor measurements.
- Have a certain temporal duration.
- Occur in a certain temporal sequence.

Example request

Find the gas turbines deployed in the train with ID T001, and the time periods of their accomplished purgings.

To capture such a complex scenario . . .

. . . we need to **enrich OBDA with temporal features**.

Approaches proposed in the literature:

1. Use standard ontologies and extend queries with temporal operators

[Gutiérrez-Basulto and Klarman 2012; Baader, Borgwardt, and Lippmann 2013; Klarman and Meyer 2014; Özçep and Möller 2014; Kharlamov et al. 2016]

However:

- Query language gets significantly more complicated.
- Effort is shifted from design time to query time.

2. Extend both query and ontology with linear temporal logic (LTL) operators

[Artale, Kontchakov, Wolter, et al. 2013; Artale, Kontchakov, Kovtunova, et al. 2015]

However:

- LTL is not suited to deal with metric temporal information.

To capture such a complex scenario . . .

. . . we need to **enrich OBDA with temporal features**.

Approaches proposed in the literature:

1. Use standard ontologies and extend queries with temporal operators

[Gutiérrez-Basulto and Klarman 2012; Baader, Borgwardt, and Lippmann 2013; Klarman and Meyer 2014; Özçep and Möller 2014; Kharlamov et al. 2016]

However:

- Query language gets significantly more complicated.
- Effort is shifted from design time to query time.

2. Extend both query and ontology with linear temporal logic (LTL) operators

[Artale, Kontchakov, Wolter, et al. 2013; Artale, Kontchakov, Kovtunova, et al. 2015]

However:

- LTL is not suited to deal with metric temporal information.

To capture such a complex scenario . . .

. . . we need to **enrich OBDA with temporal features**.

Approaches proposed in the literature:

1. Use standard ontologies and extend queries with temporal operators

[Gutiérrez-Basulto and Klarman 2012; Baader, Borgwardt, and Lippmann 2013; Klarman and Meyer 2014; Özçep and Möller 2014; Kharlamov et al. 2016]

However:

- Query language gets significantly more complicated.
- Effort is shifted from design time to query time.

2. Extend both query and ontology with linear temporal logic (LTL) operators

[Artale, Kontchakov, Wolter, et al. 2013; Artale, Kontchakov, Kovtunova, et al. 2015]

However:

- LTL is not suited to deal with metric temporal information.

Outline

unibz

We propose a different approach to temporal OBDA

- At the ontology level, we have both **static** and **temporal predicates**:
  - Static predicates to represent ordinary facts.
    E.g., Burner(b01), isMonitoredBy(b01, mf01)
  - Temporal predicates to represent temporal facts with a validity interval
    E.g., HighRotorSpeed(rs01)@[2017-06-06 12:22:50, 2017-06-06 12:23:40)

    We consider both open and closed intervals:
    $A(d)@(t_1, t_2)$, $A(d)@[t_1, t_2)$, $A(d)@(t_1, t_2]$, $A(d)@[t_1, t_2]$
- The ontology is expressed in OWL 2 QL $\rightsquigarrow$ First-order rewritability.
- We enrich it with static and temporal rules.
- We **extend the mapping mechanism** so as to retrieve also temporal information from the data, i.e., both static and temporal facts.

unibz

We propose a different approach to temporal OBDA

- At the ontology level, we have both **static** and **temporal predicates**:
  - Static predicates to represent ordinary facts.
    E.g., Burner(b01), isMonitoredBy(b01, mf01)
  - Temporal predicates to represent temporal facts with a validity interval
    E.g., HighRotorSpeed(rs01)@[2017-06-06 12:22:50, 2017-06-06 12:23:40)
    We consider both open and closed intervals:
    $A(d)@(t_1, t_2)$, $A(d)@[t_1, t_2)$, $A(d)@(t_1, t_2]$, $A(d)@[t_1, t_2]$
- The ontology is expressed in OWL 2 QL $\rightsquigarrow$ First-order rewritability.
- We enrich it with static and temporal rules.
- We **extend the mapping mechanism** so as to retrieve also temporal information from the data, i.e., both static and temporal facts.

unibz

Formal framework for temporal OBDA

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

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

Temporal OBDA builds on traditional OBDA.

A **temporal OBDA specification** is a tuple $\mathcal{P}_t = \langle \Sigma_s, \Sigma_t, \mathcal{O}, \mathcal{R}_s, \mathcal{R}_t, \mathcal{M}_s, \mathcal{M}_t, \mathcal{S} \rangle$

- Σ_s is a static vocabulary.
- $\mathcal{O}$ is an ontology.
- $\mathcal{R}_s$ is a set of static rules.
- $\mathcal{M}_s$ is a set of static mapping assertions.
- $\mathcal{S}$ is a database schema.
- Σ_t is a temporal vocabulary.
- $\mathcal{R}_t$ is a set of temporal rules.
- $\mathcal{M}_t$ is a set of temporal mapping assertions.

Formal framework for temporal OBDA

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

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

Temporal OBDA builds on traditional OBDA.

A **temporal OBDA specification** is a tuple $\mathcal{P}_t = \langle \Sigma_s, \Sigma_t, \mathcal{O}, \mathcal{R}_s, \mathcal{R}_t, \mathcal{M}_s, \mathcal{M}_t, \mathcal{S} \rangle$

- Σ_s is a static vocabulary.
- Σ_t is a temporal vocabulary.
- $\mathcal{O}$ is an ontology.
- $\mathcal{R}_s$ is a set of static rules.
- $\mathcal{R}_t$ is a set of temporal rules.
- $\mathcal{M}_s$ is a set of static mapping assertions.
- $\mathcal{M}_t$ is a set of temporal mapping assertions.
- $\mathcal{S}$ is a database schema.

Formal framework for temporal OBDA

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

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

Temporal OBDA builds on traditional OBDA.

A **temporal OBDA specification** is a tuple $\mathcal{P}_t = \langle \Sigma_s, \Sigma_t, \mathcal{O}, \mathcal{R}_s, \mathcal{R}_t, \mathcal{M}_s, \mathcal{M}_t, \mathcal{S} \rangle$

- Σ_s is a static vocabulary.
- Σ_t is a temporal vocabulary.
- $\mathcal{O}$ is an ontology.
- $\mathcal{R}_s$ is a set of static rules.
- $\mathcal{R}_t$ is a set of temporal rules.
- $\mathcal{M}_s$ is a set of static mapping assertions.
- $\mathcal{M}_t$ is a set of temporal mapping assertions.
- $\mathcal{S}$ is a database schema.

Formal framework for temporal OBDA

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

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

Temporal OBDA builds on traditional OBDA.

A **temporal OBDA specification** is a tuple $\mathcal{P}_t = \langle \Sigma_s, \Sigma_t, \mathcal{O}, \mathcal{R}_s, \mathcal{R}_t, \mathcal{M}_s, \mathcal{M}_t, \mathcal{S} \rangle$

- Σ_s is a static vocabulary.
- Σ_t is a temporal vocabulary.
- $\mathcal{O}$ is an ontology.
- $\mathcal{R}_s$ is a set of static rules.
- $\mathcal{R}_t$ is a set of temporal rules.
- $\mathcal{M}_s$ is a set of static mapping assertions.
- $\mathcal{M}_t$ is a set of temporal mapping assertions.
- $\mathcal{S}$ is a database schema.

Formal framework for temporal OBDA

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

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

Temporal OBDA builds on traditional OBDA.

A **temporal OBDA specification** is a tuple $\mathcal{P}_t = \langle \Sigma_s, \Sigma_t, \mathcal{O}, \mathcal{R}_s, \mathcal{R}_t, \mathcal{M}_s, \mathcal{M}_t, \mathcal{S} \rangle$

- Σ_s is a static vocabulary.
- Σ_t is a temporal vocabulary.
- $\mathcal{O}$ is an ontology.
- $\mathcal{R}_s$ is a set of static rules.
- $\mathcal{R}_t$ is a set of temporal rules.
- $\mathcal{M}_s$ is a set of static mapping assertions.
- $\mathcal{M}_t$ is a set of temporal mapping assertions.
- $\mathcal{S}$ is a database schema.

Outline

unibz

Static ontology – Example

We use an **ontology** to model the **static knowledge** about

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

We still use **OWL 2 QL** as the static ontology language.

Devices consist of parts, and these are monitored by many different kinds of sensors (temperature, pressure, vibration etc.).

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

Static ontology – Example

We use an **ontology** to model the **static knowledge** about

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

We still use **OWL 2 QL** as the static ontology language.

Devices consist of parts, and these are monitored by many different kinds of sensors (temperature, pressure, vibration etc.).

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

Static rules

However, OWL 2 QL is not able to capture all the static knowledge required, e.g., in the Siemens use case.

We complement this ontology with **nonrecursive Datalog static rules**.

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

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

unibz

Static rules

However, OWL 2 QL is not able to capture all the static knowledge required, e.g., in the Siemens use case.

We complement this ontology with **nonrecursive Datalog static rules**.

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

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

unibz

Temporal rules

Siemens is interested in detecting abnormal situations, and monitoring running tasks.

"Purging is Over" is a complex event of a turbine

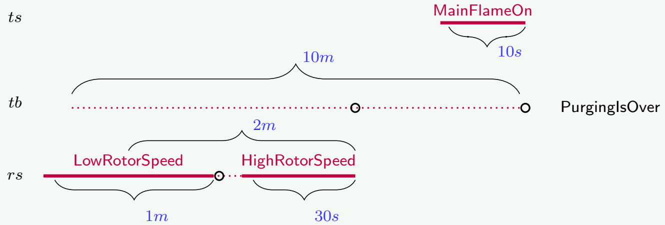
We model this situation with metric temporal rules:

$$
\begin{aligned}
\mathsf{PurgingIsOver}(tb) \;\leftarrow\;& \boxminus_{[0s,10s]}\mathsf{MainFlameOn}(ts) \land \\
& \diamondminus_{(0,10m]}\big(\boxminus_{(0,30s]}\mathsf{HighRotorSpeed}(rs) \land \\
& \qquad \diamondminus_{(0,2m]}\boxminus_{(0,1m]}\mathsf{LowRotorSpeed}(rs)\big) \land \\
& \mathsf{ColocTempRotSensors}(tb, ts, rs).\\
\mathsf{HighRotorSpeed}(tb) \;\leftarrow\;& \mathsf{rotorSpeed}(tb, v) \land v > 1260.\\
\mathsf{LowRotorSpeed}(tb) \;\leftarrow\;& \mathsf{rotorSpeed}(tb, v) \land v < 1000.
\end{aligned}
$$

Temporal rules

Siemens is interested in detecting abnormal situations, and monitoring running tasks.

"**Purging is Over**" is a complex event of a turbine



We model this situation with metric temporal rules:

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

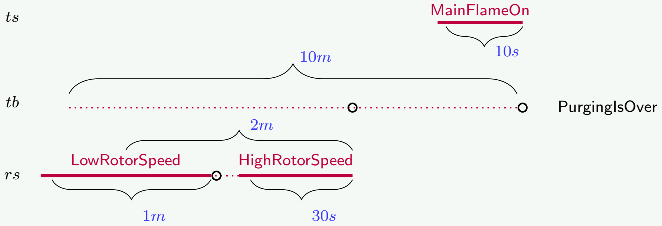
$$\mathsf{HighRotorSpeed}(tb) \leftarrow \mathsf{rotorSpeed}(tb, v) \land v > 1260.$$

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

Temporal rules

Siemens is interested in detecting abnormal situations, and monitoring running tasks.

"**Purging is Over**" is a complex event of a turbine



We model this situation with metric temporal rules:

$$
\begin{aligned}
\mathsf{PurgingIsOver}(tb) \ &\leftarrow\ \boxminus_{[0s,10s]}\mathsf{MainFlameOn}(ts) \land \\
&\quad \diamondminus_{(0,10m]}(\boxminus_{(0,30s]}\mathsf{HighRotorSpeed}(rs) \land \\
&\qquad\qquad \diamondminus_{(0,2m]}\boxminus_{(0,1m]}\mathsf{LowRotorSpeed}(rs)) \land \\
&\quad \mathsf{ColocTempRotSensors}(tb, ts, rs). \\
\mathsf{HighRotorSpeed}(tb) \ &\leftarrow\ \mathsf{rotorSpeed}(tb, v) \land v > 1260. \\
\mathsf{LowRotorSpeed}(tb) \ &\leftarrow\ \mathsf{rotorSpeed}(tb, v) \land v < 1000.
\end{aligned}
$$

We use *DatalogMTL*

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

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

$$A^+ \leftarrow A_1 \land \cdots \land A_k \qquad \text{or} \qquad \bot \leftarrow A_1 \land \cdots \land A_k,$$

where

- each A_i is either $\tau \neq \tau'$, or defined by the grammar

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

  where ϱ denotes a (left/right open or closed) interval with non-negative endpoints,
- A^+ does not contain $\diamondplus_\varrho$ or $\diamondminus_\varrho$ (since this would lead to undecidability).

unibz

Query evaluation in *DatalogMTL*

[Brandt, Kalayci, et al. 2017; Brandt, Güzel Kalayci, et al. 2018]

Theorem

Answering *DatalogMTL*queries is ExpSpace-complete in combined complexity.

We consider the **nonrecursive fragment** $Datalog_{nr}MTL$ of *DatalogMTL*:

- sufficient expressive power for many real-world situations
- computationally well-behaved

Answering $Datalog_{nr}MTL$ queries:

- Is PSpace-complete in combined complexity.
- Is in AC^0 in data complexity.
- Can be **reduced to SQL query evaluation**.

Hence, $Datalog_{nr}MTL$ is well suited as a temporal rule language for OBDA.

Query evaluation in *DatalogMTL*

[Brandt, Kalayci, et al. 2017; Brandt, Güzel Kalayci, et al. 2018]

Theorem

Answering *DatalogMTL*queries is EXPSPACE-complete in combined complexity.

We consider the **nonrecursive fragment** $\mathit{Datalog}_{nr}\mathit{MTL}$ of *DatalogMTL*:

- sufficient expressive power for many real-world situations
- computationally well-behaved

Answering $\mathit{Datalog}_{nr}\mathit{MTL}$ queries:

- Is PSPACE-complete in combined complexity.
- Is in AC^0 in data complexity.
- Can be **reduced to SQL query evaluation**.

Hence, $\mathit{Datalog}_{nr}\mathit{MTL}$ is well suited as a temporal rule language for OBDA.

Query evaluation in *DatalogMTL*

[Brandt, Kalayci, et al. 2017; Brandt, Güzel Kalayci, et al. 2018]

Theorem

Answering *DatalogMTL*queries is EXPSPACE-complete in combined complexity.

We consider the **nonrecursive fragment** *Datalog$_{nr}$MTL* of *DatalogMTL*:

- sufficient expressive power for many real-world situations
- computationally well-behaved

Answering *Datalog$_{nr}$MTL* queries:

- Is PSPACE-complete in combined complexity.
- Is in AC^0 in data complexity.
- Can be **reduced to SQL query evaluation**.

Hence, *Datalog$_{nr}$MTL* is well suited as a temporal rule language for OBDA.

Data sources: schema and data

Data sources often contain temporal information in the form of time-stamps.

Example data schema $\mathcal{S}$ for the Siemens data

It includes time-stamped sensor measurements and deployment details:

tb_measurement(timestamp, sensor_id, value),
tb_sensors(sensor_id, sensor_type, mnted_part, mnted_tb),
tb_components(turbine_id, component_id, component_type).

unibz

Data sources: schema and data

Data sources often contain temporal information in the form of time-stamps.

Example data schema $\mathcal{S}$ for the Siemens data

It includes time-stamped sensor measurements and deployment details:

tb_measurement(timestamp, sensor_id, value),
tb_sensors(sensor_id, sensor_type, mnted_part, mnted_tb),
tb_components(turbine_id, component_id, component_type).

unibz

Data sources: schema and data

Data sources often contain temporal information in the form of time-stamps.

Example data schema $\mathcal{S}$ for the Siemens data

It includes time-stamped sensor measurements and deployment details:

tb_measurement(timestamp, sensor_id, value),
tb_sensors(sensor_id, sensor_type, mnted_part, mnted_tb),
tb_components(turbine_id, component_id, component_type).

A corresponding data instance $\mathcal{D}_0$:

tb_measurement

timestamp	*sensor_id*	*value*
2017-06-06 12:20:00	rs01	570
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
...	...	...

Data sources: schema and data

Data sources often contain temporal information in the form of time-stamps.

Example data schema $\mathcal{S}$ for the Siemens data

It includes time-stamped sensor measurements and deployment details:

tb_measurement(timestamp, sensor_id, value),
tb_sensors(sensor_id, sensor_type, mnted_part, mnted_tb),
tb_components(turbine_id, component_id, component_type).

A corresponding data instance $\mathcal{D}_0$:

tb_measurement

timestamp	*sensor_id*	*value*
2017-06-06 12:20:00	rs01	570
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
...	...	...

Outline

unibz

Static mapping assertions in $\mathcal{M}_s$

Static mapping assertions: $\Phi(\vec{x}) \rightsquigarrow \Psi(\vec{x})$

- $\Phi(\vec{x})$ is a query over the source schema $\mathcal{S}$
- $\Psi(\vec{x})$ is an atom with predicate in Σ_s

Example

```
SELECT sensor_id AS X FROM tb_sensors
WHERE sensor_type = 1                                ⇝ TemperatureSensor(X)

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

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

These mappings retrieve from the database ordinary facts.

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

Static mapping assertions in $\mathcal{M}_s$

Static mapping assertions: $\Phi(\vec{x}) \rightsquigarrow \Psi(\vec{x})$

- $\Phi(\vec{x})$ is a query over the source schema $\mathcal{S}$
- $\Psi(\vec{x})$ is an atom with predicate in Σ_s

Example

```
SELECT sensor_id AS X FROM tb_sensors
WHERE sensor_type = 1                                   ⇝ TemperatureSensor(X)

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

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

These mappings retrieve from the database ordinary facts.

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

Static mapping assertions in $\mathcal{M}_s$

Static mapping assertions: $\Phi(\vec{x}) \leadsto \Psi(\vec{x})$

- $\Phi(\vec{x})$ is a query over the source schema $\mathcal{S}$
- $\Psi(\vec{x})$ is an atom with predicate in Σ_s

Example

```
SELECT sensor_id AS X FROM tb_sensors
WHERE sensor_type = 1                        ⇝ TemperatureSensor(X)

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

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

These mappings retrieve from the database ordinary facts.

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

Temporal mapping assertions in $\mathcal{M}_t$

Temporal mapping assertions: $\Phi(\vec{x}, \mathtt{begin}, \mathtt{end}) \rightsquigarrow \Psi(\vec{x})@\langle t_{\mathtt{begin}}, t_{\mathtt{end}}\rangle$

- begin and end are variables returning a date/time.
- '$\langle$' is either '(' or '[', and similarly for '$\rangle$'.
- $\Psi(\vec{x})$ is an atom with predicate in Σ_t.
- $t_{\mathtt{begin}}$ is either begin or a date-time constant, and similarly for $t_{\mathtt{end}}$.

Example

```
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                    ⇝ HighRotorSpeed(sensor_id)@[begin,end)
```

These mappings retrieve from the database temporal facts.

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

Temporal mapping assertions in $\mathcal{M}_t$

Temporal mapping assertions: $\Phi(\vec{x}, \mathtt{begin}, \mathtt{end}) \rightsquigarrow \Psi(\vec{x})@\langle t_{\mathtt{begin}}, t_{\mathtt{end}}\rangle$

- begin and end are variables returning a date/time.
- '$\langle$' is either '(' or '[', and similarly for '$\rangle$'.
- $\Psi(\vec{x})$ is an atom with predicate in Σ_t.
- $t_{\mathtt{begin}}$ is either begin or a date-time constant, and similarly for $t_{\mathtt{end}}$.

Example

```
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)

These mappings retrieve from the database temporal facts.

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

Temporal mapping assertions in $\mathcal{M}_t$

Temporal mapping assertions: $\Phi(\vec{x}, \mathtt{begin}, \mathtt{end}) \rightsquigarrow \Psi(\vec{x})@\langle t_{\mathtt{begin}}, t_{\mathtt{end}}\rangle$

- begin and end are variables returning a date/time.
- '$\langle$' is either '(' or '[', and similarly for '$\rangle$'.
- $\Psi(\vec{x})$ is an atom with predicate in Σ_t.
- $t_{\mathtt{begin}}$ is either begin or a date-time constant, and similarly for $t_{\mathtt{end}}$.

Example

```
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)

These mappings retrieve from the database temporal facts.

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

Outline

Concrete syntax for temporal OBDA specifications

Temporal OBDA specification $\mathcal{P}_t = \langle \Sigma_s, \Sigma_t, \mathcal{O}, \mathcal{R}_s, \mathcal{R}_t, \mathcal{M}_s, \mathcal{M}_t, \mathcal{S} \rangle$

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

Component	defines predicates in	in terms of predicates in	Adopted language
$\mathcal{O}$	Σ_s	Σ_s	OWL 2 QL
$\mathcal{R}_s$	Σ_s	Σ_s	non-recursive Datalog
$\mathcal{R}_t$	Σ_t	$\Sigma_s \cup \Sigma_t$	$Datalog_{nr}MTL$
$\mathcal{M}_s$	Σ_s	$\mathcal{S}$	R2RML / Ontop
$\mathcal{M}_t$	Σ_t	$\mathcal{S}$	R2RML / Ontop

unibz

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
]]
```

Temporal OBDA instance

A **temporal OBDA instance** is a pair $\mathcal{J} = \langle \mathcal{P}_t, \mathcal{D} \rangle$

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

To represent the facts generated by $\mathcal{J}$, we use **RDF Datasets**.

(We follow the model proposed by the W3C RSP Community Group.)

The temporal fact $\quad \mathsf{HighRotorSpeed}(\mathtt{rs01})@[\mathtt{2017\text{-}06\text{-}06\ 12{:}22{:}50}, \mathtt{2017\text{-}06\text{-}06\ 12{:}23{:}40})$

is modeled as the **named graph** $\quad \mathsf{GRAPH}\ g_0\ \{(\mathtt{rs01}, \mathsf{a}, \mathtt{HighRotorSpeed})\}$

and a set of triples in the default graph, to model the time interval:

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

Temporal OBDA instance

A **temporal OBDA instance** is a pair $\mathcal{J} = \langle \mathcal{P}_t, \mathcal{D} \rangle$

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

To represent the facts generated by $\mathcal{J}$, we use **RDF Datasets**.

(We follow the model proposed by the W3C RSP Community Group.)

The temporal fact $\mathsf{HighRotorSpeed}(\mathtt{rs01})@[\mathtt{2017\text{-}06\text{-}06\ 12{:}22{:}50}, \mathtt{2017\text{-}06\text{-}06\ 12{:}23{:}40})$

is modeled as the **named graph** $\mathsf{GRAPH}\ g_0\ \{(\mathtt{rs01}, \mathsf{a}, \mathtt{HighRotorSpeed})\}$

and a set of triples in the default graph, to model the time interval:

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

Temporal OBDA instance

A **temporal OBDA instance** is a pair $\mathcal{J} = \langle \mathcal{P}_t, \mathcal{D} \rangle$

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

To represent the facts generated by $\mathcal{J}$, we use **RDF Datasets**.

(We follow the model proposed by the W3C RSP Community Group.)

The temporal fact $\mathsf{HighRotorSpeed}(\mathtt{rs01})@[\mathtt{2017\text{-}06\text{-}06\ 12{:}22{:}50}, \mathtt{2017\text{-}06\text{-}06\ 12{:}23{:}40})$

is modeled as the **named graph** $\mathrm{GRAPH}\ g_0\ \{(\mathtt{rs01}, \mathsf{a}, \mathtt{HighRotorSpeed})\}$

and a set of triples in the default graph, to model the time interval:

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

Temporal OBDA instance

A **temporal OBDA instance** is a pair $\mathcal{J} = \langle \mathcal{P}_t, \mathcal{D} \rangle$

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

To represent the facts generated by $\mathcal{J}$, we use **RDF Datasets**.

(We follow the model proposed by the W3C RSP Community Group.)

The temporal fact $\mathsf{HighRotorSpeed}(\mathtt{rs01})@[\mathtt{2017\text{-}06\text{-}06\ 12{:}22{:}50}, \mathtt{2017\text{-}06\text{-}06\ 12{:}23{:}40})$

is modeled as the **named graph** GRAPH g_0 $\{(\mathtt{rs01}, \mathsf{a}, \mathtt{HighRotorSpeed})\}$

and a set of triples in the default graph, to model the time interval:

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

Temporal OBDA instance

A **temporal OBDA instance** is a pair $\mathcal{J} = \langle \mathcal{P}_t, \mathcal{D} \rangle$

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

To represent the facts generated by $\mathcal{J}$, we use **RDF Datasets**.

(We follow the model proposed by the W3C RSP Community Group.)

The temporal fact $\quad \mathsf{HighRotorSpeed}(\mathtt{rs01})@[\mathtt{2017\text{-}06\text{-}06\ 12{:}22{:}50}, \mathtt{2017\text{-}06\text{-}06\ 12{:}23{:}40})$

is modeled as the **named graph** $\quad \mathsf{GRAPH}\ g_0\ \{(\mathtt{rs01}, \mathsf{a}, \mathtt{HighRotorSpeed})\}$

and a set of triples in the default graph, to model the time interval:

$$
\begin{array}{l}
(g_0, \mathsf{a}, \mathtt{time{:}Interval}),\\
(g_0, \mathtt{time{:}isBeginningInclusive}, \mathtt{true}),\ (g_0, \mathtt{time{:}isEndInclusive}, \mathtt{false}),\\
(g_0, \mathtt{time{:}hasBeginning}, b_0),\ (g_0, \mathtt{time{:}hasEnd}, e_0),\\
(b_0, \mathtt{time{:}inXSDDateTimeStamp}, \text{‘}\mathtt{2017\text{-}06\text{-}06\ 12{:}22{:}50}\text{’}),\\
(e_0, \mathtt{time{:}inXSDDateTimeStamp}, \text{‘}\mathtt{2017\text{-}06\text{-}06\ 12{:}23{:}40}\text{’}).
\end{array}
$$

Query answering

Example query

"Find the gas turbines deployed in the train with the ID T001 and the time periods of their accomplished purgings".

This corresponds to the query: $Q(tb)@\iota$, where $Q(tb)$ is defined as

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

In "temporal" SPARQL syntax:

```
PREFIX ss: <http://siemens.com/ns#>
PREFIX st: <http://siemens.com/temporal/ns#>

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>
}
```

unibz

Query answering

Example query

"Find the gas turbines deployed in the train with the ID T001 and the time periods of their accomplished purgings".

This corresponds to the query: $Q(tb)@\iota$, where $Q(tb)$ is defined as

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

In "temporal" SPARQL syntax:

```
PREFIX ss: <http://siemens.com/ns#>
PREFIX st: <http://siemens.com/temporal/ns#>

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>
}
```

unibz

Query answering

Example query

"Find the gas turbines deployed in the train with the ID T001 and the time periods of their accomplished purgings".

This corresponds to the query: $Q(tb)@\iota$, where $Q(tb)$ is defined as

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

In "temporal" SPARQL syntax:

```
PREFIX ss: <http://siemens.com/ns#>
PREFIX st: <http://siemens.com/temporal/ns#>

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 standard SPARQL query

```
PREFIX ss: <http://siemens.com/ns#>
PREFIX st: <http://siemens.com/temporal/ns#>

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>
}
```

... is translated into the following standard SPARQL query:

```
PREFIX time: <http://www.w3.org/2006/time#>
...
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 .
}
```

Equivalent standard SPARQL query

```
PREFIX ss: <http://siemens.com/ns#>
PREFIX st: <http://siemens.com/temporal/ns#>

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>
}
```

. . . is translated into the following standard SPARQL query:

```
PREFIX time: <http://www.w3.org/2006/time#>
...
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 .
}
```

unibz

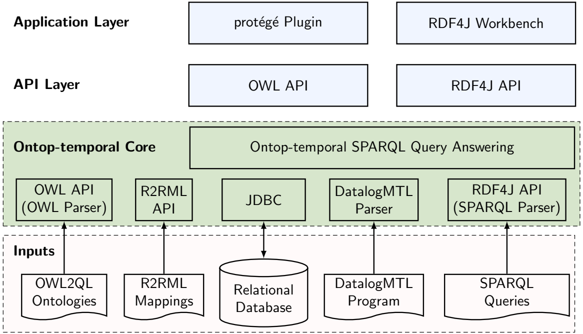
The Ontop-temporal system

We have implemented the temporal OBDA framework in the **Ontop-temporal** system [Güzel Kalayci et al. 2018], as an extension of *Ontop*.

unibz

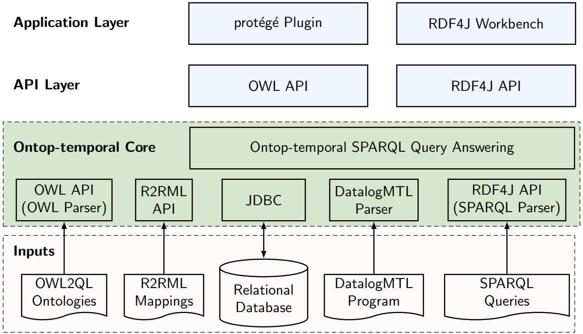
The Ontop-temporal system

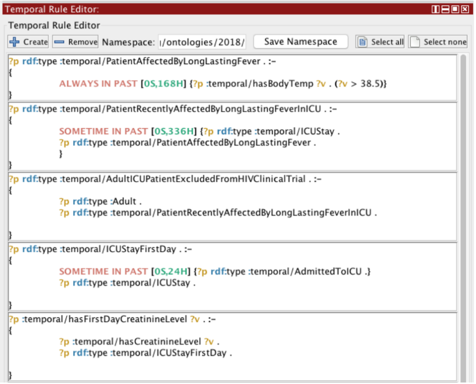
We have implemented the temporal OBDA framework in the **Ontop-temporal** system [Güzel Kalayci et al. 2018], as an extension of *Ontop*.



The Ontop-temporal system

We have implemented the temporal OBDA framework in the **Ontop-temporal** system [Güzel Kalayci et al. 2018], as an extension of *Ontop*.

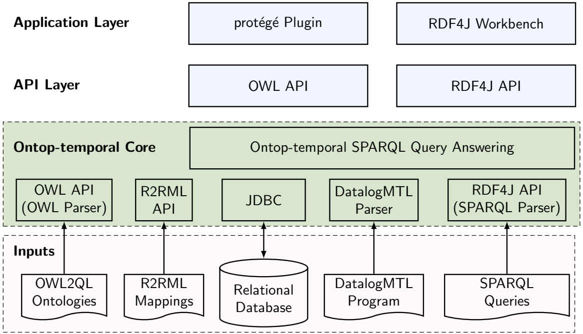


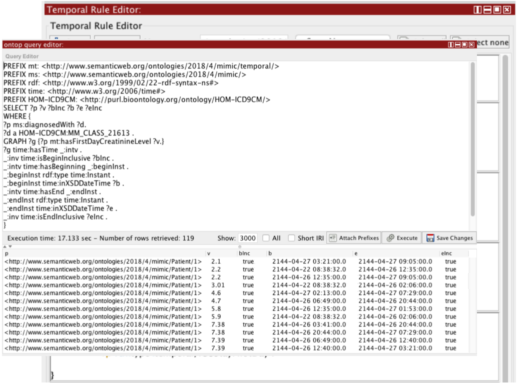


unibz

The Ontop-temporal system

We have implemented the temporal OBDA framework in the **Ontop-temporal** system [Güzel Kalayci et al. 2018], as an extension of *Ontop*.



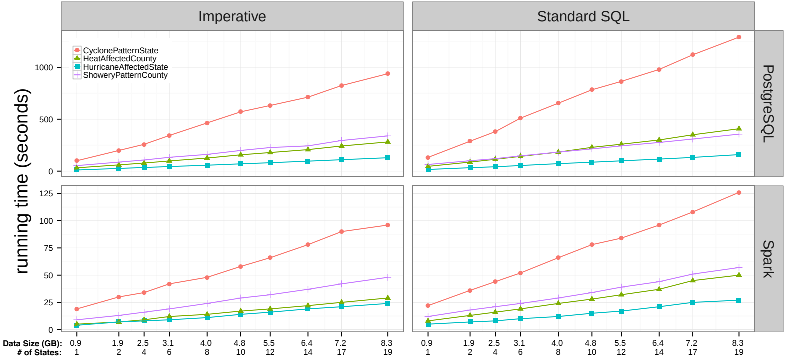


Evaluation [Brandt, Güzel Kalayci, et al. 2018; Güzel Kalayci et al. 2018]

Done on three real-world use cases from different domains: MesoWest, MIMIC-III, and Siemens.

Evaluation with respect to two aspects:

- **utility evaluation** over MIMIC-III, MesoWest, and Siemens use-cases
- **scalability evaluation** over MesoWest and Siemens use-cases



unibz

Outline

Summary and future work

Contributions

- A novel **temporal OBDA framework**.
- A **temporal rule language** for modeling complex temporal events.
- Implementation of the framework in the **Ontop-temporal** system.
- Extensive **evaluation** of the system over real-world data from different domains.

Future work

- Extension of temporal OBDA with temporal aggregates.
- Support for streaming data.
- Optimization and support for parallel processing

unibz

Thanks

Thank you for your attention!

Thanks

Thank you for your attention!

... and thanks to those who contributed to this work:

- Elem Güzel Kalayci (unibz)
- Roman Kontchakov (Birkbeck)
- Vladislav Ryzhikov (unibz, Birkbeck)
- Guohui Xiao (unibz)
- Michael Zakharyaschev (Birkbeck)

OBDA framework developed in Bolzano

ontop

`ontop.inf.unibz.it/`

unibz

References I

[1] Victor Gutiérrez-Basulto and Szymon Klarman. "Towards a Unifying Approach to Representing and Querying Temporal Data in Description Logics". In: *Proc. of the 6th Int. Conf. on Web Reasoning and Rule Systems (RR)*. Vol. 7497. Lecture Notes in Computer Science. Springer, 2012, pp. 90–105. DOI: 10.1007/978-3-642-33203-6_8.

[2] Franz Baader, Stefan Borgwardt, and Marcel Lippmann. "Temporalizing Ontology-based Data Access". In: *Proc. of the 24th Int. Conf. on Automated Deduction (CADE)*. Vol. 7898. Lecture Notes in Computer Science. Springer, 2013, pp. 330–344. DOI: 10.1007/978-3-642-38574-2_23.

[3] Szymon Klarman and Thomas Meyer. "Querying Temporal Databases via OWL 2 QL". In: *Proc. of the 8th Int. Conf. on Web Reasoning and Rule Systems (RR)*. Vol. 8741. Lecture Notes in Computer Science. Springer, 2014, pp. 92–107. DOI: 10.1007/978-3-319-11113-1_7.

[4] Özgür Lütfü Özçep and Ralf Möller. "Ontology Based Data Access on Temporal and Streaming Data". In: *Reasoning Web: Reasoning on the Web in the Big Data Era – 10th Int. Summer School Tutorial Lectures (RW)*. Vol. 8714. Lecture Notes in Computer Science. Springer, 2014, pp. 279–312.

References II

[5] Evgeny Kharlamov et al. "Ontology-Based Integration of Streaming and Static Relational Data with Optique". In: *Proc. of the 37th ACM Int. Conf. on Management of Data (SIGMOD)*. 2016, pp. 2109–2112. DOI: 10.1145/2882903.2899385.

[6] Alessandro Artale, Roman Kontchakov, Frank Wolter, and Michael Zakharyaschev. "Temporal Description Logic for Ontology-Based Data Access". In: *Proc. of the 23rd Int. Joint Conf. on Artificial Intelligence (IJCAI)*. AAAI Press, 2013, pp. 711–717.

[7] Alessandro Artale, Roman Kontchakov, Alisa Kovtunova, Vladislav Ryzhikov, Frank Wolter, and Michael Zakharyaschev. "First-order Rewritability of Temporal Ontology-mediated Queries". In: *Proc. of the 24th Int. Joint Conf. on Artificial Intelligence (IJCAI)*. AAAI Press, 2015, pp. 2706–2712.

[8] Sebastian Brandt, Elem Güzel Kalayci, Roman Kontchakov, Vladislav Ryzhikov, Guohui Xiao, and Michael Zakharyaschev. "Ontology-Based Data Access with a Horn Fragment of Metric Temporal Logic". In: *Proc. of the 31st AAAI Conf. on Artificial Intelligence (AAAI)*. AAAI Press, 2017, pp. 1070–1076.

unibz

References III

[9] Sebastian Brandt, Elem Güzel Kalayci, Vladislav Ryzhikov, Guohui Xiao, and Michael Zakharyaschev. "Querying Log Data with Metric Temporal Logic". In: *J. of Artificial Intelligence Research* 62 (2018), pp. 829–877.

[10] Elem Güzel Kalayci, Guohui Xiao, Vladislav Ryzhikov, Tahir Emre Kalayci, and Diego Calvanese. "Ontop-temporal: A Tool for Ontology-based Query Answering over Temporal Data". In: *Proc. of the 27th ACM Int. Conf. on Information and Knowledge Management (CIKM)*. 2018, pp. 1927–1930. DOI: 10.1145/3269206.3269230.