Ontology Based Data Access on Temporal and Streaming Data*

Özgür L. Özçep and Ralf Möller

Institute for Softwaresystems (STS)
Hamburg University of Technology
Hamburg, Germany
{oezguer.oezcep,moeller}@tu-harburg.de

Abstract. Though processing time-dependent data has been investigated for a long time, the research on temporal and especially stream reasoning over linked open data and ontologies is reaching its high point these days. In this tutorial, we give an overview of state-of-the art query languages and engines for temporal and stream reasoning. On a more detailed level, we discuss the new language STARQL (Reasoning-based Query Language for Streaming and Temporal ontology Access). STARQL is designed as an expressive and flexible stream query framework that offers the possibility to embed different (temporal) description logics as filter query languages over ontologies, and hence it can be used within the OBDA paradigm (Ontology Based Data Access in the classical sense) and within the ABDEO paradigm (Accessing Big Data over Expressive Ontologies).

Keywords: Ontology Based Data Access, streams, temporal logics, rewriting, unfolding, semantic web

1 Introduction

Ontology based data access (OBDA) [20] stands for a paradigm of accessing huge data sets through an interface query language that relies on a signature for which constraints are modeled in a knowledge base called ontology. Skimming the publications of conferences/journals on description logics, extended database management systems or the semantic web shows that OBDA has become an important topic with various subtopics and new slightly deviating research aims—in particular, widening its original scope from lightweight representation languages to more expressive ones (e.g., [58] describes an approach for accessing big data with expressive ontologies (ABDEO)). Moreover, OBDA seems to have found its way into industrial applications. In the EU funded FP7 project OPTIQUE [22], an OBDA based software platform is developed to fulfill the needs of the two industrial stakeholders STATOIL and SIEMENS.

* This work has been supported by the European Commission as part of the FP7 project Optique http://www.optique-project.eu/.

A look into recent publications [15,8,18] reveals that OBDA is ripe for the integration of formal approaches dealing with the processing of temporal and streaming data. Upcoming research in temporalizing and streamifying OBDA is going to push OBDA's industrial attractiveness because use cases with some or other form of processing temporal and/or streaming data abound—be it event recognition, monitoring, sensor networking, text processing, video processing, time-series applications, just to name a few.

The rich literature on temporal logics and stream processing on the level of sensor data and relational stream data management systems (see the following sections) provide good directions for the actual venture of temporalizing and streamifying OBDA. The neat semantics of temporal logics combined with the practical and well-proven sliding window operator for streams founds the basis on which to built high-level query languages fitting to the OBDA paradigm and related paradigms such as ABDEO.

Next to a discussion of recent approaches for temporal and streamified OBDA we are also going to illustrate a new query language framework called STARQL which can be used within the classical OBDA as well as the ABDEO paradigm. It heavily relies on a window operator for building sequences of ABoxes; the sequence structure is used to combine classical intra-ABox reasoning/rewriting with temporal inter-ABox reasoning.

The paper is structured as follows. Section 2 introduces the paradigm of ontology based data access. Discussing in detail classical OBDA centered around the family of description logics DL-Lite in Sect. 3, we also discuss OBDA in a broader sense for more expressive logics in Sect. 4. Approaches introducing a temporal dimension into OBDA are discussed in 5, and, in a more detailed manner, approaches introducing streams into OBDA are discussed in Section 6. Section 7 before the conclusion contains an overview of the new stream-temporal language STARQL framework.

2 Ontology Based Data Access

Ever since its introduction, Ontology Based data Access (OBDA) has become a widely accepted research topic in different areas such as the semantic web, database theory, and the description logics, and, moreover, it finds its way into industrial applications, two of which are the STATOIL and the SIEMENS use cases in the FP7 framework OPTIQUE `http://www.optique-project.eu/`.

The driving idea behind ontology based data access is to keep the source data where they are and access them with a query interface over a declarative knowledge base called ontology. In description logic speak, the ontology is made up of a TBox, which models the terminological part of the domain by general subconcept and subrole axioms and perhaps additional constraints such as functionality assertions, and an ABox, which is the logical presentation of the data as assertional axioms produced my mappings from the data sources.

In the classical OBDA approach, the ABox representation of the data by mappings is just for the purpose of defining the semantics of queries w.r.t. an ontology. So, the ABox is not materialized for the purpose of query answering; instead, complete and correct answering w.r.t. the ontology is handled by compiling the knowledge TBox into the given query. Afterwards, the compiled query, also called the rewritten query, is unfolded w.r.t. the mappings into a query over the data sources, e.g., an SQL query over a relational database.

What are the benefits of OBDA querying over direct querying within the language of the data sources? The first aspect is that one has a declarative language w.r.t. a model described in the TBox and the data in the ABox, thereby allowing to distinguish between a neat representation of intensional knowledge and factual knowledge.

The second aspect is that of semantic integration [63]. Different heterogeneous sources can be accessed under a common ontological interface. Users of the query language on the ontological have to consider/use only the language of the ontology and the associated query language, but do not have to deal with possibly different languages of different sources. Additionally, different models can be implemented by different TBoxes without the need to recompile queries. This offers to set up test scenarios, in which different models (represented by the TBox) can be tested with the same queries (as long as the TBoxes use the same signature.) Following this line, if a TBox has proven to be good or useful, it can be used with various data sources, as long as the mappings are adapted to the various data source in different use cases.

The OBDA approach offers a way to deal with the indefiniteness and incompleteness in data that differs radically from the NULL value approach in relational databases. OBDA abandons the closed world approach by considering many possible models of the ontology. The role of the axioms in the ontology then is to minimize the indefiniteness in the data by axioms constraining all possible models to the intended and most likely ones.

In the classical OBDA approach one has, moreover, the benefit of the rewriting approach, namely that the huge data set does not have to be transformed into the working memory; so, one can rely on the optimization and index mechanisms of the secondary memory data sources, as a query can be compiled directly into a query over the datasources. Clearly a caveat is that the rewritten query may be exponentially bigger than the original query, so that additional optimizations already during the rewriting steps may be necessary. Some optimization strategies are discussed in [73].

As of the time of writing, OBDA related research has diverged into different investigations that cannot be labelled as OBDA in the classical sense. So, we propose the following distinctions in order to get a clear picture. There is, first, OBDA in a narrow sense by which we mean that there is an access to data with mappings producing only a virtual or materialized ABox but with no TBox at all or of expressiveness on the level of RDFS++. Second, OBDA in the classical sense uses logics in the DL-Lite family that allow for the generation of objects (true existentials on the right hand sides of general inclusion axioms) but still allow for a reduction of reasoning w.r.t. the TBox to answering of rewritten first order logic queries on the original data. For some applications even the expressiveness of the classical OBDA paradigm is not sufficient, and hence there is a need for the even more challenging paradigm of accessing big data w.r.t. expressive ontologies (ABDEO).

Orthogonally to these categorization one can also observe that the source data not necessarily have to be relational databases but can equally be RDF triple sources or other non-relational DBs. More radically, in some approaches the data of the original sources are imported into an own DB, which then is the proper source w.r.t. which the queries on the ontological level are queried. A benefit of this approach is the fact that one can define index structures and optimization strategies which are more appropriate for the domain and also for the rewriting strategy one is going to apply. In general, the data may even be preprocessed. The strategy to actually use the data underlying the ABox during the rewriting process is also part of a different rewriting strategy called *combined rewriting* [48].

Next to this hierarchy of OBDA related approaches one has to consider the ontology based access on stream data (OBSA) as an extra paradigm category. The idea of this paradigm is to make the access on big data feasible by first partitioning the data into small chunks (modules), then, second, ordering them w.r.t. some priority criterion and third streaming these into the query answering w.r.t. the fixed ordering (compare the survey [78]). One particular but also the most common case of (OBSA) is the one where the ordering is given by a temporal ordering so that the streams are timestamped data.

3 Classical OBDA

We recapitulate the most important notions related to classical OBDA. For a more detailed view, the reader is referred to [20]. Classical OBDA investigates query answering over an ontology, using mappings to get the data from the sources to the ontology level—the main aim being the reduction of the demanding query answering problem to a model checking/query answering problem over the data sources, which are in many cases relational databases.

The idea of aiming at this reduction is motivated by the demand to enable computationally feasible reasoning services over large ABoxes. Because the size of the TBox (and the queries) is small with respect to the size of the ABoxes, computational feasibility is measured with respect to the size of the ABox alone, thereby fixing all other parameters (TBox, query respectively). The resulting type of complexity is called *data complexity*. Aiming at the reduction, which is also called first order logic (FOL) rewritability and which is explained in detail below, is indeed a successful venture with respect to computational feasibility. This is due to the fact that the data complexity of answering first order logic queries w.r.t. DL-Lite ontologies is in the low boolean circuits complexity class AC^0, which, roughly, is the class of problems that can be decided in constant time with the help of polynomially many processors.

3.1 DL-Lite

Descriptions logics [11] have proven to be an adequate representation language for ontologies, as they have a formal semantics and show good computation properties at least w.r.t. various standard reasoning services such as subsumption testing, satisfiability testing, query answering etc. In the center of classical OBDA stands query answering and also, related to it, satisfiability testing of ontologies. An ontology is defined as a triple $\mathcal{O} = (Sig, \mathcal{T}, \mathcal{A})$ over signature, a TBox and an ABox. In all DLs the signature is made up by subsets of a set of concept symbols N_C, a set of role symbols N_R, and a set of individual constant symbols N_i. DLs with concrete domains or datatypes also allow for additional constants (and predicates) with fixed meanings over the concrete domain. The DLs differ in the set of concept/role constructors they offer and in the constraints for building TBox and ABox axioms. Typically TBox axioms are concept subsumptions $C \sqsubseteq D$ or role subsumptions $R \sqsubseteq S$ and ABox axioms have the form $C(a)$ or $R(a, b)$, where C, D stand for concept descriptions, R, S for role descriptions and a, b for individual constants.

In the focus of of OBDA stands the family of DLs called DL-Lite [6] as it is tailored towards FOL rewritability. DL-Lite is the language family underlying the OWL 2 QL profile of the W3C recommend web ontology language OWL `http://www.w3.org/TR/owl2-profiles/#OWL_2_QL`. As FOL rewritability is a very strong property it should be no surprise that only lightweight logics such as DL-Lite are considered as representation language for the ontology and moreover that there are also limitations on the query language, which in this case is unions of conjunctive queries (UCQs). (But note, that the limits of expressivity under FOL rewriting can still be pushed a little bit further as shown by the extended family of Datalog$^\pm$ family [19].)

To make the discussion more concrete we give the syntax of a DL-Lite language and its semantics in Fig. 1. The TBox axioms are additionally constrained by the demand that functional roles are not allowed to occur on the right hand side of role axioms. The semantics of concept descriptions is defined recursively on the basis of an interpretations $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, which consists of a domain $\Delta^{\mathcal{I}}$ and a denotation function $\cdot^{\mathcal{I}}$. The denotation of concept symbols (atomic concepts) A are subsets $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ of the domain; role symbols P are denoted by binary relations $P^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and constants a are denoted by elements of the domain $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$. The modeling relation is denoted by $\models$ and one defines that $\mathcal{I}$ models or makes true an axiom ax iff $\mathcal{I} \models ax$. An ontology is called satisfiable if there is an interpretation $\mathcal{I}$ makes all axioms in the TBox and the ABox true. An ontology $\mathcal{O}$ entails an axiom ax, shortly: $\mathcal{O} \models ax$ iff all models of $\mathcal{O}$ are also models of ax.

3.2 Query Answering and Rewritability

An *FOL query* $Q = \psi(\boldsymbol{x})$ is a first-order logic formula $\psi(\boldsymbol{x})$ whose free variables are the ones in the n-ary vector of variables $\boldsymbol{x}$; the variables in $\boldsymbol{x}$ are called *distinguished variables.* If $\boldsymbol{x}$ is empty, the query is called boolean. Let $\boldsymbol{a}$ be a

$$\begin{array}{ll}
R \longrightarrow P \mid P^- & \\
B \longrightarrow A \mid \exists R & \\
C \longrightarrow B \mid \neg B & \\
\textit{TBox:} & B \sqsubseteq C, (\text{func } R), \\
 & R_1 \sqsubseteq R_2 \\
\textit{ABox:} & A(a), R(a,b)
\end{array}
\qquad
\begin{array}{rcl}
(P^-)^{\mathcal{I}} & = & \{(d,e) \mid (e,d) \in P^{\mathcal{I}}\} \\
(\exists R)^{\mathcal{I}} & = & \{d \in \Delta^{\mathcal{I}} \mid \exists e.(d,e) \in R^{\mathcal{I}}\} \\
(\neg C)^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
\mathcal{I} \models B \sqsubseteq C & \text{iff} & B^{\mathcal{I}} \subseteq C \\
\mathcal{I} \models R_1 \sqsubseteq R_2 & \text{iff} & R_1^{\mathcal{I}} \subseteq R_2 \\
\mathcal{I} \models B(a) & \text{iff} & a^{\mathcal{I}} \in B^{\mathcal{I}} \\
\mathcal{I} \models R(a,b) & \text{iff} & (a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}} \\
\mathcal{I} \models (\text{func } R) & \text{iff} & R^{\mathcal{I}} \text{ is a (partial) function}
\end{array}$$

Fig. 1. DL-Lite

vector of constants from the signature of the ontology. The semantics of n-ary FOL queries with respect to an interpretation $\mathcal{I}$ is given by the set $Q^{\mathcal{I}}$ of n-ary tuples $\boldsymbol{d}$ over the domain $\Delta^{\mathcal{I}}$ such that $\mathcal{I}_{[\boldsymbol{x} \mapsto \boldsymbol{d}]} \models \psi(\boldsymbol{x})$. Here, $\mathcal{I}_{[\boldsymbol{x} \mapsto \boldsymbol{d}]}$ extends $\mathcal{I}$ by interpreting the variables in $\boldsymbol{x}$ by the elements in $\boldsymbol{d}$.

The crucial notion of answers w.r.t. an ontology is handled by an *certain answer semantics* also known from database theory. We are not going to discuss the appropriateness of this kind of semantics but just state its definition. (For an adequateness discussion of certain answer semantics in particular w.r.t aggregation we refer the reader to [49]). Given an ontology $(Sig, \mathcal{T}, \mathcal{A})$, the set of certain answers is denoted $cert(Q, \mathcal{T} \cup \mathcal{A})$ and it consists of n-ary tuples of constants $\boldsymbol{a}$ from Sig such that $\psi[\boldsymbol{x}/\boldsymbol{a}]$ (i.e. the formula resulting from $\psi(\boldsymbol{x})$ by applying the substitution $[\boldsymbol{x}/\boldsymbol{a}]$) is entailed by the ontology.

$$cert(\psi(\boldsymbol{x}), \mathcal{T} \cup \mathcal{A}) = \{\boldsymbol{a} \mid \mathcal{T} \cup \mathcal{A} \models \psi[\boldsymbol{x}/\boldsymbol{a}]\}$$

FOL queries are too complex to be used as queries on the ontological level. Hence, in order to guarantee FOL rewritability two well known weaker subclasses of FOL queries are considered, *conjunctive queries (CQ)* and *unions of conjunctive queries (UCQ)*. A CQ is a FOL query in which $\psi(\boldsymbol{x})$ is an existentially quantified conjunction of atomic formulas $at(\cdot)$, $\psi(\boldsymbol{x}) = \exists \boldsymbol{y} \bigwedge_i at_i(\boldsymbol{x}, \boldsymbol{y})$. The UCQs allow disjunctions of CQs, i.e., $\psi(\boldsymbol{x})$ can have the form $\exists \boldsymbol{y_1} \bigwedge_{i_1} at_{i_1}(\boldsymbol{x}, \boldsymbol{y}_1) \vee \cdots \vee \exists \boldsymbol{y_n} \bigwedge_{i_n} at_{i_n}(\boldsymbol{x}, \boldsymbol{y}_n)$. Note that the existential quantifiers in UCQs are interpreted in the same way as for FOL formulas (*natural domain semantics*) and not with respect to a given set of constants mentioned in the signature (*active domain semantics*). This is the strength of this query language and the critical place for weakening it if the OBDA application at hand demands TBox languages stronger than DL-Lite.

In the following, let the canonical model of an ABox $\mathcal{A}$, denoted $DB(\mathcal{A})$, be the minimal Herbrand model of $\mathcal{A}$, i.e. the model, where the domain is made up by all constants occurring in the ABox, all constants are interpreted by themselves, and where $a^{DB(\mathcal{A})} \in A^{DB(\mathcal{A})}$ iff $A(a) \in \mathcal{A}$ and $(a^{DB(\mathcal{A})}, b^{DB(\mathcal{A})} \in R^{DB(\mathcal{A})}$ iff $R(a,b) \in \mathcal{A}$. *Checking the satisfiability of ontologies is FOL rewritable* iff for all TBoxes $\mathcal{T}$ there is a boolean FOL query $Q_{\mathcal{T}}$ such that for all $\mathcal{A}$ it is the case that the ontology $\mathcal{T} \cup \mathcal{A}$ is satisfiable just in case the query $Q_{\mathcal{T}}$ evaluates to false in the model $DB(\mathcal{A})$. *Answering queries from a subclass $\mathcal{C}$ of FOL queries w.r.t. to ontologies is FOL rewritable* iff for all TBoxes $\mathcal{T}$ and queries $Q = \psi(\boldsymbol{x})$ in $\mathcal{C}$ there is a FOL query $Q_{\mathcal{T}}$ such that for all ABoxes $\mathcal{A}$ it is the case that $cert(Q, \mathcal{T} \cup \mathcal{A}) = Q_{\mathcal{T}}^{DB(\mathcal{A})}$.

For members of the DL-Lite family it can be shown [20] that the satisfiability check is FOL rewritable. As an example take $\mathcal{T} = \{A \sqsubseteq \neg B\}$ and $\mathcal{A} = \{A(a), B(a)\}$, then satisfiability is tested by answering the query $Q_{\mathcal{T}} = \exists x.A(x) \wedge B(x)$ w.r.t. $DB(\mathcal{A})$, resulting in the answer yes and indicating that $\mathcal{T} \cup \mathcal{A}$ is unsatisfiable. Moreover, answering UCQs can be shown to be FOL rewritable [20]. The rewriting technique used in [20] is called *perfect rewriting*. It considers the (positive) axioms as rules and uses a backward-chaining method to blow up the query with new CQ covering the rules' additional implications. FOL rewritability of satisfiability is a prerequisite for answering queries because in case the ontology is not satisfiable the set of certain answers is identical to all tuples of constants in the signature.

In what sense is the fact that DL-Lite provides existential quantification with natural domains semantics a benefit over those approaches such as the very narrow OBDA approaches using RDFS++? Having such existential quantification operators means that one can ask for objects which are not mentioned in the ABox but whose existence is guaranteed w.r.t. the TBox. Let us illustrate this strength with a risk management scenario for measurement data and event data from sensors and control units of turbines. The TBox is assumed to contain a (predictive) subsumption that says that if a turbine shows some anomaly in some fixed sensor, then there is a risk of a can flame failure.

$$\exists showsAnomalyInTempSens \sqsubseteq \exists risk.canFlameFailure$$

These observations have direct consequences for possible queries. For example, the engineer might be interested in all turbines for which there is a risk for can flame failure. This could be formulated in following conjunctive query

$$Q = \exists y.risk(x, y) \wedge canFlameFailure(y)$$

The variable y cannot be bound to an object within the data. But nonetheless, this query may have positive answers due to the predictive subsumption above which would lead to a reformulation of the query, adding a CQ and giving the UCQ

$$Q_{rew} = (\exists y.risk(x, y) \wedge canFlameFailure(y)) \vee showsAnomalyInTempSens(x)$$

Without the subsumption (resulting from the time series analysis), some turbines wold not have been identified as being in a risk to run in a can flame failure.

3.3 Mappings

In the classical OBDA setting, the ABox is not given in advanced but produced by mappings [71]. These are formally represented as rules with queries of the ontological level, called the target, as the head of the rule (here the left-hand side) and queries in the the data source language (in many cases SQL) as the body of the rule, which is noted here always on the right-hand side. We are going to present mappings in the logical notation. A recent W3C recommended mapping language in machine readable form is R2RML, a mapping language from relational databases to RDF (`http://www.w3.org/TR/r2rml/`). As constructing mappings is a non-trivial task recent research considers also bootstrapping mappings, see, e.g., the technical report [28] and the papers to be fond therein.

We illustrate mappings within a sensor measurement scenario, assuming that there is one central DB with sensor measurement data and also sensor descriptions w.r.t. the DB schema in Fig 2. The ontology is assumed to model sensors,

```
SENSOR(SID, CID, Sname, TID, description)
SENSORTYPE(TID, Tname)
COMPONENT(CID, superCID, AID, Cname)
ASSEMBLY(AID, AName, ALocation)
MEASUREMENT(MID, MtimeStamp, SID, Mval)
MESSAGE(MesID, MesTimeStamp, MesAssemblyID,
        catID, MesEventText)
CATEGORY(catID, catName)
```

Fig. 2. Part of the relational schema in a measurement DB

measurements, events etc. in the same manner as the nearly standard semantic sensor networks (SSN) ontology authored by the members of the W3C Sensor Network Incubator Group [27]. (See also the final report at `http://www.w3.org/2005/Incubator/ssn/XGR-ssn-20110628/`.) A general ontology of this kind can be extended for specific sensor measurement scenarios by introducing new names to the signature of the ontology and adding new ontology axioms. Here, we assume that there is a concept symbol *Sens* (for sensors) and an attribute symbol *name*. ABox assertions saying which element is a sensor and what their names are, are produced by the following mapping:

$m : Sens(x) \wedge name(x, y) \longleftarrow$
`SELECT f(SID) as x, Sname as y FROM SENSOR`

The information in the row of the measurement table is mapped to unary facts ($Sens(x)$) and binary atomic facts ($name(x, y)$). If the table `SENSOR` contains a row

```
(123, comp45, TC255, TempSens, ‘A temperature sensor’)
```

then the mapping produces the conjunction of ABox assertions $Sens(f(123)) \wedge name(f(123), TempSens123)$.

In the DL-Lite setting, mappings have in general on their left-hand side CQs and on their right-hand side SQL queries. The mappings are safe in the sense that all variables used on the left-hand side must occur on the right-hand side as columns; but the source query may contain additional variables.

The term `f(SID)` denotes an individual constant, constructed as a functional term indicating the value of the attribute `SID` of the sensor. All expressions `f(SID)` could be mapped to the more convenient atomic names of the form, e.g., s_i. If the ontology language allows for datatypes or concrete domains—as we assume here—then we can use attribute values directly without the need of an additional functional symbol. This is demonstrated above for the column `Sname` containing strings.

For different purposes mappings can be split up into a simpler form where the target consist of an atomic query only. Within the splitting the source query is projected to the variables occurring in the atom; in the case of the query above the resulting split mappings would be as follows:

$$\begin{aligned} m_1 &: Sens(x) &&\longleftarrow \texttt{SELECT f(SID) as x,FROM SENSOR} \\ m_2 &: name(x,y) &&\longleftarrow \texttt{SELECT f(SID) as x, Sname as y FROM SENSOR} \end{aligned}$$

For a given database DB and a set of mappings M, the induced ABox $\mathcal{A}(M, DB))$ is just the union of the ABox assertions produced by the mappings in M over the DB. The semantics of query answering w.r.t. a set of mappings over a DB and a TBox is just the certain answer semantics introduced above and applied to the ontology $(Sig, \mathcal{T}, \mathcal{A}(M, DB))$.

Now the important point is that the induced ABox is not materialized for query answering but is kept virtual. Queries over the induced ABox are *unfolded* to queries over the DB. So, in the classical approach a UCQ over a TBox and the induced ABox of mappings w.r.t. a DB is first rewritten into a FOL query, then this query is unfolded into an SQL query over the DB (using the mappings) and then the unfolded query is evaluated over the DB, given the set of answers to the original query.

There is no canonical way for unfolding a UCQ into a SQL query, and, indeed, different strategies for unfolding a UCQ w.r.t DL-Lite ontologies are discussed in the literature, e.g., one strategy is introduced in[71], another in [74]. The common idea of both strategies is to view the mappings as logical rules and use logical programming ideas such as resolution to get the unfolded query. We do not spell out the procedure but only mention very roughly, that unfolding an atomic query q results in an SQL query which is a union of all the source queries Ψ_i from mappings $m_i : q \longleftarrow \Psi_i$ with q as their target. Recursively, if two queries q_1, q_2 are unfolded to SQL queries Ψ_1 and Ψ_2, then conjunctive query $q_1 \wedge q_2$ is mapped to a join of Ψ_1 and Ψ_2. The disjunction $q_1 \vee q_2$ (if it corresponds to a UCQ) is unfolded to the union of Ψ_1 and Ψ_2.

As the rewriting of queries may lead to an exponential blow-up, optimizations on different levels (rewriting, unfolding and mappings) are a must for OBDA systems. Different optimization strategies are discussed in [73], [75] and implemented, e.g., in the -ontop- OBDA system (`http://ontop.inf.unibz.it/`).

4 OBDA in the Broad Sense: ABDEO

Though DL-Lite provides existential quantification with natural domain semantics in the TBox and in the query language, for many use cases it does not have sufficient representation capabilities. For example, qualified existential quantification on the left-hand side of TBox axioms is not allowed, though such constructors are necessary for identifying interesting (though still tree-shaped) patterns and using them to deduce new knowledge.

Another logical constructor which cannot be handled with DL-Lite are transitive declarations for roles because transitive roles lead directly to non-FOL rewritability. Transitive relations are useful modeling means for modeling part-of-relations of components in a complex system such as a turbine. The typical entity-relationship modeling methodology for representing chains or tree-like structures by a self-referencing foreign key is illustrated with the `COMPONENT` schema of Fig. 2 and picked up here again.

```
COMPONENT(CID, superCID, AID, Cname)
ASSEMBLY(AID, AName, ALocation)
```

The `superCID` column specifies the next upper component at which the component with identifier `CID` is attached (foreign key to attribute `CID` of `COMPONENT` itself). The turbine assembly at which the component is (indirectly) attached is given by `AID` (foreign key from `COMPONENT` to `ASSEMBLY`).

Every component has a reference to a top component, which again has a reference to a top component, and so on until the upper most component on the level directly under the assembly level. A useful qualitative modeling of the component hierarchy relies on the part-of relation. A mapping, generating the direct part-of relation between components would be as follows:

$partOf(x, y) \longleftarrow$ `SELECT f(CID) AS x, g(superCID) AS y FROM COMPONENT`

That the part-of relation is transitive can be declared by using a TBox axiom inducing further tuples in the part-of relation that are not directly visible in the SQL schema. So, if for example there are two ABox assertions $partOf(comp1, comp2)$, $partOf(comp2, comp3)$ induced by the mappings, then the transitivity condition entails $partOf(comp1, comp3)$. A Boolean query w.r.t. the ABox and the extended TBox asking whether $comp1$ is a part of $comp3$ would thus have to be answered with yes. A similar query could not directly be formulated in pure SQL, which does not support recursion.

As a resume, many use cases demand transitive roles, qualified existentials on the left-hand side and also disjunctions, hence there are more than one good reasons to consider more expressive logics such as $\mathcal{SHI}$ [44]. Of course, FOL rewritability does not hold for $\mathcal{SHI}$ ontologies, so a different strategy has to be invented in order to deal with really large ABoxes. A promising approach is ABox modularization [79], which uses relevant information from the TBox in order to built small modules of ABoxes. The relevant TBox information is stored in a data structure called $\forall$-info structure. This information is propagated along the role edges of the ABox as long as necessary. Modules consist of subgraphs induced by an individual constant and those individuals that are reached by the propagation.

The benefit of the modularization approach is that reasoning services such as instance retrieval (asking queries of the form $C(x)$) over the whole ABox can be reduced to a small set of ABox modules, which in turn are small in many practical applications. In [58], the approach is extended to the reasoning service of answering grounded CQs, i.e., CQs where the existentials adhere to the active domain semantics.

5 Temporalizing OBDA

Inspecting the whole life cycle of the query from being issued by a user until its being answered by an OBDA query answering system hints to the possible components for extensions w.r.t. temporal or streaming aspects. Though this observation does not say anything about which components' extension is appropriate for which use case, it provides a good aid which may help in categorizing the different approaches. So, the approaches may be distinguished w.r.t. the extensions of the TBox language, the ABox language, the query language, the language of the mappings, and the presupposed language of the data sources. In all this cases, the extension concerns syntactical aspects and also semantical aspects as well as ontological aspects in the original philosophical sense.

Processing time-dependent data has been investigated for a long time. Starting from early approaches in computer vision [61,62,72], *temporal data processing* using declarative techniques has been an important research topic in artificial intelligence (e.g., [50,51,41,40,13,9]). All these approaches use some form of temporal logic. Here we are going to discuss shortly the relevant notions for temporal extensions of OBDA based on the main ideas of temporal logics and then give pointers to the state of the art for temporal OBDA. In later sections we go more into detail w.r.t. the processing of temporal streams.

The simplest way to deal with temporal aspects is to refer to time points just with a simple attribute (as, e.g., done in the temporal extension of OWL called tOWL [57]). This is the most conservative strategy where the extension of non-temporal OBDA concerns only the domain of the models with objects to which time is attributed. Taking, e.g., the measurement scenario, a time attribute is attached to measurement (and message event) objects. Consider the following mapping producing assertions that describe measurements, their attached measured values and the times attached to them.

$$\left\{\begin{array}{c} meas(x) \wedge \\ val(x,y) \wedge \\ time(x,z) \end{array}\right\} \longleftarrow \begin{array}{l} \texttt{SELECT f(MID) AS m, Mval AS y, MtimeStamp AS z} \\ \texttt{FROM MEASUREMENT} \end{array}$$

The mapping is a classical mapping and the assertions induced by it are ordinary ABox assertions. The crucial point is that there have to be proper objects that reify temporal events (there was a something measured at that and that time). This is discussed in the literature under the term *temporal reification* (e.g. [2]).

Though reification means less adaptation needs, it leads to less control on the time aspects, as these are hidden in some objects of the domain. Moreover, reification has in many cases a higher ontological commitment than in the measurement scenario, where the presumption of measurements objects is plausible. For a discussion of these points we refer the reader to [38].

Non-reified extensions of OBDA consider time as a necessary dimension in the model semantics for assertions. Time is modelled with a structure $(T, \leq)$, also called *flow of time*. Depending on the needs of the use case, the set of time points T may have different properties, being discrete, such as the natural numbers, being dense, such as the rational numbers, being (left, right) bounded, being continuous such as the real numbers, being linear vs. branching etc. For an in-depth discussion of axiomatization and model theoretic aspects of the time domain the reader is referred to the early monograph [12].

A very general way to combine the flow of time into a logic is described in the literature on first-order temporal logic [43]. Instead of considering one single interpretation, the semantics rests now on a family of interpretations $\mathcal{I}_t$, one for each time point $t \in T$ on which a formula can be evaluated.[1] In many applications, it is assumed that the domains of all $\mathcal{I}_t$ are the same and that the individual constants are rigid, i.e., interpreted with the same object in all $\mathcal{I}_t$.

The decision which semantical objects are becoming temporal (non-sentential objects such as roles and concept as done in [8] vs. sentential objects such as ABox and TBox axioms (as done implicitly in [15] or explicitly in [54]) differs w.r.t. the needs of the use case. In first-order temporal logic one can refer directly to the time points and express further conditions, in the more state oriented approach of modal temporal logics, it is possible to talk about the validity of a formula in some state, referring to other states by modal operators such "In some state before ", "In some state in the future" etc.

Referring to the time argument as a time tag, one could formulate that the assertion of a sensor s_0 showing the value 90° is true at second 3 as the timestamped ABox assertion $val(s_0, 90°)\langle 3s \rangle$. Such an ABox assertion would be true in the given family of interpretations $(\mathcal{I}_t)_{t \in T}$ if $\mathcal{I}_{3s} \models val(s_0, 90°)\langle 3s \rangle$.

Approaches for temporalizing OBDA in the very narrow sense can be found in particular in [77], [68], [39], [54], all describing temporal extensions for RDF triples. In [54], the W3C recommend query language SPARQL is extended to work with RDF quadruples, where the fourth component is not only a time point but may be an interval, the intended meaning being that the interval describes the time period at which the fact expressed in the RDF triple is valid.

Two recent examples for temporalizing classical OBDA are described in [15] and [8]. The authors of [15] use a classical temporal logic inspired approach. The TBox is a classical DL-Lite TBox, in which all axioms are assumed to hold at all time points. The temporalized ABox is a finite sequence of pure DL-

[1] An alternative approach is to extend one interpretation to a two-sorted interpretation with a sort for the objects of the domain and a sort for the time points, cf. [43].

Lite ABoxes. The real temporal extension concerns the query language TCQ, which is a combination of embedded CQs with an outer propositional linear temporal logic template (LTL). To illustrate TCQ, assume, e.g. that there is an information need for turbines that have been at least two times in a critical situation the last three time units before, one would formulate a CQ $critical(x, y)$ formalizing the critical property and use previous operators $\bigcirc^{-1}$ and the some time in the future operator $\diamond$ within an LTL outer template given the TCQ denoted $Q(x, y)$.

$$Critical(x, y) = Turbine(x) \land showsMessage(x, y) \land FailureMessage(y)$$
$$Q(x, y) = \bigcirc^{-1} \bigcirc^{-1} \bigcirc^{-1}(\diamond(Critical(x, y) \land \bigcirc\diamond Critical(x, y)))$$

The authors [15] extend the notion of rewritability to sequences of ABoxes, which is in essence local rewritability w.r.t. each time point, and demonstrate two different algorithms for answering queries. Mapping aspects are not discussed in the article.

In the approach of [8], the temporal extension mainly concerns the TBox language, which allows to use modal logical operators in the front of concept descriptions and role descriptions. For example, on can express the fact that if a turbine shows an anomaly at some time, then some time in the future it will shut down itself:

$$showsAnomaly \sqsubseteq \diamond UnplanedShutDown$$

ABox assertions are extended with a time argument, so that one can formulate $val(s_0, 90°, 3s)$; the query language is an UCQ language, where the atoms have also time arguments and where one can quantifiy over the time argument and also formulate constraints using the ordering relation on the time flow. For example, one may ask whether there was a time between 3s and 6s where some turbine showed an anomaly: $\exists x \exists t.3s \leq t \leq 6s \land showsAnomaly(x, t)$. The authors show, that under some completeness assumption regarding ABox assertions, FOL rewritability for consistency checking and query answering holds. As in [15], mapping and unfolding aspects are not discussed.

Research on temporalizing ABDEO is in the beginning and still there have to be invented approaches that do 1) temporal query answering over very expressive TBoxes and 2) very large ABoxes that are 3) either virtually constructed or at least materialized w.r.t. some set of declarative mappings. So we can only mention approaches that fulfill a subset of the three conditions above. Similar to [15], the approach of [10] uses LTL operators in the query language but this time using TBoxes in the more expressive DL $\mathcal{ALC}$. Mappings and the largeness of ABoxes are not discuessed. A general overview of temporal DLs can be found in [7]. OWL related temporalizations are discussed in [42,67,60,57].

6 Stream Processing

Stream processing has strong connections to the processing of temporal data in a temporal DB using temporal (logic) operators; nonetheless, the scenarios, the objects of interest, the theoretical and practical challenges are different from those of temporal logics/ temporal DBs. While query answering on temporal DBs is a one-step activity on static historical data, answering queries on streams is a reactive, continuous activity (hence the notion of a continuous query).

A challenging fact of stream processing is the high frequency of the data in some application contexts. A second challenging fact is the changing frequency (burstiness) of incoming data, which is quite a natural phenomenon for event messages, which are produced in the moment the related events occur. Furthermore, in contrast to temporal DBs settings, stream scenarios come up with multiple queries as there may be many streams corresponding, e.g., to different sensors and control units, and as there are many features that one wants to query, such as different statistical and time-series features.

6.1 Stream Definition

Though there exist various stream definitions over various research communities, and even even within researcher of the same community, a common aspect of all streams is that they are constituted by a (potentially infinite) sequence of data elements from some domain. The sequence is thought to be ordered-isomorphic to the natural numbers, so that there is always a least element of a subset of the stream (and additionally there is always a unique predecessor and successor of an element in the stream). Following usual notation, we will represent streams in set notation, it being understood that there is an ordering isomorphic to the ordering of the natural numbers. We denote this ordering by $\leq_{ar}$, where ar stands for "arrival".

Though not restricted to, temporal streams are one of the most common stream types to occur in applications. A *temporal stream* is defined to be a set of timestamped domain elements $d\langle t\rangle$. The first argument is instantiated by an object d from a domain D, which we call the domain of streamed objects. The second argument is instantiated by a timestamp t from a linear time flow $(T, \leq)$, i.e., $\leq$ is a transitive, reflexive, antisymmetric, and total order on T. In the examples introduced below, we work with a discrete time flow T, e.g., natural numbers with the usual ordering. But our model introduced is also applicable to dense time domains such as the rational numbers $\mathbb{Q}$ or even continuos time domains such as the the real number $\mathbb{R}$, the crucial point being the fact, that streams have the isomorphism-type of the natural numbers.

According to the definition, a temporal stream may contain two different objects d_1 and d_2 with the same timestamp t, and even many different occurrences of the same time tagged tuple $d\langle t\rangle$. Moreover, it may be the case that there are timestamps which do not occur in the stream; the latter is useful in particular for those situations where there is no information at some time point from T—and hence this representation is also useful to model varying frequencies of stream element occurrences.

The elements of the stream are thought to be arriving at the query answering system in some order, which is inherent in the definition of streams as sequences. In a *synchronized stream setting*, one demands that the timestamps in the arrival ordering make up a monotonically increasing sequence, so that $\leq$ is conform with $\leq_{ar}$. In an *asynchronous stream setting*, it may be the case that elements with earlier timestamps arrive later than data with later timestamps. In particular, in this latter case, the order needed for a stream to be isomorphic to the natural numbers does not have to adhere to the order $\leq$ of the flow of time $(T, \leq)$.

The distinction regarding synchronicity hints to possible layers of streams. For example, in [53], the authors distinguish between raw, physical, and logical streams. Raw streams correspond to temporal streams according to our definition and are intended to model the streams arriving at a data stream management system (DSMS). Logical streams are abstractions of raw streams where the order of the sequence is ignored, so they can be defined as multi-sets of timestamped tuples.[2] Physical streams allow not only for timestamps but also half-open intervals $[t_s, t_e)$ as time annotations, the ordering of the stream being non-decreasing w.r.t. the start timestamps t_s.

The semantics for the relational stream query language CQL [5] is defined on the basis of synchronized streams. The new query language STARQL (see below) in its full version also allows asynchronous streams. But here for ease of exposition, we will assume that STARQL operates on synchronous streams, and even simpler logical streams (as we do not discuss sequence depending operators in this paper). It should be stressed, that though a layered approach as that of [53], where all synchronization is handled on the levels below, is in principle adequate it is not a must when designing a stream processing system on the abstraction level of ontologies. Regarding the synchronicity aspect, e.g., it is a matter of flexibility to give also the user of the ontological query language a means to specify the way he wants to handle asynchronous streams directly, and even doing the specification for each stream query—independently of the other queries.[3]

Orthogonally to these layered distinction of stream types, streams are categorized according to the type of the domains. In the context of OBDA for streams, at least two different domains D of streamed objects have to be taken into consideration. The first domain consists of relational tuples from some schema; in this case, we call the stream a *relational* stream. The second domain is made up by data facts, either represented as ABox assertions or as RDF triples. In this case we just talk of a stream of ABox assertions, RDF streams resp. In case of relational streams all tuples adhere to the same schema, hence relational streams are homogeneous, whilst in the case of streams of ABox assertions/RDF triples the used logical vocabulary must not be restricted to some relation symbols. In so far, these streams are inhomogeneous. Nonetheless, one may restrict the signature of the assertions to a specific signature, thereby replacing the role of the relational schema in relational streams by a signature.

[2] Note that the original definition in [53] would also consider uncountable sets as streams, if one chooses $\mathbb{R}$ as time domain, so that the intuition of a stream as a set ordered as the natural numbers cannot be applied here.

[3] A possibility for this is the use of a slack parameter. And indeed STARQL as defined in the technical report [66] is intended to handle also these.

6.2 Query Constructors over Streams

The idea of a sliding window over a stream is a useful constructor abstraction for processing streams. In the following, we are going to discuss the window operator and more generally the usual constructs for query languages over relational streams—focussing on one of the early relational query languages, the continuous query language CQL [5]. As a side effect of this focussing strategy we will have laid the ground for understanding the unfolding method for transforming STARQL queries into CQL queries.

CQL is a query language for relational streams. Next to streams, CQL presupposes a data structure called *relation*; this data structure lifts relations of classical relational algebra to the temporal setting; formally, let be given a classical relational schema, then a (temporal) relation R is a total function from the time domain T to the set of finite bags (multi-sets) of tuples over the given schema. The (classical) relation at time point $t \in T$ is called *instantaneous relation.*

CQL defines operators that map streams to relations (the most important being the window operator), operators that map relations to streams, and moreover relation-to-relation operators, which adapt the usual relational algebra operators to the temporal setting. Stream-to-stream operators can be simulated by the given operators.

The most important stream-to-relation construct is that of a sliding window. Let S denote a stream and wr be an element of the time domain T. The time-based sliding window operator with window range wr is denoted by `[Range wr]` and attached to stream arguments in post-fix notation. The relation R denoted by `S [Range wr]` is defined for every $t \in T$ as follows:

$$R(t) = \{s \mid (s, t') \in S \text{ and } (t' \leq t) \text{ and } t' \geq max\{t - wr, 0\}\}$$

So the bag of tuples at time point t consists of all tuples from S whose timestamps are in the interval $[t - wr, t]$—with an intuitive handling of the cases of all t with $t \leq wr$. The special case of a window with zero window range is also denoted by `[Now]`; the case of an unbounded window by `[Unbound]`.

The following example illustrates the effects of time sliding windows. Let be given a stream S of timestamped tuples having the form $(sensor, value)\langle time\rangle$. The smallest time granularity of time measurements is seconds, so we can presuppose that T is given by the natural numbers standing for time points measured in seconds. Now let the stream S start as follows:

$$\begin{aligned} S = \{&(s_0, 80°)\langle 0\rangle, (s_1, 93°)\langle 0\rangle, (s_0, 81)\langle 1\rangle, (s_0, 82°)\langle 2\rangle, (s_0, 83°)\langle 3\rangle, \\ &(s_0, 85°)\langle 5\rangle, (s_0, 86°)\langle 6\rangle....\} \end{aligned}$$

Then the relation $R =$ `S [Range 2]` is given by:

$t:$	0	1	2	3	4	5	6
$R(t):$	$\{(s_0,80),$ $(s_1,93)\}$	$\{(s_0,80),$ $(s_1,93),$ $(s_0,81)\}$	$\{(s_0,80),$ $(s_1,93),$ $(s_0,81),$ $(s_0,82)\}$	$\{(s_0,81),$ $(s_0,82),$ $(s_0,83)\}$	$\{(s_0,82),$ $(s_0,83)\}$	$\{(s_0,82),$ $(s_0,83),$ $(s_0,85)\}$	$\{(s_0,83),$ $(s_0,85),$ $(s_0,86)\}$

Please note, that the bag-approach for defining the relation is not unproblematic; the bags are timestamp agnostic, i.e., the tuples within a bag do not contain timestamps anymore. Formulating functionality constraints then becomes a demanding issue as these are not formulated w.r.t. a specific schema but directly over the domain. Take, for example, in a measurement scenario the functionality dependency (fd) constraint on measurement tuples with the content that a sensor can have maximally one value for every time point, resp. Applying a window operator to the stream measurement may lead to a bag of tuples where the same sensor has more than one value. So, streaming these tuples out again (with e.g. the RStream operator below) may lead to a stream of tuples violating the fd constraint.

A generalized version of the sliding window allows the specification of the sliding parameter, i.e. the frequency at which the window is slided forward in time. The semantics of the relation `R = S [Range wr Slide sl]` is as follows:

$$R(t) = \begin{cases} \emptyset & \text{if } t < sl \\ \{s \mid (s,t') \in S \text{ and } max\{\lfloor t/sl \rfloor \cdot sl - wr, 0\} \leq t' \leq \lfloor t/sl \rfloor \cdot sl\} & \text{else} \end{cases}$$

So the sliding window operator takes snapshots of the stream every sl second; for all time points in between multiples of sl the snapshot is the same. The relation of the window range parameter wr and the slide parameter sl determines the effects of the sliding window; if t$wr = sl$, then the window is tumbling, if $wr > sl$, then the window contents are overlapping, an if $sl > wr$, then the operator has a sampling effect. For other stream-to-relation operators such as the tuple based window operators or partition operators, the reader is referred to the original article [5].

Relation-to-relation operations are the usual ones known form SQL adapted to the new temporal relation setting by using them time-point wise on the instantaneous relations. In particular, the join of (temporal) relations is done pointwise. For example, the CQL expression in Listing 1.1 joins the relations of the window application results to a measurement and an event stream, filters a subset out according to a condition and projects out the sensor IDs.

As relation-to-stream operators, the authors of [5] define *Istream* (giving a stream of newly inserted tuples w.r.t. the last time point), *Dstream* (giving a stream of newly deleted tuples w.r.t. the last point) and *Rstream* (returning all elements in the relation). Assuming that for $t < 0$ one specifies $R(t) = \emptyset$ for

```
SELECT m.sensorID
FROM Msmt[Range 1] as m, Events[Range 2] as e
WHERE m.val > 30 AND e.category = Alarm AND
      m.sensorID = e.sensorID
```

Listing 1.1. Example relation-to-relation operators in CQL

$t < 0$ the formal definitions are:

$$
\begin{aligned}
Istream(R) &= \bigcup_{tinT} (R(t) \setminus R(t-1)) \times \{t\} \\
Dstream(R) &= \bigcup_{t \in T} (R(t-1) \setminus R(t)) \times \{t\} \\
Rstream(R) &= \bigcup_{t \in T} R(t) \times \{t\}
\end{aligned}
$$

6.3 Streamifying OBDA

Streamified OBDA approaches, in which the OBDA is to be understood in the classical sense, presuppose some stream engine on the level below ontologies which are accessed by mappings. So we first give an overview of the non-ontological stream processing systems and then discuss higher level (OBDA) stream processing.

Within stream processing on the level below ontologies two different perspective exist, the sensor perspective (semantic sensor networks) and the database perspective used in data stream management systems (DSMS). The sensor perspective of stream-based data processing pursues the idea of pushing data processing to the sensors and, as a consequence, investigates approximations for statistical data analysis operations in order to cope with memory and power limitations of sensors [29]. Query languages for sensor networks are investigated in [16,37,36,35]. Also here, data processing is pushed to the sensor level.

Stream processing on the level of data management systems has been an active research field which emerged ten years ago. A well known stream data management system is the academic prototype STREAM from Stanford which is based on the relational stream query language CQL [5] discussed above. Other academic data stream management systems (DSMS) with SQL like query languages are TelegraphCQ (Berkeley) [26], Aurora/Borealis (Brandeis, Brown and MIT) [45], or PIPES (Marburg University) [52,24,25,23,53]. To complete the list with commercial systems, we mention the stand-alone systems StreamBase, Truviso (extension of TelegraphCQ incorporated into Cisco products), or the stream add-ons for well known relational DBMSs (MySQL, PostgreSQL, DB2 etc.). Also PIPES now has a commercial successor developed by Software AG. Though much progress has been achieved on data stream management systems, there is still no agreement on a common (SQL) standard query language over streams. (Some first steps are discussed in [46]).

Much of the ideas and methods used in relational have inspired the recent streamified OBDA systems. Various systems extending the SPARQL language `http://www.w3.org/TR/rdf-sparql-query/` have been developed—prominent ones being, next to C-SPARQL, the SPARQL$_{stream}$ system [17,18], or CQELS [69,70]. C-SPARQL and SPARQLstream use a black box approach w.r.t. the underlying streaming engine, whilst CQELS has whitebox approach.

All these systems use a window operator inspired from CQL, where the window's semantics is a bag semantics with tuples of bindings for the query, subqueries resp. Hence, as the time attribute is not contained within the tuples all the approaches have to cope with similar (functional) constraint issues as discussed for CQL.

The SRBenchmark [81] and also the benchmarks provided in [55] show that the 2012 versions of the stream engines offer only basic functionalities. In particular w.r.t. the OBDA related properties one may state that only C-SPARQL is attributed to incorporate entailments of the TBox represented in an RDFS+ ontology. Only [17,18] discusses mappings, resulting in a new mapping language S2O, which is applied to map low sensor streams (using the source language Snee [35]) to RDF streams. But the exact semantics of the produced time stamps in the RDF streams is not laid out in [18], so that it is not clear how a time stamped tripled would interact with a temporal ABox and lead to further entailments. In contrast to the use of time as in STARQL (see below), time is referred to in a reified manner.

Somewhat related to streamified OBDA but at least clearly to be coined as high-level stream processing approaches are those using ideas of *complex event processing*. Examples are EP-SPARQL/ETALIS [4,3], T-REX using the event specification language TESLA [30,31,32], or Commonsens [76] and its open source successor ESPER (`esper.codehaus.org`). The approach by [56] is more on the OBDA line; it tries to compute predictions on streams (using autocorrelation of ontologies) over the lightweight description logic $\mathcal{EL}$; query answering is not in the focus of the approach.

7 A New Stream Temporal Query Language: STARQL

STARQL (Streaming and Temporal ontology Access with a Reasoning-based Query Language, pronounced Star-Q-L) is a query language framework in the intersection of classical OBDA, ABDEO and OBSA. Considering the fact that there are already many streaming languages, that, more or less, fit to the OBDA paradigm for query answering over streams [33,17,18,69], it is a justified question why to define another query language?

All of the mentioned approaches for stream processing on the RDF level model the window contents as a multi-set (bag) of variable bindings for the open variables in the query. But this solution has three main problems. First, the semantics based on the variable-binding solution presupposes mixed interim states in which the constraints and consequences of the ontologies (in particular the inconsistencies) are faded out. Second, such solutions do not adhere to the requirements of an orthogonal query language, according to which the inputs and interim-outputs are data structures in the same categories. And last but not least, if one considers KBs that allow for the formulation of persistency assumptions—which should be possible in the ABDEO paradigm—then one has to keep track of the time points in the window operators, as facts on previous time points may lead to consequences on later time points. For example, if a female person gives birth to a child at some time point, then it is a mother at all following time points.

The resume of these observations is that there is a good justification for defining a new query language and semantics with a necessary extension on the window concepts, fitting better into the OBDA/ABDEO paradigms. The main idea of STARQL, described in detail in the technical reports [66,64], is to provide ABox sequence constructors that group timestamped ABox assertions into ABoxes. The sequence sets up a nearly standard context in which OBDA (or ABDEO) reasoning services can be applied.

7.1 An Example From the Measurement Scenario

We are going to illustrate the STARQL constructs in a sensor measurement scenario where a stream S_{Msmt} of timestamped ABox assertions gives the values of a sensor s_0. The initial part up to second 5 is denoted $S_{Msmt}^{\leq 5}$.

$$\begin{aligned} S_{Msmt}^{\leq 5} = \{ & \mathit{val}(s_0, 90^\circ)\langle 0s\rangle, \mathit{val}(s_0, 93^\circ)\langle 1s\rangle, \mathit{val}(s_0, 94^\circ)\langle 2s\rangle \\ & \mathit{val}(s_0, 92^\circ)\langle 3s\rangle, \mathit{val}(s_0, 93^\circ)\langle 4s\rangle, \mathit{val}(s_0, 95^\circ)\langle 5s\rangle\} \end{aligned}$$

The stream may be materialized or virtual. The latter case will be illustrated below with mappings for timestamped ABox assertions from the stream query language CQL generating the upper level stream.

Assume that an engineer wants to express his information need for the fact whether the temperature value in sensor s_0 grew monotonically in the last 2 seconds, the information being updated every one second. In STARQL this can be formulated as follows.

```
CREATE STREAM S_out_1 AS
SELECT { s0 rdf:type RecentMonInc }<NOW>
FROM S_Msmt [NOW-2s, NOW]->1s
SEQUENCE BY StdSeq AS SEQ
HAVING
  FORALL i < j IN SEQ,?x,?y:
  IF ({ s0 val ?x }<i>  AND { s0  val ?y }<j>) THEN ?x <= ?y
```

Listing 1.2. Basic STARQL example

STARQL uses a mixed SQL and domain calculus notation for the realization of the information need. The window operator works like the window operator of CQL, but in one important point it differs from it: it does not delete the timestamps of the incoming ABox assertions. Moreover, the window operator it is syntactically represented with a suggestive interval notation containing a variable `NOW` for the evolving time. The slide parameter of one second is given by the forward arrow notation `->`. The output of the query is a stream of RDF tuples `{ s0 rdf:type RecentMonInc }<NOW>` (or written as ABox assertion $RecentMonInc(s_0)\langle NOW\rangle$) where the evolving time variable `NOW` is instantiated with the actual time.

The output of the window operators is a stream of temporal ABoxes, i.e., for every time point in the time domain T, here the natural numbers, one has a set of timestamped ABox assertions (resp. timestamped RDF tuples.) The result is illustrated in Figure for the time up to second 5.

Time (in seconds)	Temporal ABox
$0s$	$\{val(s_0, 90°)\langle 0s\rangle\}$
$1s$	$\{val(s_0, 90°)\langle 0s\rangle, val(s_0, 93°)\langle 1s\rangle\}$
$2s$	$\{val(s_0, 90°)\langle 0s\rangle, val(s_0, 93°)\langle 1s\rangle, val(s_0, 94°)\langle 2s\rangle\}$
$3s$	$\{val(s_0, 93°)\langle 1s\rangle, val(s_0, 94°)\langle 2s\rangle, val(s_0, 92°)\langle 3s\rangle\}$
$4s$	$\{val(s_0, 94°)\langle 2s\rangle, val(s_0, 92°)\langle 3s\rangle, val(s_0, 93°)\langle 4s\rangle\}$
$5s$	$\{val(s_0, 92°)\langle 3s\rangle, val(s_0, 93°)\langle 4s\rangle, val(s_0, 95°)\langle 5s\rangle\}$

Fig. 3. Temporal ABoxes produced by the window operator

So far, the STARQL query does not quite differ from the stream extended SPARQL query languages CSPARQL, SPARQL$_{Stream}$ or CQELS. The main difference is the new `SEQUENCE BY` constructor, which at every time point t merges the assertions in the temporal ABox for time point t according to a method given by keyword directly after the constructor, here `StdSeq`, which denotes the standard sequencing method. The standard sequencing method gathers all ABox assertions with the same timestamp into the same (pure) ABox.

The result, here and for other sequencing strategies, is a sequence of pure ABoxes at every evolving time point `NOW`. In the example above the sequence at time point 5 seconds is depicted in Fig. 7.1. In this case, the sequencing is trivial

Time (in seconds)	ABox sequence (with time stamp annotated ABoxes)
$5s$	$\{val(s_0, 92°)\}\langle 3s\rangle, \{val(s_0, 93°)\}\langle 4s\rangle, \{val(s_0, 95°)\}\langle 5s\rangle$

Fig. 4. Sequence of ABoxes at time point 5s

as there is only one ABox assertion for the timestamps 3s, 4s, 5s, resp.

In STARQL the ABoxes in the sequence are not represented by timestamps but state numbers in the order of the timestamps. These states can be referred to by specific variables (i, j above). In case of the ABox sequence above, the resulting ABox sequencing is depicted in Fig. 7.1. The reason for choosing the

Time (in seconds)	ABox sequence (with state annotated ABoxes)
$5s$	$\{val(s_0, 92°)\}\langle 0\rangle, \{val(s_0, 93°)\}\langle 1\rangle, \{val(s_0, 95°)\}\langle 2\rangle$

Fig. 5. Sequence of ABoxes at time point 5 with state annotation

state annotated representation is, first, that it provides the ground for incorporating temporal logics in the modal logic traditions such as LTL, which have proven to be useful specification checking tools. The semantics of modal temporal logics such as LTL are based on states. In particular, it is also possible to incorporate the LTL approach for temporal DL-Lite knowledge bases [15,14] into the `HAVING` clause fragment. (We note, here, that the semantics of the `HAVING` clause is slightly different from the semantics of the query language TCQ in [15]. But actually, the fragment of `HAVING` clauses that uses only operators of [15] is as expressive as TCQ.)

The second reason is that for other sequencing strategies the time annotations are not unique. For example, in STARQL we foresee sequencing strategies based on arbitrary equivalence relations over the time domain—with an additional constraint requiring that the equivalence classes are connected intervals. (So the equivalence relation can be thought of doing a time roughening.) All time points in an equivalence class are equally good candidates for the annotation hence there is no unique timestamp choice. Clearly, one can choose a canonical candidate (such as the left interval point) or even the whole interval by itself. But then one has to choose the `HAVING` language very carefully. Adding, e.g., time annotations to states in LTLs leads to highly complex logics known as metric temporal logics [59].

The expressive strength of STARQL lies in its `HAVING` clause language which allows to use FOL for querying the ABox sequence. In the example above, the formula realizes the known condition of monotonicity. The `HAVING` clause allows for embedding pure (non-temporal) CQs, here represented by binary graph patterns, and attaches a state to them. Here and in the following we call those queries occurring with a state tag *embedded conditions* and the language in which the are formulated the *embedded condition language.* In this example, the embedded conditions are $val(s_0, ?x)$ and $val(s_0, ?y)$ and the embedded condition language is UCQ.

The intended meaning of the expression $val(s_0, ?x)\langle i\rangle$ is that one wants to find all $?x$ that are certain answers of the query $Q = val(s_0, ?x)$ w.r.t. the i^{th} ABox—and also the TBox and the static ABox (see below)— that is, in the notation introduced above, one calculates $cert(Q, \mathcal{A}_i \cup \mathcal{A}_{static} \cup \mathcal{T})$. Similarly, $val(s0, ?y)\langle j \rangle$ finds the values $?y$ w.r.t. to the j^{th} ABox in the sequence. These steps in evaluating the `HAVING` clause are intra-ABox steps, as they are done locally w.r.t. (pure) ABoxes in the sequence. Though regarding the semantics intra-ABox query answering is just the one for UCQS over pure DL-Lite ontologies, the crucial difference is the fact that the ABoxes are not static but are updated dynamically. On the top of the intra-ABox step one has an inter-ABox step, which is realized by the FOL formula around the embedded conditions; in the case of this example, the FOL formula constrains the variables stemming from the intra-ABox evaluations to those fulfilling $?x <= ?y$.

Already in this example we see the subtleties of the `HAVING` clause language. For example, if we replaced $i < j$ by $i <= j$, then the `HAVING` clause would also test for every time point $i = j$ whether all values i are smaller or equal than all values at $j = i$, which amounts to saying that there can be at most one value for $s0$ at every state within the sequence.

The STARQL semantics, which we have sketched here is similar to the epistemic semantics of the query language EQL [21]: At every time point (state) i one considers only those bindings for which it is (certainly) known that they make the embedded CQs true. In contrast to EQL, in STARQL we have an explicit temporal domain, and we have an explicit safety mechanism for the formulas [65].

7.2 Extended Examples

The information need of the engineer may be more complex in that he asks for all temperature sensors that show a monotonic increase in the last two seconds. Temperature sensors are assumed to be declared in a static ABox, which is produced by classical mappings of the data sources (e.g. SQL databases). Temperature sensors found in the static ABox are delegated to the `HAVING` clause to specify the monotonicity condition. The sources to be used (such as the static ABox) are specified after keyword `USING`.

Due to possible inferences of the TBox together with the ABox, the query refers also to the TBox. For example, assume that the TBox contains the axiom $BurnerTipTempSens \sqsubseteq TempSens$. A complete answer to the STARQL query would first have to rewrite the embedded CQs. The embedded $TempSens(?s)$ would result in the new embedded $TempSens(?s) \vee BurnerTipTempSens(?s)$. This type of rewriting uses the rewriting technique appropriate for the underlying chosen embedded condition language, here perfect rewriting for UCQs. Perfect rewriting is possible, as in the framework of STARQL, the TBox is assumed to be a classical TBox without any additional temporal constructors. All subsumptions in the TBox hold at every time point.

If we were considering a different embedded condition language, for example grounded CQs within the ABDEO approach [58], then query answering could not be realized with rewriting but would have to use, e.g., an ABox modularization approach.

```
1  CREATE STREAM S_out_2 AS
2
3  SELECT  { ?s rdf:type RecentMonInc }<NOW>
4  FROM S_Msmt [NOW-2s, NOW]->1s
5  USING   STATIC ABOX <http://Astatic>,
6          TBOX <http://TBox>
7  WHERE { ?s:type TempSens }
8  SEQUENCE BY StdSeq AS SEQ
9  HAVING
10  FORALL i < j IN SEQ,?x,?y:
11  IF ({ ?s val ?x }<i>  AND { ?s  val ?y }<j>) THEN ?x <= ?y
```

Listing 1.3. STARQL example for using ABox and TBox

A more compact formulation of the extended monotonicity query uses a library entry for the monotonicity condition; after the keyword `CREATE AGGREGATE OPERATOR` is given the name of the library entry `monInc`.

```
1  CREATE STREAM S_out_3 AS
2
3  SELECT  { ?s rdf:type RecentMonInc }<NOW>
4  FROM S_Msmt [NOW-2s, NOW]->1s
5  USING   STATIC ABOX <http://Astatic>,
6          TBOX <http://TBox>
7  WHERE { ?s :type TempSens }
8  SEQUENCE BY StdSeq AS SEQ
9  HAVING monInc(SEQ, {?s val *})
10
11 CREATE AGGREGATE OPERATOR monInc(seq,  f(*)) AS
12   FORALL i <= j in SEQ,x,y:
13   IF   (f(x)<i>  AND  f(y)<j>) THEN x <= y
```

Listing 1.4. STARQL example for aggregation definition

The variables that can be selected in the `SELECT` line (line 3) are not restricted to those specified within the `WHERE` clause but may also refer to (open) variables in the `HAVING` clause. If, e.g., one wants to know at every second the value for the sensor s_0 in the last two seconds, than this can be queried in STARQL as depicted in 1.5.

Let us give an example with a more complex condition in the `HAVING` clause using a further constructor, multiple streams and the average operator (Listing 1.6). We are interested in those temperature sensors with a recent monotonic increase which are known to be in an operational mode (not service maintenance

```
CREATE STREAM S_out_4 AS
SELECT { s0 :val ?x }<NOW>
FROM S_Msmt [NOW-2s, NOW]->1s
SEQUENCE BY StdSeq AS SEQ
HAVING  EXISTS i IN SEQ ({ s0 val ?x }<i>
```

Listing 1.5. Using variables from the HAVING clause

mode) in the last 2s. Furthermore we want to get the average of the values in the last 2s. Next to the monotonicity condition we have a further condition using the FORALL operator. It evaluates the boolean condition in its scope on all ABoxes in the sequence and outputs the truth value true if the condition holds in all ABoxes.

```
CREATE STREAM S_out_5 AS

SELECT   { ?s rdf:type RecentMonInc }<NOW>,
         { ?s hasMean AVG(?sens val *) }<NOW>
FROM SMsmt NOW-2s, NOW]->1s
USING    STATIC ABOX <http://Astatic>,
         TBOX <http://TBox>
WHERE { ?s rdf:type TempSens }
SEQUENCE BY StdSeq as SEQ
HAVING  monInc(?s val *) AND
         FORALL i IN SEQ  { ?s rdf:type InOperationalMode }<i>
```

Listing 1.6. STARQL example for complex filter condition

The STARQL stream query language fulfills the desirable orthogonality property as it takes streams of timestamped assertions as input and produces again streams of timestamped assertions. This approach is motivated by the idea that getting answers to queries is not only an activity of GUI client programs (it is not only variable bindings that one wants to determine by queries), but query outcomes are going to be used as input to other queries as well as the generation of (temporal) ABox assertions in the application scenario itself. The expressions following the SELECT clause are templates for generating further timestamped ABox assertions (based on the intended interpretations within the query). The produced ABox assertions hold only within the output stream in which they are generated—and not universally. Otherwise we would have to handle recursion in queries—which, though possible using some kind of fixpoint operator, might lead to bad performance due to theoretically high complexity of the query answering problem. Hence the stream of ABox assertions generated by a query is in a different "category" than the assertions in the static ABox and the historical ABox. This is a kind of a locality principle.

Though the ABox assertions are limited to hold in the output streams, they may interact with the TBox, leading to entailed assertions. Assume that the TBox contains the following axiom stating that a sensor with a recent monotonic increase is a sensor in a critical mode:

$$RecMonInc \sqsubseteq Critical$$

The engineer could ask for those temperature sensors in a critical mode at every second on the stream S_{out_3} generated by one of our queries above. The components at which the temperature sensors had been in a critical state are declared as those ones that have to be removed in the next service maintenance in 10^4 seconds (see Listing 1.7).

```
CREATE STREAM S_out_6 AS

SELECT { ?comp removeDueToSensor  ?s}<NOW + 10^4s>
FROM S_out_3  [NOW, NOW]->1s
USING   STATIC ABOX <http://Astatic>,
        TBOX <http://TBox>
WHERE { ?s rdf:type TempSens . ?sens partOf ?comp }
SEQUENCE BY StdSeq AS SEQ
HAVING EXISTS i IN SEQ { ?sens rdf:type Critical }<i>
```

Listing 1.7. STARQL example for scope locality

The query is evaluated on the stream S_{out_3} which contains assertions of the form $RecMonInc(sens)\langle t\rangle$. At every second only the actual assertion is put into the temporal ABox (window range = 0s) so that the sequence contains only a trivial sequence of length 1 (at most). The `EXISTS` operator just tests whether the condition $Critical(sens)$ holds in some ABox in the sequence. Now, the stream S_{out_3} does not contain any ABox assertions with the concept symbol $Critical$, but such assertions are entailed by assertions of the form $RecMonInc(sens)\langle t\rangle$ in S_{out_3} and the TBox. Hence, the query above will indeed find those components that have to be removed due to a critical sensor. The example might use oversimplification but the reader should be able to understand the main idea.

Changing the stream from S_{out_3} to S_{Msmt} and thereby keeping everything else the same, will lead to a query which does not find any components—as the TBox and the assertions in S_{Msmt} do not entail assertions of the form $Critical(sens)$. Hence, the answers of a query really depend on the streams to which the queries are issued.

The general motivation of this approach is similar to the `CONSTRUCT` operator in the SPARQL query language, which provides the means to prescribe the format in which the bindings of the variables should be outputted. But in contrast, our approach also considers the outputs as ABox assertions that are part of an ontology. The idea of viewing the query head as a template for general ABox assertions was described already in the query language nRQL [80] coming with the Racer system.

7.3 Unfolding STARQL into CQL

We are going to illustrate the unfolding mechanism for the monotonicity example. Let be given a CQL stream of measurements $Msmt$, where the tuples adhere to the schema `Msmt(MID, MtimeStamp, SID, Mval)`. A mapping takes a CQL query over this stream and produces a stream of timestamped ABox assertions of the from $val(x, y)\langle t\rangle$.

$val(x, y)\langle z\rangle \longleftarrow$

```
          SELECT Rstream(f(SID) as x, Mval as y, MtimeStamp as t)
          FROM Msmt[NOW]
```

For this example we assume that s_0 is a shortcut for the compound individual constant $f(TC255)$, i.e., s_0 is the abstract representation of the sensor named TC255 in the data.

The stream query language STARQL follows a locality principle that allows to choose static ABoxes, TBoxes, and also the streams w.r.t. which the query is evaluated. This means in particular that the streams referred to in a query must be defined either by mappings or on a higher level by another STARQL query. So, assume that an input stream S is defined by the following mappings:

$$ax_1\langle t\rangle \leftarrow \Psi_1, ax_2\langle t\rangle \leftarrow \Psi_2, \ldots, ax_n\langle t\rangle \leftarrow \Psi_n$$

where for all i ax_i is an ABox assertion templates (in RDF speak: basic graph patterns) and Ψ_i is a CQL query containing all variables in ax_i within its Select head. The virtual stream consists of the time wise union of time tagged instantiations of the templates $ax_i\langle t\rangle$. Note, that this definition fixes only a logical stream. If one wants to work with streams having an arrival ordering, than one has either to fix the ordering according to some method or work with an indeterminism given by the query answering system which chooses the exact ordering. For all our examples, the exact arrival sequence is not relevant as we do not discuss stream operators that depend on the exact ordering. Hence, we can safely assume that the stream of timestamped ABox assertions/RDF tuples fixed by the mappings is a logical stream.

We assume that we have an input stream S_{msmt} defined exactly by the mapping above and take the basic monotonicity query in Listing 1.2.

The main problem for an unfolding strategy for STARQL queries is new data structure of an ABox sequence, which is not directly representable in CQL. A further demanding aspect is the fact that the `HAVING` clause is quite complex. Regarding the former, we therefore assume that the STARQL queries only use the standard sequencing, so that for every time point t_{NOW} from every state

i in the sequence associated with t_{NOW} one can reconstruct the timestamps of the tuples occurring in the ABox $\mathcal{A}_i$.

A second assumption is that all tuples in the input stream contain an attribute timestamp the value of which is the same as the value of the timestamp. Such a method is also discussed in [5], in order to do computations directly on the timestamps. So, we may assume a default stream-to-stream operator implemented into the CQL answering system and applied directly after every window-to-stream operator. It takes elements $d\langle t\rangle$ of a stream and returns an element $(d, t)\langle t\rangle$, thereby extending the schema of the tuples in the input stream by a time attribute for the output schema. If d already contains a time attribute, then it is overwritten by the new values. In the definition of the mapping for $val(x, y)\langle t\rangle$, we applied this assumption, where we refer to the time attribute `MtimeStamp` of the input stream `Msmt`. Note that this assumption mitigates the weakness of time-ignoring bag semantics in the window operators of CQL.

Regarding the complexity of the `HAVING` clause, the STARQL grammar (see [64,65]) uses a safety mechanism that restricts the use of variables by adornments. For example, in the `HAVING` clause $?y > 3$, the variable $?y$ is not safe, as the set of certain answers with bindings for $?y$ would be infinite. On the other hand, in $val(s_0, ?y)\langle i\rangle \wedge (?y > 3)$ the variable $?y$ is safe, as it is bounded by $val(s_0, ?y)\langle i\rangle$, which gives a finite set of bindings. Actually, safety has not only to guarantee the finiteness of the non-bounded variables but also the domain independence [1], as the target language CQL (as a SQL extension) is also domain independent. Domain independence ensures that the query language can be evaluated only on the basis of the constants in the query and the database/streams (thus using only the active domain) and does not have to incorporate the whole domain.

The safety conditions guarantee, that the `HAVING` clause can be transformed into a FOL formula which is in so called *safe range normal form.* Such formulas can further be transformed to relational algebraic normal form (RANF), for which a direct translation into the relational algebra and so also into SQL is possible. This SQL query depends on the states in the sequence associated with the evolving time point t_{NOW}. But the association of states and timestamps is unique: we can refer to states i just by the timestamps stored in the attribute value of the tuple.

The `HAVING` clause in the STARQL query of Listing 1.2 is given in relational algebraic normal form in Listing 1.8. The whole unfolded CQL query for

```
NOT EXISTS  i,j in SEQ ?x,?y:
({ s0 val ?x }<i>  AND { s0  val ?y }<j>) AND  ?x > ?y
```

Listing 1.8. Normalized monotonicity condition

the STARQL query of Listing 1.2 is given in Listing 1.9. Here, the view windowRelation creates a temporal relation of measurements according to the time window specified in the mapping. The expression after the `RStream` constructs (as a string) the RDF triple as expected by the STARQL query.

```
CREATE VIEW windowRelation as
SELECT *  FROM Msmt[RANGE 2s Slide 1s];

SELECT
 Rstream('{ s0 rdf:Type RecMonInc }'||'<'||timestamp||'>' )
 FROM windowRel
 WHERE windowRel.SID = 'TC255'  AND
 NOT EXISTS (
  SELECT * FROM
  (SELECT timestamp as i, value as x FROM windowRelation),
  (SELECT timestamp as j, value as y FROM windowRelation)
  WHERE   i < j AND x > y );
```

Listing 1.9. Monotonicity STARQL query unfolded into CQL

8 Conclusion

Research on temporal and streamified OBDA has just begun. It can profit by the many ideas, formalisms, and techniques known from the vast literature on temporal logics and relational stream processing. But as the benchmarks in particular w.r.t. OBDA streaming engines [81] show, the theoretical formalization has still not settled—not to speak of the implementation and optimization aspects, in particular the demanding scalability issues for stream processing (number of continuous queries, number of streams, frequency, size of the static ABox etc.)

As part of a possible theoretical formalization, we introduced the STARQL query language that uses the crucial data structure of streams of ABox sequences. The ABox sequencing strategy can be considered as syntactic sugar only for those cases where the chosen sequencing strategy is as simple as that of standard sequencing. For non-standard sequencing strategies, such building only consistent ABoxes which are queried, a reduction to the standard window operator is not possible. As of now, STARQL is implemented and tested in the OPTIQUE project within a prototype that uses a stream extended version of ADP [47], a highly distributed data management system, as the data source to which STARQL queries are unfolded.

Next to the in-depth theoretical foundation of streamifying classical OBDA, further relevant research is going to be in the direction of streamifying and temporalizing ABDEO. The idea of modularization has to be adapted to handle fast importing of modules for intra-ABox reasoning, using time as a special modularization parameter.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley (1995)
2. Allen, J.F.: Towards a General Theory of Action and Time. Artificial Intelligence 23(2), 123–154 (1984)
3. Anicic, D., Fodor, P., Rudolph, S., Stojanovic, N.: Ep-sparql: a unified language for event processing and stream reasoning. In: Srinivasan, S., Ramamritham, K., Kumar, A., Ravindra, M.P., Bertino, E., Kumar, R. (eds.) WWW. pp. 635–644. ACM (2011)
4. Anicic, D., Rudolph, S., Fodor, P., Stojanovic, N.: Stream reasoning and complex event processing in etalis. Semantic Web 3(4), 397–407 (2012)
5. Arasu, A., Babu, S., Widom, J.: The CQL continuous query language: semantic foundations and query execution. The VLDB Journal 15, 121–142 (2006)
6. Artale, A., Calvanese, D., Kontchakov, R., Zakharyaschev, M.: The DL-Lite family and relations. J. Artif. Intell. Res. (JAIR) 36, 1–69 (2009)
7. Artale, A., Franconi, E.: A survey of temporal extensions of description logics. Annals of Mathematics and Artificial Intelligence 30(1-4), 171–210 (Mar 2001)
8. Artale, A., Kontchakov, R., Wolter, F., Zakharyaschev, M.: Temporal description logic for ontology-based data access. In: Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence. pp. 711–717. IJCAI'13, AAAI Press (2013)
9. Artikis, A., Skarlatidis, A., Portet, F., Paliouras, G.: Logic-based event recognition. Knowledge Eng. Review 27(4), 469–506 (2012)
10. Baader, F., Borgwardt, S., Lippmann, M.: Temporalizing ontology-based data access. In: CADE-13 (2013)
11. Baader, F., Nutt, W.: Basic description logics. In: Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P. (eds.) The Description Logic Handbook, pp. 43–95. Cambridge University Press (2003)
12. van Benthem, J.: The Logic of Time: A Model-Theoretic Investigation into the Varieties of Temporal Ontology and Temporal Discourse. Reidel, 2. edn. (1991)
13. Bohlken, W., Neumann, B., Hotz, L., Koopmann, P.: Ontology-based realtime activity monitoring using beam search. In: Crowley, J.L., Draper, B.A., Thonnat, M. (eds.) ICVS. Lecture Notes in Computer Science, vol. 6962, pp. 112–121. Springer (2011)
14. Borgwardt, S., Lippmann, M., Thost, V.: Temporal query answering in dl-lite. In: Eiter et al. [34], pp. 80–92
15. Borgwardt, S., Lippmann, M., Thost, V.: Temporal query answering in the description logic dl-lite. In: Fontaine, P., Ringeissen, C., Schmidt, R.A. (eds.) Frontiers of Combining Systems. LNCS, vol. 8152, pp. 165–180. Springer Berlin Heidelberg (2013)
16. Brenninkmeijer, C.Y.A., Galpin, I., Fernandes, A.A.A., Paton, N.W.: A semantics for a query language over sensors, streams and relations. In: Gray, W.A., Jeffery, K.G., Shao, J. (eds.) BNCOD. Lecture Notes in Computer Science, vol. 5071, pp. 87–99. Springer (2008)
17. Calbimonte, J.P., Corcho, O., Gray, A.J.G.: Enabling ontology-based access to streaming data sources. In: Proceedings of the 9th international semantic web conference on The semantic web - Volume Part I. pp. 96–111. ISWC'10, Springer-Verlag, Berlin, Heidelberg (2010)

18. Calbimonte, J.P., Jeung, H., Corcho, O., Aberer, K.: Enabling query technologies for the semantic sensor web. Int. J. Semant. Web Inf. Syst. 8(1), 43–63 (Jan 2012)
19. Calì, A., Gottlob, G., Lukasiewicz, T.: Datalog+/-: A unified approach to ontologies and integrity constraints. In: Proceedings of the 12th International Conference on Database Theory. pp. 14–30. ACM Press (2009)
20. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rodríguez-Muro, M., Rosati, R.: Ontologies and databases: The DL-Lite approach. In: Tessaris, S., Franconi, E. (eds.) Semantic Technologies for Informations Systems – 5th Int. Reasoning Web Summer School (RW 2009), Lecture Notes in Computer Science, vol. 5689, pp. 255–356. Springer (2009)
21. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Epistemic first-order queries over description logic knowledge bases. In: Proc. of the 19th Int. Workshop on Description Logics (DL 2006). CEUR Electronic Workshop Proceedings, http://ceur-ws.org/, vol. 189, pp. 51–61 (2006)
22. Calvanese, D., Giese, M., Haase, P., Horrocks, I., Hubauer, T., Ioannidis, Y.E., Jiménez-Ruiz, E., Kharlamov, E., Kllapi, H., Klüwer, J.W., Koubarakis, M., Lamparter, S., Möller, R., Neuenstadt, C., Nordtveit, T., Özçep, Ö.L., Rodriguez-Muro, M., Roshchin, M., Savo, D.F., Schmidt, M., Soylu, A., Waaler, A., Zheleznyakov, D.: Optique: Obda solution for big data. In: Proc. ESWC (Satellite Events). pp. 293–295 (2013)
23. Cammert, M., Heinz, C., Kramer, J., Seeger, B., Vaupel, S., Wolske, U.: Flexible multi-threaded scheduling for continuous queries over data streams. In: Data Engineering Workshop, 2007 IEEE 23rd International Conference on. pp. 624–633 (april 2007)
24. Cammert, M., Heinz, C., Krämer, J., Seeger, B.: Sortierbasierte joins über datenströmen. In: Vossen, G., Leymann, F., Lockemann, P.C., Stucky, W. (eds.) BTW. LNI, vol. 65, pp. 365–384. GI (2005)
25. Cammert, M., Krämer, J., Seeger, B., Vaupel, S.: An approach to adaptive memory management in data stream systems. In: Liu, L., Reuter, A., Whang, K.Y., Zhang, J. (eds.) ICDE. p. 137. IEEE Computer Society (2006)
26. Chandrasekaran, S., Cooper, O., Deshpande, A., Franklin, M.J., Hellerstein, J.M., Hong, W., Krishnamurthy, S., Madden, S., Raman, V., Reiss, F., Shah, M.A.: Telegraphcq: Continuous dataflow processing for an uncertain world. In: CIDR (2003)
27. Compton, M., Barnaghi, P., Bermudez, L., GarcÃa-Castro, R., Corcho, O., Cox, S., Graybeal, J., Hauswirth, M., Henson, C., Herzog, A., Huang, V., Janowicz, K., Kelsey, W.D., Phuoc, D.L., Lefort, L., Leggieri, M., Neuhaus, H., Nikolov, A., Page, K., Passant, A., Sheth, A., Taylor, K.: The {SSN} ontology of the {W3C} semantic sensor network incubator group. Web Semantics: Science, Services and Agents on the World Wide Web 17(0), 25–32 (2012)
28. Console, M., Horrocks, I., Jimenez-Ruiz, E., Kharloamov, E., Lenzerini, M., Rosati, R., Ruzzi, M., Santarelli, V., Savo, D.F., Soylu, A., Thorstensen, E., Zheleznyakov, D.: Deliverable D4.1 – WP4 Year 1 progress report (ontology and mapping management). Deliverable FP7-318338, EU (October 2013)
29. Cormode, G.: The continuous distributed monitoring model. SIGMOD Record 42(1), 5–14 (2013)
30. Cugola, G., Margara, A.: Tesla: A formally defined event specification language. In: Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems. pp. 50–61. DEBS '10, ACM, New York, NY, USA (2010)
31. Cugola, G., Margara, A.: Complex event processing with t-rex. Journal of Systems and Software 85(8), 1709–1728 (2012)

32. Cugola, G., Margara, A.: Processing flows of information: From data stream to complex event processing. ACM Comput. Surv. 44(3), 15 (2012)
33. Della Valle, E., Ceri, S., Barbieri, D., Braga, D., Campi, A.: A first step towards stream reasoning. In: Domingue, J., Fensel, D., Traverso, P. (eds.) Future Internet – FIS 2008, Lecture Notes in Computer Science, vol. 5468, pp. 72–81. Springer Berlin / Heidelberg (2009)
34. Eiter, T., Glimm, B., Kazakov, Y., Krötzsch, M. (eds.): Informal Proceedings of the 26th International Workshop on Description Logics, Ulm, Germany, July 23 - 26, 2013, CEUR Workshop Proceedings, vol. 1014. CEUR-WS.org (2013)
35. Galpin, I., Brenninkmeijer, C., Gray, A., Jabeen, F., Fernandes, A., Paton, N.: Snee: a query processor for wireless sensor networks. Distributed and Parallel Databases 29, 31–85 (2011), 10.1007/s10619-010-7074-3
36. Galpin, I., Brenninkmeijer, C.Y., Jabeen, F., Fernandes, A.A., Paton, N.W.: Comprehensive optimization of declarative sensor network queries. In: Proceedings of the 21st International Conference on Scientific and Statistical Database Management. pp. 339–360. SSDBM 2009, Springer-Verlag, Berlin, Heidelberg (2009)
37. Galpin, I., Brenninkmeijer, C.Y.A., Jabeen, F., Fernandes, A.A.A., Paton, N.W.: An architecture for query optimization in sensor networks. In: Alonso, G., Blakeley, J.A., Chen, A.L.P. (eds.) ICDE. pp. 1439–1441. IEEE (2008)
38. Galton, A.: Reified temporal theories and how to unreify them. In: Proceedings of the 12th international joint conference on Artificial intelligence - Volume 2. pp. 1177–1182. IJCAI'91, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1991)
39. Gutierrez, C., Hurtado, C., Vaisman, R.: Temporal rdf. In: In European Conference on the Semantic Web (ECSW' 05). pp. 93–107 (2005)
40. Heintz, F., Kvarnström, J., Doherty, P.: Stream-based reasoning support for autonomous systems. In: Coelho, H., Studer, R., Wooldridge, M. (eds.) ECAI. Frontiers in Artificial Intelligence and Applications, vol. 215, pp. 183–188. IOS Press (2010)
41. Heintz, F., Rudol, P., Doherty, P.: From images to traffic behavior - a uav tracking and monitoring application. In: FUSION. pp. 1–8. IEEE (2007)
42. Hobbs, J.R., Pan, F.: An ontology of time for the semantic web. ACM Transactions on Asian Language Information Processing 3(1), 66–85 (Mar 2004)
43. Hodkinson, I., Reynolds, M.: Temporal logic. In: Blackburn, P., van Benthem, J., Wolter, F. (eds.) Handbook of Modal Logic, vol. 6, chap. 11, pp. 655–720. Elsevier Science (2006)
44. Horrocks, I., Sattler, U.: A description logic with transitive and inverse roles and role hierarchies. Journal of Logic and Computation 9(3), 385–410 (1999)
45. Hwang, J.H., Xing, Y., Çetintemel, U., Zdonik, S.B.: A cooperative, self-configuring high-availability solution for stream processing. In: ICDE. pp. 176–185 (2007)
46. Jain, N., Mishra, S., Srinivasan, A., Gehrke, J., Widom, J., Balakrishnan, H., Çetintemel, U., Cherniack, M., Tibbetts, R., Zdonik, S.: Towards a streaming sql standard. Proc. VLDB Endow. 1(2), 1379–1390 (Aug 2008)
47. Kllapi, H., Bilidas, D., Ioannidis, Y., Koubarakis, M.: Deliverable D7.1: Techniques for distributed query planning and execution: One-time queries. Deliverable, Optique (2013)
48. Kontchakov, R., Lutz, C., Toman, D., Wolter, F., Zakharyaschev, M.: The combined approach to ontology-based data access. In: Walsh, T. (ed.) IJCAI. pp. 2656–2661. IJCAI/AAAI (2011)

49. Kostylev, E.V., Reutter, J.: Answering counting aggregate queries over ontologies of the DL-Lite family. In: Proceedings of the 27th AAAI Conference on Artificial Intelligence (AAAI-13), Bellevue, Washington, (2013)
50. Kowalski, R.A., Sadri, F.: Towards a logic-based unifying framework for computing. CoRR abs/1301.6905 (2013)
51. Kowalski, R.A., Toni, F., Wetzel, G.: Towards a declarative and efficient glass-box CLP language. In: WLP. pp. 138–141 (1994)
52. Krämer, J., Seeger, B.: A temporal foundation for continuous queries over data streams. In: Haritsa, J.R., Vijayaraman, T.M. (eds.) COMAD. pp. 70–82. Computer Society of India (2005)
53. Krämer, J., Seeger, B.: Semantics and implementation of continuous sliding window queries over data streams. ACM Trans. Database Syst. 34(1), 1–49 (Apr 2009)
54. Kyzirakos, K., Karpathiotakis, M., Koubarakis, M.: Strabon: A Semantic Geospatial DBMS. In: International Semantic Web Conference. Boston, USA (Nov 2012)
55. Le-Phuoc, D., Dao-Tran, M., Pham, M.D., Boncz, P., Eiter, T., Fink, M.: Linked stream data processing engines: Facts and figures. In: Cudre-Mauroux, P., Heflin, J., Sirin, E., Tudorache, T., Euzenat, J., Hauswirth, M., Parreira, J., Hendler, J., Schreiber, G., Bernstein, A., Blomqvist, E. (eds.) The Semantic Web–ISWC 2012, pp. 300–312. Lecture Notes in Computer Science, Springer Berlin Heidelberg (2012), http://dx.doi.org/10.1007/978-3-642-35173-0_20
56. Lecue, F., Pan, J.Z.: Predictive learning in sensor networks. Submitted to IJCAI
57. Milea, V., Frasincar, F., Kaymak, U.: tOWL: A Temporal Web Ontology Language. IEEE Transactions on Systems, Man and Cybernetics 42(1), 268–281 (2012)
58. Möller, R., Neuenstadt, C., Ozçep, O., Wandelt, S.: Advances in accessing big data with expressive ontologies. In: Timm, I.J., Thimm, M. (eds.) KI 2013: Advances in Artificial Intelligence. Lecture Notes in Computer Science, vol. 8077, pp. 118–129. Springer Berlin Heidelberg (2013), http://dx.doi.org/10.1007/978-3-642-40942-4_11
59. Montanari, A., Policriti, A.: Decidability results for metric and layered temporal logics. Notre Dame Journal of Formal Logic 37, 37–260 (1996)
60. Motik, B.: Representing and querying validity time in RDF and OWL: a logic-based approach. In: Proceedings of the 9th international semantic web conference on The semantic web - Volume Part I. pp. 550–565. ISWC'10, Springer-Verlag, Berlin, Heidelberg (2010)
61. Neumann, B., Novak, H.J.: Event models for recognition and natural language description of events in real-world image sequences. In: Bundy, A. (ed.) IJCAI. pp. 724–726. William Kaufmann (1983)
62. Neumann, B., Novak, H.J.: Noas: Ein system zur natürlichsprachlichen beschreibung zeitveränderlicher szenen. Inform., Forsch. Entwickl. 1(2), 83–92 (1986)
63. Noy, N.F.: Semantic integration: a survey of ontology-based approaches. SIGMOD Record 33(4), 65–70 (2004)
64. Özçep, O.L., Möller, R., Neuenstadt, C.: Obda stream access combined with safe first-order temporal reasoning. Technical report, Hamburg University of Technology (2014)
65. Özçep, O.L., Möller, R., Neuenstadt, C.: A stream-temporal query language for ontology based data access. To be published in Proceedings of the 7th International Workshop on Description Logics (2014)
66. Özçep, Ö.L., Möller, R., Neuenstadt, C., Zheleznyakov, D., Kharlamov, E.: Deliverable D5.1 – a semantics for temporal and stream-based query answering in an OBDA context. Deliverable FP7-318338, EU (October 2013)

67. Papadakis, N., Stravoskoufos, K., Baratis, E., Petrakis, E.G.M., Plexousakis, D.: Proton: A prolog reasoner for temporal ontologies in owl. Expert Syst. Appl. 38(12), 14660–14667 (Nov 2011)
68. Perry, M.: : A Framework to Support Spatial, Temporal and Thematic Analytics over Semantic Web Data. Ph.D. thesis, Wright State UNiversity (2008)
69. Phuoc, D.L., Dao-Tran, M., Parreira, J.X., Hauswirth, M.: A native and adaptive approach for unified processing of linked streams and linked data. In: Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N.F., Blomqvist, E. (eds.) International Semantic Web Conference (1). Lecture Notes in Computer Science, vol. 7031, pp. 370–388. Springer (2011)
70. Phuoc, D.L., Nguyen-Mau, H.Q., Parreira, J.X., Hauswirth, M.: A middleware framework for scalable management of linked streams. J. Web Sem. 16, 42–51 (2012)
71. Poggi, A., Lembo, D., Calvanese, D., Giacomo, G.D., Lenzerini, M., Rosati, R.: Linking data to ontologies. Journal of Data Semantics 10, 133–173 (2008)
72. Rist, T., Herzog, G., André, E.: Ereignismodellierung zur inkrementellen high-level bildfolgenanalyse. In: Buchberger, E., Retti, J. (eds.) ÖGAI. Informatik-Fachberichte, vol. 151, pp. 1–11. Springer (1987)
73. Rodriguez-Muro, M., Calvanese, D.: High performance query answering over dl-lite ontologies. In: Brewka, G., Eiter, T., McIlraith, S.A. (eds.) KR. AAAI Press (2012)
74. Rodriguez-Muro, M., Calvanese, D.: Quest, an owl 2 ql reasoner for ontology-based data access. In: Klinov, P., Horridge, M. (eds.) OWLED. CEUR Workshop Proceedings, vol. 849. CEUR-WS.org (2012)
75. Rodriguez-Muro, M., Kontchakov, R., Zakharyaschev, M.: Query rewriting and optimisation with database dependencies in ontop. In: Eiter et al. [34], pp. 917–929
76. Søberg, J., Goebel, V., Plagemann, T.: Deviation detection in automated home care using commonsens. In: PerCom Workshops. pp. 668–673. IEEE (2011)
77. Tappolet, J., Bernstein, A.: Applied temporal rdf: Efficient temporal querying of rdf data with sparql. In: Proceedings of the 6th European Semantic Web Conference on The Semantic Web: Research and Applications. pp. 308–322. ESWC 2009 Heraklion, Springer-Verlag, Berlin, Heidelberg (2009)
78. Valle, E.D., Schlobach, S., Krötzsch, M., Bozzon, A., Ceri, S., Horrocks, I.: Order matters! harnessing a world of orderings for reasoning over massive data. Semantic Web 4(2), 219–231 (2013)
79. Wandelt, S., Möller, R.: Towards abox modularization of semi-expressive description logics. Applied Ontology 7(2), 133–167 (2012)
80. Wessel, M., Möller, R.: A high performance semantic web query answering engine. In: Horrocks, I., Sattler, U., Wolter, F. (eds.) Description Logics. CEUR Workshop Proceedings, vol. 147. CEUR-WS.org (2005)
81. Zhang, Y., Minh Duc, P., Corcho, O., Calbimonte, J.P.: Srbench: A Streaming RDF/SPARQL Benchmark. In: Proceedings of International Semantic Web Conference 2012 (November 2012)