Accepted Manuscript

An ontology-mediated analytics-aware approach to support monitoring and diagnostics of static and streaming data

Evgeny Kharlamov, Yannis Kotidis, Theofilos Mailis, Christian Neuenstadt, Charalampos Nikolaou, Özgür Özçep, Christoforos Svingos, Dmitriy Zheleznyakov, Yannis Ioannidis, Steffen Lamparter, Ralf Möller, Arild Waaler

PII: S1570-8268(19)30001-0
DOI: https://doi.org/10.1016/j.websem.2019.01.001
Reference: WEBSEM 494

To appear in: *Web Semantics: Science, Services and Agents on the World Wide Web*

Received date : 20 October 2017
Revised date : 18 November 2018
Accepted date : 3 January 2019

Please cite this article as: E. Kharlamov, Y. Kotidis, T. Mailis et al., An ontology-mediated analytics-aware approach to support monitoring and diagnostics of static and streaming data, *Web Semantics: Science, Services and Agents on the World Wide Web* (2019), https://doi.org/10.1016/j.websem.2019.01.001

An Ontology-mediated Analytics-aware Approach to Support Monitoring and Diagnostics of Static and Streaming Data

Evgeny Kharlamov[a,b,c], Yannis Kotidis[d], Theofilos Mailis[e], Christian Neuenstadt[f], Charalampos Nikolaou[a], Özgür Özçep[f], Christoforos Svingos[e], Dmitriy Zheleznyakov[a], Yannis Ioannidis[e], Steffen Lamparter[g], Ralf Möller[f], Arild Waaler[b]

[a] *University of Oxford, Department of Computer Science, Wolfson Building, Parks Road, OX1 3QD, Oxford, UK.*
[b] *University of Oslo, Department of Computer Science, P.O. Box 1080 Blindern, N-0316 Oslo, Norway.*
[c] *Bosch Centre for Artificial Intelligence, Robert Bosch GmbH, Renningen, 71465 Stuttgart, Germany.*
[d] *Athens University of Economics and Business, 76 Patission Street, 10434, Athens, Greece.*
[e] *National and Kapodistrian University of Athens, Panepistimiopolis, Ilissia, 15784, Athens, Greece.*
[f] *University of Luebeck, Ratzeburger Allee 160, 23562, Lübeck, Germany.*
[g] *Siemens Corporate Technology, Siemens AG, Otto-Hahn-Ring 6, 81739, Munich, Germany.*

Abstract

Streaming analytics that requires integration and aggregation of heterogeneous and distributed streaming and static data is a typical task in many industrial scenarios including the case of industrial IoT where several pieces of industrial equipment such as turbines in Siemens are integrated into an IoT. The OBDA approach has a great potential to facilitate such tasks; however, it has a number of limitations in dealing with analytics that restrict its use in important industrial applications. We argue that a way to overcome those limitations is to extend OBDA to become analytics, source, and cost aware. In this work we propose such an extension. In particular, we propose an ontology, mapping, and query language for OBDA, where aggregate and other analytical functions are first class citizens. Moreover, we develop query optimisation techniques that allow to efficiently process analytical tasks over static and streaming data. We implement our approach in a system and evaluate our system with Siemens turbine data.

Keywords: Ontology Based Data Access, Data Integration, IoT, Streaming Data, Static Data, Optimisations, Siemens.

1. Introduction

Ontology Based Data Access (OBDA) [1, 2] is an approach to access information stored in multiple data sources via an abstraction layer that mediates between the data sources and data consumers. On the one hand, this layer uses an *ontology* to provide a uniform conceptual schema that describes the problem domain of the underlying data independently of how and where the data is stored. On the other hand, this layer uses declarative *mappings* to specify how the ontology is related to the data by associating elements of the ontology to queries over data sources. The ontology and mappings are used to *transform* queries over ontologies, i.e., *ontological queries*, into *data queries* over data sources. As well as abstracting away from details of data storage and access, the ontology and mappings provide a declarative, modular and query-independent specification of both the conceptual model and its relationship to the data sources; this simplifies development and maintenance and allows for easy integration with existing data management infrastructure.

In Figure 1 we present a conceptual architecture of classical OBDA where on the data layer there is static relational data. Mappings are used to connect the data to the ontology and access to the data is realised by means of data extraction queries posed over the ontology.

A number of systems that at least partially implement OBDA have been recently developed; they include D2RQ [3], Mastro [4], morph-RDB [5], Ontop [6], OntoQF [7], Ultrawrap [8], Virtuoso, Spyder, and others [9, 10]. Some of them were successfully used in various applications including cultural heritage [11], governmental organisations [12], and industry [13, 14, 15, 16].

Despite their success, OBDA systems are not tailored towards analytical tasks that are naturally based on data aggregation and correlation. Moreover, they offer a limited or no support for queries that combine streaming and static data. At the same time, such tasks would naturally benefit from OBDA as we illustrate next.

Email addresses: `evgeny.kharlamov@cs.ox.ac.uk` (Evgeny Kharlamov), `evgeny.kharlamov@ifi.uio.no` (Evgeny Kharlamov), `evgeny.kharlamov@de.bosch.com` (Evgeny Kharlamov), `kotidis@aueb.gr` (Yannis Kotidis), `theofilos@image.ntua.gr` (Theofilos Mailis), `neuenstadt@ifis.uni-luebeck.de` (Christian Neuenstadt), `babis.nikolaou@cs.ox.ac.uk` (Charalampos Nikolaou), `oezcep@ifis.uni-luebeck.de` (Özgür Özçep), `c.svingos@di.uoa.gr` (Christoforos Svingos), `dmitriy.zheleznyakov@cs.ox.ac.uk` (Dmitriy Zheleznyakov), `yannis@di.uoa.gr` (Yannis Ioannidis), `steffen.lamparter@siemens.com` (Steffen Lamparter), `moeller@ifis.uni-luebeck.de` (Ralf Möller)

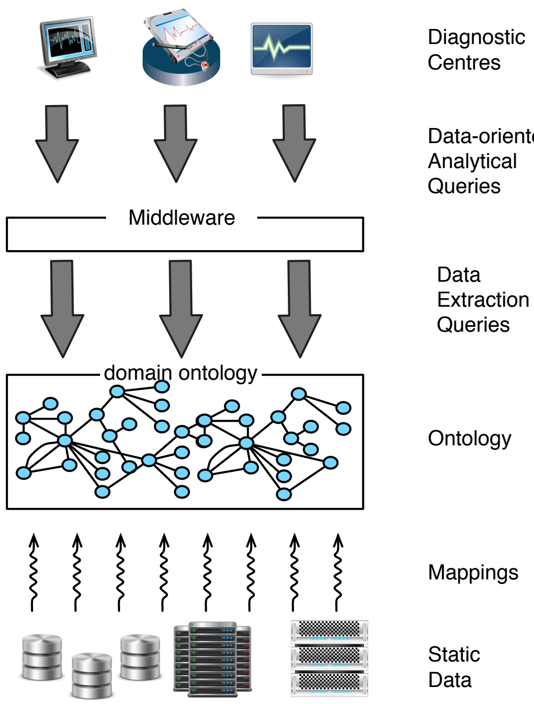


Figure 1: Conceptual architecture of OBDA

Example 1. *A typical scenario that involves analytical tasks and requires access to static and streaming data is industrial diagnostics and monitoring of equipment. Siemens has several service centres dedicated to diagnostics of thousands of power-generation appliances located across the globe [15]. A usual task of a service centre is to detect in real-time potential faults of a turbine caused by, e.g., an undesirable pattern in temperature's behaviour within various components of the turbine. Consider a (simplified) example of such a task:*

> In a given turbine, report all temperature sensors that are *reliable* (i.e., with the average score of validation tests at least 90%) and whose measurements within the last 10 min were *similar* (i.e., Pearson correlated by at least 0.75) to measurements reported last year by a reference sensor that had been functioning in a critical mode.

This task requires to extract, aggregate and correlate static data about the turbine's structure, streaming data produced by up to 2,000 sensors installed in different parts of the turbine, and historical operational data of the reference sensor stored in multiple data sources. Accomplishing such a task currently requires to pose a collection of hundreds of queries, the majority of which are semantically the same (they ask about temperature), but syntactically differ (they are over different schemata). This takes up to 80% of the overall diagnostic time that Siemens engineers as well as engineers in other large service companies typically have to spend [15].

ODBA can naturally allow to save a lot of this time since ontologies can help to 'hide' the technical details of how *the data is produced, represented, and stored in data sources, and to show only* what *this data is about. Thus, one would be able to formulate this diagnostic task using only one ontological query instead of a collection of hundreds data queries that today have to be written or configured by IT specialists. Clearly, this collection of queries does not disappear: the OBDA query transformation will automatically compute them from the high-level ontological query using the ontology and mappings.*

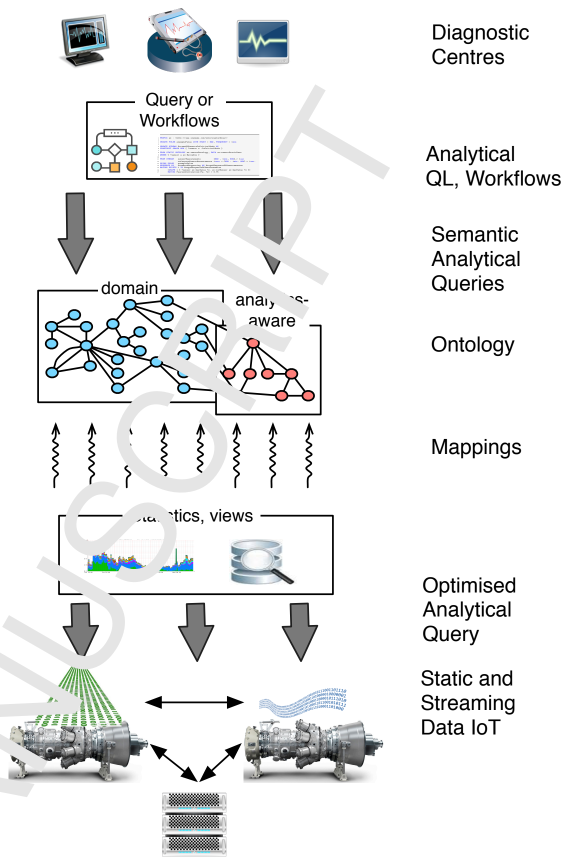


Figure 2: Conceptual architecture of analytics-enhanced OBDA

Equipment diagnostics such as the ones in the example scenario typically make heavy use of aggregation and correlation functions as well as arithmetic operations. In our running example, the aggregation function min and the comparison operator $\geq$ are used to specify what makes a sensor reliable and to define a threshold for similarity. Performing such operations in OBDA can be done either on the level of *(i)* ontological queries or *(ii)* data queries specified in the mappings. We argue that both options are unsatisfactory. Indeed, Option (i) requires that all relevant values should be retrieved prior to performing grouping and arithmetic operations. This can be highly inefficient, as it fails to exploit source capabilities (e.g., access to pre-computed averages), and value retrieval may be slow and/or costly, e.g., when relevant values are stored remotely. Moreover, it adds to the complexity of application queries, and thus limits the benefits of the abstraction layer. We illustrate this option in Figure 1 where a devoted middleware preprocesses analytical queries by 'isolating' in them data extraction queries, and postprocess answers retrieved by the latter queries using the analytical functions of the original analytical queries. Option (ii) requires that all aggregation functions and comparison operators are moved to mapping queries. This is brittle and inflexible, as values such as 90% and 0.75, which are used to define 'reliable sensor' and 'similarity', cannot be specified in the ontological query, but must be 'hard-wired' in the mappings, unless an appropriate extension to the query language or the ontology are developed. In order to address these issues, OBDA should become

> *analytics-aware* by supporting declarative representations of basic analytics operations and using these to efficiently answer higher level queries.

In practice this requires enhancing OBDA technology with ontologies, mappings, and query languages capable of capturing operations used in analytics, but also extensive modification of OBDA query preprocessing components, i.e., reasoning and query transformation, to support these enhanced languages.

Moreover, analytical tasks as in the example scenario should typically be executed continuously in data intensive and highly distributed environments of streaming and static data. Efficiency of such execution requires non-trivial query optimisation. However, optimisations in existing OBDA systems are usually limited to minimisation of the textual size of the generated queries, e.g. [17], with little support for distributed query processing, and no support for optimisation for continuous queries over sequences of numerical data and, in particular, computation of data correlation and aggregation across static and streaming data. In order to address these issues, OBDA should become

> *source and cost aware* by supporting both static and streaming data sources and offering a robust query planning component and indexing that can estimate the cost of different plans, and use such estimates to produce low-cost plans.

Note that the existence of materialised and pre-computed subqueries relevant to analytics within sources and archived historical data that should be correlated with current streaming data implies that there is a range of query plans which can differ dramatically with respect to data transfer and query execution time.

In this paper we make the first step to extend OBDA systems towards becoming analytics, source, and cost aware. In particular this will make such OBDA solution compliant to the Siemens requirements for turbine diagnostics. Consider a high level illustration of our approach in Figure 2: diagnostic engineers in diagnostic centres can create analytical queries and workflows over ontologies by relying on classical and analytical constructs offered by ontologies (that are analytically enhanced). Such semantic analytical queries are then rewritten with the help of the enhanced ontology and unfolded into analytical data queries with the help of enhanced (analytics-aware) mappings. The resulting data queries are optimised and executed over the underlying data sources.

We see particular benefits of our analytics-aware OBDA for *Internet of Things* (IoT). Indeed, in the case of industrial IoT, that is typically considered in the context of Industry 4.0, various smart machines that are equipped with sensors exchange messages and resort to various sources of information to optimise production outputs and costs. In such IoT context, it is critical to have analytical rather than data access queries that are supported by state-of-the-art OBDA systems. In Figure 2 we schematically depict an IoT with turbines and external data.

The list of our contributions is the following:

- We proposed analytics-aware OBDA components, i.e.,
  - the ontology language $\textit{DL-Lite}_{\mathcal{A}}^{\mathsf{agg}}$ that extends $\textit{DL-Lite}_{\mathcal{A}}$ with
    - attributes that have bag (multiset) extensions and closed-world semantics, and
    - concepts that are defined using results of the evaluation of aggregate functions;
  - the query language STARQL over $\textit{DL-Lite}_{\mathcal{A}}$ ontologies that combine streaming and static data;
  - the analytics-aware relational query language $\mathrm{SQL}^{\oplus}$ for static and streaming data; and
  - a mapping language relating $\textit{DL-Lite}_{\mathcal{A}}^{\mathsf{agg}}$ vocabulary and STARQL constructs with $\mathrm{SQL}^{\oplus}$ queries over static and streaming data.
- We developed efficient query transformation techniques for turning STARQL queries over $\textit{DL-Lite}_{\mathcal{A}}^{\mathsf{agg}}$ ontologies into $\mathrm{SQL}^{\oplus}$ queries using our mappings.
- We developed the following source and cost aware query optimisation techniques:
  - Query optimisations on live streams:
    - in-memory indexing structures and algorithms;
    - the adaptive stream indexing technique that decides when to build the aforementioned indexes.
  - Query optimisations on archived information:
    - efficient storage of archived streams for hybrid operations (i.e., complex analytics between live and archived streams);
    - materialised window signatures that summarise important features of archived streams;
    - the Locality Sensitive Hashing technique for fast computation of complex hybrid operations.

- We developed elastic infrastructure that automatically distributes analytical computations and data over a computational cloud for faster query execution.
- We implemented
  - the highly optimised engine ExaStream capable of handling complex streaming and static queries;
  - a dedicated STARQL2SQL$^{\oplus}$ translator that transforms STARQL queries into queries over static and streaming data; and
  - an integrated OBDA system that relies on the aforementioned and third-party components.
- We conducted a performance evaluation of our OBDA system with large scale Siemens data using analytical tasks.

Delta from Previous Publications

We reported some ideas on analytics-aware OBDA in our paper in the emerging applications track of ISWC 2016 [18]. Moreover, an earlier version of the STARQL query language has been presented in [19] and of ExaStream in [20, 21]. However, this work significantly extends our previous publications as follows:

- *DL-Lite$_{\mathcal{A}}^{\mathsf{agg}}$ analytics-aware ontology language*: In [18] we gave only a short introduction of *DL-Lite$_{\mathcal{A}}^{\mathsf{agg}}$*. In this submission we formally introduce its syntax and semantics, study the computational properties of the associated problems of satisfiability and query answering; we also include formal proofs.
- *STARQL query language*: The version of STARQL presented in this paper extends the one in [19] with the ability to use aggregate concepts. Moreover, in [18] we only briefly mentioned that this can be done, while in this submission we give an extended presentation of the STARQL language. Finally, in this paper we give an operational semantics of STARQL which we did not present previously and that is more practical from the point of view of implementation.
- *OBDA and mappings with bag semantics*: In [18] we only gave examples of mappings connecting predicates of *DL-Lite$_{\mathcal{A}}^{\mathsf{agg}}$* ontologies to relational queries. In this submission we formally introduce such mappings as a component of *extended OBDA settings*. Contrary to the set-based semantics of classical OBDA settings [1], extended OBDA settings and mappings are given a semantics that is based on bags, which is more faithful to the semantics of SQL and database systems. We also study conjunctive query answering and rewriting in this setting.
- ExaStream *backend optimisation techniques*: In [18] we introduced materialised window signatures for hybrid operations between live and archived streams. In this submission we combine materialised window signatures with the Locality Sensitive Hashing technique, for fast computation of complex analytics between live and archived streams. The combined algorithm requires much less computation. Additionally we introduce some hybrid in-memory indexing structures specifically tailored for streaming information along with the adaptive stream indexing technique that decides when its benencial to build these indexes on a specific window.
- ExaStream *implementation*: The implementation of ExaStream as presented in [18, 20, 21] is extended in this submission by implementing the aforementioned optimisation techniques.
- *Evaluation*: In [18] we evaluated the effect of distribution and the effect of materialised window signatures on complex analytics between live and archived streams. In this submission we additionally evaluate our novel in-memory indexing structures and the adaptive stream indexing technique. Furthermore we evaluate the integration of materialised window signatures with the Locality Sensitive Hashing technique.

Structure of the Paper

In Sections 2-5 we introduce our novel OBDA components, in Section 6 we discuss how we implemented a system that accounts for them, in Sections 7–8 we present backend optimisations and their evaluations, and in Section 9–10 we discuss related work and conclude.

We now give a more detailed structure.

In Section 2 we start with an analytics-aware ontology language *DL-Lite$_{\mathcal{A}}^{\mathsf{agg}}$* for capturing static aspects of the domain of interest where ontologies and aggregate functions are treated as first class citizens. In Section 3 we introduce STARQL that allows to combine static conjunctive queries over *DL-Lite$_{\mathcal{A}}^{\mathsf{agg}}$* with continuous diagnostic queries that involve simple combinations of time aware data attributes, time windows, and functions, e.g., correlations over streams of attribute values. Using STARQL queries one can retrieve entities (e.g., sensors) that pass two 'filters': static and continuous. In our running example a static 'filter' checks whether a sensor is reliable, while a continuous 'filter' checks whether the measurements of the sensor are Pearson correlated with the measurements of reference sensor. In Section 4 we present an analytics-aware relational query language for static and streaming data SQL$^{\oplus}$. In Section 5 we connect the previous sections: we explain how to bridge STARQL queries over *DL-Lite$_{\mathcal{A}}^{\mathsf{agg}}$* and SQL$^{\oplus}$ queries. To this end we review necessary background on the classical OBDA approach to bridge ontological and data oriented queries with the help of mappings and a two-stage query transformation procedure that reformulates ontological queries into data queries. Then, we explain how we extend the classical mappings to our setting by defining mappings that relate aggregate and non-aggregate concepts, properties, and attributes occurring in queries over ontologies into database schemata and relate functions and constructs of STARQL continuous 'filters' into corresponding functions and constructs over databases, and to extend the two-stage query transformation procedure. Then, we dive in detailed example-driven explanations of STARQL query transformation procedures, and discuss their correctness. In Section 6 we present our system that combines our novel components: *(i)* ontology language, *(ii)* query language over ontologies, *(iii)* query language over data, and *(iv)* mappings between the ontology and data query languages and query transformation procedures. In Section 7 we discuss how to optimise backend queries in $\mathrm{SQL}^{\oplus}$. Then, in Section 8 we present experimental evaluation of the backend where we emphasise the effect of the optimisations. Finally, in Section 9 we discuss related work, and in Section 10 we conclude and present future work.

2. $DL\text{-}Lite_{\mathcal{A}}^{\mathsf{agg}}$: An Ontology Language with Aggregates

Our ontology language, $DL\text{-}Lite_{\mathcal{A}}^{\mathsf{agg}}$, is an extension of $DL\text{-}Lite_{\mathcal{A}}$ [1] with concepts that are based on aggregation of attribute values. The semantics for such concepts adapts the closed-world semantics [22]. The main reason why we rely on this semantics is to avoid the problem of empty answers for aggregate queries under the certain answers semantics [23, 24]. In $DL\text{-}Lite_{\mathcal{A}}^{\mathsf{agg}}$ we distinguish between individuals and data values from countable sets Γ and D that intuitively correspond to the datatypes of RDF. For simplicity of presentation we assume that D is the set of rational numbers. We also distinguish between atomic roles P that denote binary relations between pairs of individuals, and attributes F that denote binary relations between individuals and data values. In $DL\text{-}Lite_{\mathcal{A}}^{\mathsf{agg}}$, attributes F are allowed to contain the same tuple multiple times as these duplicates might be produced by the evaluation of the mappings over the database. Retaining these duplicates is crucial for applications that employ aggregation and recent works caring for data aggregation have considered similar settings [25, 26].

Before proceeding to the formal definitions, we introduce the notion of a *bag* (or *multiset*) which, informally, is a collection that allows for multiple repetitions of its elements. A *bag* over a set M is a function $\Omega : M \to \mathbb{N}_0$, where $\mathbb{N}_0$ is the set of nonnegative integers. The value $\Omega(c)$ is called *the multiplicity of c in Ω*. A bag Ω is *finite* if there are finitely many $c \in M$ with $\Omega(c) > 0$. The *empty bag* $\emptyset$ over M is the bag satisfying $\emptyset(c) = 0$ for all $c \in M$. We also define the binary operation of *bag intersection* $\Cap$ for such bags as follows: for every $c \in M$, it holds that $(\Omega_1 \Cap \Omega_2)(c) = \min\{\Omega_1(c), \Omega_2(c)\}$.

2.1. Syntax of $DL\text{-}Lite_{\mathcal{A}}^{\mathsf{agg}}$

Assume a vocabulary consisting of countably infinite and pair-wise disjoint sets standing for atomic concepts $\mathbf{C}$, atomic roles $\mathbf{R}$, and atomic attributes $\mathbf{A}$. Let also agg be an aggregate function (e.g., min, max, count, countd, sum, avg), let r be a rational number, and $\circ$ be a comparison predicate on rational numbers, e.g., $\geq, \leq, <, >, =$, or $\neq$. The grammar for concepts and roles in $DL\text{-}Lite_{\mathcal{A}}^{\mathsf{agg}}$ is defined based on the above vocabulary as follows, where $A \in \mathbf{C}$, $P \in \mathbf{R}$, $F \in \mathbf{A}$:

$$B \to A \mid \exists R, \quad C \to B \mid \exists F,$$
$$E \to \circ_r (\mathsf{agg}\ F), \quad R \to P \mid P^{-}.$$

We call expressions B, C, and E *basic*, *extended*, and *aggregate* concepts, respectively, and call expression R a *basic role*.

A $DL\text{-}Lite_{\mathcal{A}}^{\mathsf{agg}}$ ontology $\mathcal{O}$ is a finite set of axioms. We consider the following types of axioms: *(i) concept inclusions* of the form $E \sqsubseteq B$ and $C \sqsubseteq B$, and *role inclusions* of the form $R_1 \sqsubseteq R_2$, *(ii) functionality* axioms on roles of the form $(\mathsf{funct}\ R)$, and *(iii) concept, role, and attribute denials* of the form $B_1 \sqcap B_2 \sqsubseteq \bot$, $R_1 \sqcap R_2 \sqsubseteq \bot$, and $F_1 \sqcap F_2 \sqsubseteq \bot$, respectively.

Let $a, b \in \Gamma$ and $v \in D$. A $DL\text{-}Lite_{\mathcal{A}}^{\mathsf{agg}}$ *dataset* $\mathcal{D}$ is a finite bag over the set of assertions of the form $A(a)$, $P(a, b)$, and $F(a, v)$ where in addition it is required that assertions of the form $A(a)$ and $P(a, b)$ occur in $\mathcal{D}$ at most once. Intuitively, $\mathcal{D}$ allows only multiple occurrences for attribute assertions.

We require that if $(\mathsf{funct}\ R)$ is in $\mathcal{O}$, then $R' \sqsubseteq R$ is *not* in $\mathcal{O}$ for any R' different from R. This syntactic condition, as well as the fact that we do not allow concepts of the form $\exists F$ and aggregate concepts to appear on the right-hand side of inclusions ensure good computational properties of $DL\text{-}Lite_{\mathcal{A}}^{\mathsf{agg}}$. The former restriction is inherited from $DL\text{-}Lite_{\mathcal{A}}$ while the latter can be shown using techniques of [22] (see following sections).

Example 2. *The following concept inclusion comprises a $DL\text{-}Lite_{\mathcal{A}}^{\mathsf{agg}}$ ontology capturing the notion of reliable sensors as this was introduced in our running example:*

$$\geq_{0.9} (\mathsf{min}\ \mathit{testScore}) \sqsubseteq \mathit{Reliable}. \tag{1}$$

Here Reliable is an atomic concept, testScore is an atomic attribute, and $\geq_{0.9}$ $(\mathsf{min}\ \mathit{testScore})$ is an aggregate concept that captures individuals with one or more testScore values whose minimum is at least 0.9.

2.2. Semantics of $DL\text{-}Lite_{\mathcal{A}}^{\mathsf{agg}}$

We define the semantics of $DL\text{-}Lite_{\mathcal{A}}^{\mathsf{agg}}$ in terms of interpretations $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ that assign to individuals in Γ an element of their domain $\Delta^{\mathcal{I}}$, assign to data values in D the corresponding rational number in $\mathbb{Q}$, and assign to atomic concepts $A \in \mathbf{C}$, to atomic roles $P \in \mathbf{R}$, and to atomic attributes $F \in \mathbf{A}$, a subset of $\Delta^{\mathcal{I}}$, a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and a bag over $\Delta^{\mathcal{I}} \times \mathbb{Q}$, respectively. Moreover, for an atomic role $P \in \mathbf{R}$, a basic role R, and a data value

$r \in D$, interpretation $\mathcal{I}$ satisfies:

$$\begin{aligned}
(P^-)^\mathcal{I} &= \{(a,b) \in \Delta^\mathcal{I} \times \Delta^\mathcal{I} \mid (b,a) \in P^\mathcal{I}\},\\
(\exists R)^\mathcal{I} &= \{a \in \Delta^\mathcal{I} \mid \text{exists } b \in \Delta^\mathcal{I} \text{ with } (a,b) \in R^\mathcal{I}\},\\
(\exists F)^\mathcal{I} &= \{a \in \Delta^\mathcal{I} \mid \text{exists } v \in \mathbb{Q} \text{ with } F^\mathcal{I}(a,v) > 0\},\\
(\circ_r(\mathsf{agg}\ F))^\mathcal{I} &= \{a \in \Delta^\mathcal{I} \mid \\
&\qquad \mathsf{agg}\{\!\!\{v : m \mid v \in \mathbb{Q}, m = F^\mathcal{I}(a,v)\}\!\!\} \circ r^\mathcal{I}\}.
\end{aligned}$$

Here, $\{\!\!\{\cdot\}\!\!\}$ denotes a bag and its meaning is well-defined since bags over a set M can been seen as sets of elements $c : m$ where $c \in M$ and $m \in \mathbb{N}_0$. Also, expression $\mathsf{agg}\{\!\!\{\cdot\}\!\!\}$ denotes the evaluation of aggregate agg over the provided bag $\{\!\!\{\cdot\}\!\!\}$. In our setting, expression $\mathsf{agg}\{\!\!\{\cdot\}\!\!\}$ always evaluates to a rational number.

Please note that although the semantics interprets attributes F as bags, extended concepts based on attributes, such as $\exists F$, are given a classical set-based semantics. This is in contrast to the recent work in [25] that defined bag interpretations as functions assigning to concepts and roles bags over $\Delta^\mathcal{I}$ and $\Delta^\mathcal{I} \times \Delta^\mathcal{I}$, respectively. In the following, we assume the standard name assumption for interpretations $\mathcal{I}$, which requires that individuals and data values are interpreted as themselves, i.e., $c^\mathcal{I} = c$ for each $c \in \Gamma \cup D$. This effectively makes $\Delta^\mathcal{I}$ and $\mathbb{Q}$ equal to Γ and D, respectively.

The notion of a *model* for interpretations $\mathcal{I}$, *DL-Lite*$^{\mathsf{agg}}_\mathcal{A}$ ontologies $\mathcal{O}$, and datasets $\mathcal{D}$ is defined similarly to [22, 25]. We say that an interpretation $\mathcal{I}$ is a *model* of $\mathcal{O} \cup \mathcal{D}$, written as $\mathcal{I} \models \mathcal{O} \cup \mathcal{D}$, if all of the following hold:

(i) $a^\mathcal{I} \in A^\mathcal{I}$ if $\mathcal{D}(A(a)) = 1$, $(a^\mathcal{I}, b^\mathcal{I}) \in P^\mathcal{I}$ if $\mathcal{D}(P(a,b)) = 1$, and $F^\mathcal{I}(a^\mathcal{I}, v^\mathcal{I}) = \mathcal{D}(F(a,v))$ for all assertions of the form $A(a)$, $P(a,b)$, and $F(a,v)$;

(ii) $S_1^\mathcal{I} \subseteq S_2^\mathcal{I}$, for each concept and role inclusion axiom $S_1 \sqsubseteq S_2$ in $\mathcal{O}$;

(iii) $(a,b) \in R^\mathcal{I}$ and $(a,c) \in R^\mathcal{I}$ implies $b = c$, for each functionality axiom $(\mathsf{funct}\ R)$ in $\mathcal{O}$;

(iv) $S_1^\mathcal{I} \cap S_2^\mathcal{I} = \emptyset$, for each denial axiom $S_1 \sqcap S_2 \sqsubseteq \bot$ in $\mathcal{O}$ where S_1 and S_2 are both concepts or roles;

(v) $F_1^\mathcal{I} \Cap F_2^\mathcal{I} = \emptyset$, for each denial axiom $F_1 \sqcap F_2 \sqsubseteq \bot$ in $\mathcal{O}$.

Requirements *(ii)*–*(iv)* are as in the set case, whereas requirement *(v)* is the natural extension of requirement *(iv)* to bags [25]. Requirement *(i)* is a mixture of set and closed-world semantics to reflect the closed-world nature of attributes: models of $\mathcal{O} \cup \mathcal{D}$ shall interpret attributes F according to the assertions on F found in the dataset.

Example 3. *Consider the dataset*

$$\begin{aligned}
\mathcal{D} = \{\!\!\{ & \mathit{Reliable}(s_0) : 1, \mathit{testScore}(s_1, 0.9) : 2,\\
& \mathit{testScore}(s_2, 0.95) : 1, \mathit{testScore}(s_2, 0.98) : 1,\\
& \mathit{testScore}(s_3, 0.5) : 1, \mathit{testScore}(s_3, 0.9) : 1\}\!\!\}.
\end{aligned}$$

For every model $\mathcal{I}$ of $\mathcal{D}$ and the ontology in Equation (1)*, it holds that* $(\geq_{0.9}\ (\mathsf{min}\ \mathit{testScore}))^\mathcal{I} = \{s_1, s_2\}$ *and* $s_0 \in \mathit{Reliable}^\mathcal{I}$*; thus* $\{s_0, s_1, s_2\} \subseteq \mathit{Reliable}^\mathcal{I}$*.*

An important reasoning task in ontologies is *satisfiability checking* that asks whether an ontology has a model. Given a *DL-Lite*$^{\mathsf{agg}}_\mathcal{A}$ ontology $\mathcal{O}$ and dataset $\mathcal{D}$, one can easily show that satisfiability checking for $\mathcal{O} \cup \mathcal{D}$ can be decided in polynomial time in the size of $\mathcal{O} \cup \mathcal{D}$ provided that computation of aggregate functions can be done in polynomial time in the size of $\mathcal{D}$. Indeed, this can be shown by a reduction to satisfiability checking in *DL-Lite*$_\mathcal{A}$.

Proposition 1. *Let $\mathcal{O}$ be a DL-Lite$^{\mathsf{agg}}_\mathcal{A}$ ontology with aggregate functions computable in polynomial time. Let also $\mathcal{D}$ be a dataset. Then, satisfiability checking for $\mathcal{O} \cup \mathcal{D}$ can be decided in polynomial time in the size of $\mathcal{O} \cup \mathcal{D}$.*

Proof. Given $\mathcal{O}$ and $\mathcal{D}$ we construct in polynomial time in the size of $\mathcal{O} \cup \mathcal{D}$ a *DL-Lite*$_\mathcal{A}$ ontology $\mathcal{O}'$ and a dataset $\mathcal{D}'$ such that $\mathcal{O} \cup \mathcal{D}$ is satisfiable if and only if $\mathcal{O}' \cup \mathcal{D}'$ is satisfiable. Then, the claim follows from Theorem 4.22 in [1], which shows that satisfiability checking in *DL-Lite*$_\mathcal{A}$ can be done in polynomial time in the size of both the ontology and the dataset.

In proof of the above claim, let $\mathcal{O}'$ be the *DL-Lite*$_\mathcal{A}$ ontology obtained from $\mathcal{O}$ by replacing each aggregate concept of the form $\circ_r(\mathsf{agg}\ F)$ appearing in the axioms of $\mathcal{O}$ with a fresh atomic concept U. Let $\mathcal{D}'$ be defined as the *set* of assertions corresponding to $\mathcal{D}$ extended with the set of assertions $\{U(a) \mid \mathsf{agg}\{\!\!\{v : m \mid v \in \mathbb{Q}, m = \mathcal{D}(F(a,v))\}\!\!\} \circ r\}$, for each aggregate concept $\circ_r(\mathsf{agg}\ F)$ in $\mathcal{O}$ and concept U introduced in $\mathcal{O}'$ for $\circ_r(\mathsf{agg}\ F)$.

Suppose now that $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ is a model of $\mathcal{O} \cup \mathcal{D}$ and let $\mathcal{I}' = (\Delta^\mathcal{I}, \cdot^{\mathcal{I}'})$ be the interpretation such that: *(i)* $S^{\mathcal{I}'} = S^\mathcal{I}$, for every $S \in \mathbf{C} \cup \mathbf{R}$, *(ii)* $F^{\mathcal{I}'} = \{(a,v) \in \Delta^\mathcal{I} \times \Delta^\mathcal{I} \mid F^\mathcal{I}(a,v) > 0\}$, for every $F \in \mathbf{A}$, and *(iii)* $U^{\mathcal{I}'} = (\circ_r(\mathsf{agg}\ F))^\mathcal{I}$, for every concept U introduced in $\mathcal{O}'$ for an aggregate concept $\circ_r(\mathsf{agg}\ F)$ in $\mathcal{O}$. It is now straightforward to check that $\mathcal{I}'$ is a model of $\mathcal{O}' \cup \mathcal{D}'$.

For the other direction, assume that $\mathcal{I}' = (\Delta^{\mathcal{I}'}, \cdot^{\mathcal{I}'})$ is a model of $\mathcal{O}' \cup \mathcal{D}'$. Observe that concepts U and $\exists F$ appear only in the left-hand side of concept inclusion axioms in $\mathcal{O}'$, thus, the subinterpretation $\mathcal{I}'' = (\Delta^{\mathcal{I}'}, \cdot^{\mathcal{I}''})$ of $\mathcal{I}'$ defined such that $S^{\mathcal{I}''} = S^{\mathcal{I}'}$, $U^{\mathcal{I}''} = \{a^{\mathcal{I}'} \in \Delta^{\mathcal{I}'} \mid U(a) \in \mathcal{D}'\}$, and $F^{\mathcal{I}''} = \{(a^{\mathcal{I}'}, v^{\mathcal{I}'}) \in \Delta^{\mathcal{I}'} \times \mathbb{Q} \mid F(a,v) \in \mathcal{D}'\}$, where $S \in \mathbf{C} \cup \mathbf{R}$, $F \in \mathbf{A}$, and U is the concept corresponding to an aggregate concept $\circ_r(\mathsf{agg}\ F)$, is also a model of $\mathcal{O}' \cup \mathcal{D}'$. Now, let $\mathcal{I} = (\Delta^{\mathcal{I}'}, \cdot^\mathcal{I})$ be the interpretation such that $S^\mathcal{I} = S^{\mathcal{I}''}$, for every $S \in \mathbf{C} \cup \mathbf{R}$, and $F^{\mathcal{I}'}(a,v) = \mathcal{D}(F(a,v))$, for every $F \in \mathbf{A}$. By construction, $\mathcal{I}$ is a model of $\mathcal{O} \cup \mathcal{D}$. □

2.3. Query Answering in *DL-Lite*$^{\mathsf{agg}}_\mathcal{A}$

Our query language for querying *DL-Lite*$^{\mathsf{agg}}_\mathcal{A}$ ontologies will be the class of *conjunctive queries* that consists of all expressions of the form $q(\vec{x})$:- $\mathsf{conj}(\vec{x})$, where $\vec{x}$ is a tuple of variables of arity k, conj is a conjunction of atoms of the form $A(t)$, $E(t)$, $P(t_1, t_2)$, or $F(t, s)$ with $A \in \mathbf{C}$, $P \in \mathbf{R}$, $F \in \mathbf{A}$, $E = \circ_r(\mathsf{agg}\ F)$, and t, t_1, t_2 being either variables or constants from Γ, and s being either a variable or constant from D. We also assume that every variable in $\vec{x}$ appears in some atom in conj. Following the standard approach for ontologies, we adopt the semantics of *certain answers* for answering conjunctive queries. Informally, the certain answers $\mathsf{cert}(q, \mathcal{O}, \mathcal{D})$ to a query q over the union of an ontology $\mathcal{O}$ and dataset $\mathcal{D}$ comprises all tuples of arity k over $\Gamma \cup D$ for which the query is entailed by the ontology. Formally, this set is defined as

$$\mathsf{cert}(q, \mathcal{O}, \mathcal{D}) = \{\vec{t} \in (\Gamma \cup D)^k \mid \mathcal{I} \models \mathsf{conj}(\vec{t}) \text{ for each model } \mathcal{I} \text{ of } \mathcal{O} \cup \mathcal{D}\}.$$

Example 4. *Let $\mathcal{O}$ be the ontology in Equation* (1) *and $\mathcal{D}$ be the dataset specified in Example 3. Consider also the conjunctive query $q(x)$:- Reliable(x) that asks for all reliable sensors. Following the observation made in Example 3, every model $\mathcal{I}$ of $\mathcal{O} \cup \mathcal{D}$ satisfies $\{s_0, s_1, s_2\} \subseteq Reliable^{\mathcal{I}}$, hence, the certain answers to q over $\mathcal{O} \cup \mathcal{D}$ is* $\mathsf{cert}(q, \mathcal{O}, \mathcal{D}) = \{s_0, s_1, s_2\}$.

We now show that conjunctive query answering in *DL-Lite*$_{\mathcal{A}}^{\mathsf{agg}}$ is tractable, assuming that computation of aggregate functions can be done in time polynomial in the size of the data. This is proved in the proposition below by reducing conjunctive query answering over ontologies with aggregates to the corresponding problem over aggregate-free ontologies with closed predicates [22]. This is possible due to the fact that each aggregate concept and each attribute behaves like a closed predicate in the setting of [22], in the sense that its interpretation—given an ontology $\mathcal{O}$ and dataset $\mathcal{D}$—is determined and fixed by $\mathcal{D}$. Before stating the proposition, we introduce the notion of *safety* for *DL-Lite$_{\mathcal{A}}$ ontologies with closed predicates*, where the syntax of such ontologies follows that of *DL-Lite*$_{\mathcal{A}}^{\mathsf{agg}}$ with the exception that concept inclusions are formed only between extended concepts, whereas the semantics is the standard one [1].

Definition 1 ([22])**.** *Let $\mathcal{O}$ be a DL-Lite$_{\mathcal{A}}$ ontology and Σ be a finite set of predicates from $\mathbf{C} \cup \mathbf{R} \cup \mathbf{A}$. We call the pair $(\mathcal{O}, \Sigma)$ an* ontology with closed predicates *and say that $(\mathcal{O}, \Sigma)$ is* safe *if there are no concepts C_1, C_2 and no role R such that* (i) *C_1 is satisfiable in $\mathcal{O}$ and different from $\exists R'$ with $\mathcal{O} \models R' \sqsubseteq R$,* (ii) *$\mathcal{O} \models C_1 \sqsubseteq \exists R$ and $\mathcal{O} \models \exists R^- \sqsubseteq C_2$,* (iii) *$C_2$ mentions a predicate in Σ, and* (iv) *every role R' with $\mathcal{O} \models C_1 \sqsubseteq \exists R'$ and $\mathcal{O} \models R' \sqsubseteq R$ mentions a predicate outside Σ.*

The theorem below states that safety of *DL-Lite$_{\mathcal{A}}$* ontologies with closed predicates makes conjunctive query answering equivalent to the corresponding problem in *DL-Lite$_{\mathcal{A}}$* ontologies.

Theorem 1 ([22])**.** *Let $(\mathcal{O}, \Sigma)$ be a DL-Lite$_{\mathcal{A}}$ ontology with closed predicates and let $q(\vec{x})$ be a conjunctive query of arity k. If $(\mathcal{O}, \Sigma)$ is safe, then, for every dataset $\mathcal{D}$ satisfiable with $(\mathcal{O}, \Sigma)$, the certain answers to $q(\vec{x})$ over $(\mathcal{O}, \Sigma)$ and $\mathcal{D}$ coincide with the certain answers to $q(\vec{x})$ over $\mathcal{O} \cup \mathcal{D}$.*

We are now able to prove that query answering in *DL-Lite*$_{\mathcal{A}}^{\mathsf{agg}}$ is tractable.

Proposition 2. *Let $\mathcal{O}$ be a DL-Lite$_{\mathcal{A}}^{\mathsf{agg}}$ ontology with aggregate functions computable in polynomial time, let $\mathcal{D}$ be a dataset, and let $q(\vec{x})$ be a conjunctive query of arity k and of fixed size. Checking whether $\vec{a} \in \mathsf{cert}(q, \mathcal{O}, \mathcal{D})$ for a tuple $\vec{a} \in (\Gamma \cup D)^k$ can be decided in polynomial time in the size of $\mathcal{O} \cup \mathcal{D}$.*

Proof. Given $\mathcal{O}$, $\mathcal{D}$, and q we construct in polynomial time in the size of $\mathcal{O} \cup \mathcal{D}$ a safe *DL-Lite$_{\mathcal{A}}$* ontology with closed predicates $(\mathcal{O}', \Sigma)$, a dataset $\mathcal{D}'$, and a query q' such that $\mathsf{cert}(q, \mathcal{O}, \mathcal{D}) = \mathsf{cert}_{\Sigma}(q', \mathcal{O}', \mathcal{D}')$, where $\mathsf{cert}_{\Sigma}(q', \mathcal{O}', \mathcal{D}')$ denotes the set of certain answers to q' over $(\mathcal{O}', \Sigma)$ and $\mathcal{D}'$. By safety of $(\mathcal{O}', \Sigma)$ and Theorem 1, we have that $\mathsf{cert}_{\Sigma}(q', \mathcal{O}', \mathcal{D}')$ coincides with the certain answers to q' over the *DL-Lite$_{\mathcal{A}}$* ontology $\mathcal{O}' \cup \mathcal{D}'$ whenever $\mathcal{D}'$ is satisfiable with $(\mathcal{O}', \Sigma)$. Since satisfiability of $(\mathcal{O}', \Sigma)$ with $\mathcal{D}'$ can be checked in polynomial time in the size of $\mathcal{O}'$ and $\mathcal{D}'$ [22, 1] and the same is true for checking whether a tuple $\vec{a}$ from $(\Gamma \cup D)^k$ is a certain answer to $q(\vec{x})$ over $\mathcal{O}' \cup \mathcal{D}'$ [1, Theorem 5.17], the claim then follows.

In proof of the above claim, let $\mathcal{O}'$ and $\mathcal{D}'$ be defined as in the proof of Proposition 1. Let also q' be the query obtained from q by replacing each aggregate atom $E(v)$ in q with the atom $U(v)$, where E is $\circ_r(\mathsf{agg}\ F)$ and U is the concept used to replace E in the derivation of $\mathcal{O}'$ from $\mathcal{O}$. Let also Σ comprise all attributes F appearing in $\mathcal{O}'$ and all concepts U in $\mathcal{O}'$ for an aggregate concept E in $\mathcal{O}$. Given that concepts U and $\exists F$ appear only in the left-hand side of concept inclusion axioms in $\mathcal{O}'$ and that the only predicates in Σ are exactly the U's and all attributes F in $\mathcal{O}'$, this means that there is no concept C_2 that could be employed to satisfy requirements (ii) and (iii) of Definition 1, thus, $\mathcal{O}'$ is safe.

To show that $\mathsf{cert}(q, \mathcal{O}, \mathcal{D}) = \mathsf{cert}_{\Sigma}(q', \mathcal{O}', \mathcal{D}')$, it suffices to prove that there is a one-to-one correspondence between the models of $\mathcal{O} \cup \mathcal{D}$ and those of $(\mathcal{O}', \Sigma)$ and $\mathcal{D}'$ such that if $\mathcal{I}$ is a model of the former ontology and $\mathcal{I}'$ is the corresponding model of the latter one, then $\mathcal{I} \models \mathsf{conj}(\vec{a})$ if and only if $\mathcal{I}' \models \mathsf{conj}(\vec{a})$, for all tuples $\vec{a} \in (\Gamma \cup D)^k$. Observe that a one-to-many correspondence between these two sets of models has been already established in the proof of Proposition 1, which considered the mapping of the *DL-Lite*$_{\mathcal{A}}^{\mathsf{agg}}$ ontology $\mathcal{O} \cup \mathcal{D}$ to the *DL-Lite$_{\mathcal{A}}$* ontology $\mathcal{O}' \cup \mathcal{D}'$ without the use of closed predicates. Notice, however, that in the presence of the closed predicates in Σ and for the models $\mathcal{I}''$ and $\mathcal{I}'$ of $\mathcal{O}' \cup \mathcal{D}'$ considered in the last paragraph of that proof, we have that $\mathcal{I}'' = \mathcal{I}'$, thus, this correspondence becomes one-to-one. Note also that the equivalence $\mathcal{I} \models \mathsf{conj}(\vec{a})$ if and only if $\mathcal{I}' \models \mathsf{conj}(\vec{a})$ holds trivially by construction of $\mathcal{I}'$ on the basis of $\mathcal{I}$. □

In addition to the tractability of query answering in $DL\text{-}Lite_{\mathcal{A}}^{\mathsf{agg}}$, one can show that the standard query rewriting algorithm of [1] proposed for $DL\text{-}Lite_{\mathcal{A}}$ as a part of query transformation procedure (with an extension discussed in Section 5) also works for $DL\text{-}Lite_{\mathcal{A}}^{\mathsf{agg}}$ and SQL.

2.4. Discussion

Note that our aggregate concepts can be encoded as aggregate queries over attributes as soon as the latter are interpreted under the closed-world semantics. Indeed, the certain answers for the atomic query $q(x)$:- $(\circ_r(\mathsf{agg}\ F))(x)$ would be the same as for the following aggregate query:

$$\mathsf{sql}_{\circ_r(\mathsf{agg}\ F)}(x) = \mathsf{SELECT}\ \ x\ \ \mathsf{FROM}\ \ F(x,y)$$
$$\mathsf{GROUP\ BY}\ \ x\ \ \mathsf{HAVING}\ \ \mathsf{agg}(y) \circ r. \quad (2)$$

Thus, one can reduce conjunctive query answering over our analytics aware $DL\text{-}Lite_{\mathcal{A}}^{\mathsf{agg}}$ ontologies to aggregate query answering over classical $DL\text{-}Lite_{\mathcal{A}}$ ontologies as soon as the closed-world semantics is exploited for the interpretation of data attributes. At the same time, we argue that in a number of applications, such as monitoring and diagnostics at Siemens [15], explicit aggregate concepts of $DL\text{-}Lite_{\mathcal{A}}^{\mathsf{agg}}$ give us significant modelling and query formulation advantages over $DL\text{-}Lite_{\mathcal{A}}$ since in such applications concepts are naturally based on aggregate values of potentially many different attributes. For instance, in Siemens the notion of reliability is naturally based on aggregation over various attributes, i.e., it should be modelled as $E_i \sqsubseteq \mathit{Reliable}$ for many different aggregate concepts E_i, and reliability is also commonly exploited in diagnostic queries. In the case of $DL\text{-}Lite_{\mathcal{A}}^{\mathsf{agg}}$, in all such diagnostic queries it suffices to use only one atom $\mathit{Reliable}(x)$. In contrast, in the case of $DL\text{-}Lite_{\mathcal{A}}$, each such diagnostic query would have to contain the whole union $\mathit{Reliable}(x) \cup_i \mathsf{sql}_{E_i}(x)$. Thus, Siemens diagnostics queries over $DL\text{-}Lite_{\mathcal{A}}$ would be much more complex than the ones over $DL\text{-}Lite_{\mathcal{A}}^{\mathsf{agg}}$. Moreover, in the case of $DL\text{-}Lite_{\mathcal{A}}$, the diagnostics queries of the form $\mathsf{sql}_{E_i}(x)$ will have to be adjusted each time the notion of reliability is modified, while, in the case of $DL\text{-}Lite_{\mathcal{A}}^{\mathsf{agg}}$, only the ontology and not the queries should be adjusted.

3. STARQL: A Query Language over $DL\text{-}Lite_{\mathcal{A}}^{\mathsf{agg}}$ Ontologies for Static and Streaming Data

In this section we will give an overview of STARQL, illustrate it on our running example, and then explain its syntax and semantics. Moreover, we will compare STARQL to state-of-the art query languages over RDF streams in terms of their syntactic features. We refer the reader to [27] where we compare STARQL's implementation with respect to other systems in terms of architectural and implementation aspects. We also refer the reader to [28] were we compare STARQL with the LTL-based description logic of TCQs [29], and show that a safe fragment of TCQs is captured by STARQL.

3.1. Overview and Example

STARQL is a query language over ontologies that allows to query both streaming and static data and supports not only standard aggregates such as count and avg, but also more advanced aggregation functions from our backend system such as Pearson correlation.

Each STARQL query takes as input a static $DL\text{-}Lite_{\mathcal{A}}^{\mathsf{agg}}$ ontology and a static dataset (logical view of data stored in a relational DB) as well as a set of live and historic streams. The output of the query is a stream of timestamped data assertions about objects that occur in the static input data and satisfy two kinds of filters: *(i) static*, that is, a conjunctive query over the input static ontology and data and *(ii) streaming*, that is, a diagnostic query over the input streaming data—which can be live and archived (i.e., static)— that may involve typical mathematical, statistical, and event pattern features needed in diagnostic scenarios for streaming data. Therefore, any STARQL query Q_{starql} is essentially a conjunction of two queries: a static conjunctive query Q_{StatCQ} over $DL\text{-}Lite_{\mathcal{A}}^{\mathsf{agg}}$, and a streaming query Q_{Stream} over $DL\text{-}Lite_{\mathcal{A}}$:

$$Q_{\mathsf{starql}} \approx Q_{\mathsf{StatCQ}} \wedge Q_{\mathsf{Stream}}. \quad (3)$$

The syntax of STARQL is inspired by the W3C standardised SPARQL query language, allowing for nesting of queries. Moreover, STARQL has a formal semantics that combines open and closed-world reasoning and extends snapshot semantics for window operators [30] with sequencing semantics that can handle integrity constraints such as functionality assertions.

In Figure 3 we present a STARQL query that captures the diagnostic task from our running example and uses concepts, roles, and attributes from the Siemens ontology [15, 31, 32, 33, 34, 35, 36] and Eq. (1). The query has three parts: declaration of the output stream (Lines 5 and 6); sub-query over the static data (Lines 8 and 9) that, in the running example, corresponds to '*return all temperature sensors that are reliable, i.e., with the average score of validation tests at least 90%*'; and sub-query over the streaming data (Lines 11–17) that, in the running example, corresponds to '*whose measurements within the last 10 min Pearson correlate by at least 0.75 to measurements reported by a reference sensor last year*'. Moreover, in Line 1 the namespace that is used in the sub-queries is declared, i.e., the URI of the Siemens ontology, and in Line 3 the pulse of the streaming sub-query is defined.

3.2. Syntax and Comparison to other Languages

We now enumerate the main clauses of STARQL and illustrate them using the query in Figure 3:

`CREATE PULSE` clause declares a global time tick specified by an update frequency and a starting point (here set to `NOW` to specify that the streaming starts with the registration of the query). The pulse determines the time points `NOW` (as referenced in line 6 of 3) at which

```
PREFIX ex : <http://www.siemens.com/onto/gasturbine/>

CREATE PULSE examplePulse WITH START = NOW, FREQUENCY = 1min

CREATE STREAM StreamOfSensorsInCriticalMode AS
CONSTRUCT GRAPH NOW { ?sensor a :InCriticalMode }

FROM STATIC ONTOLOGY ex:sensorOntology, DATA ex:sensorStaticData
WHERE { ?sensor a ex:Reliable }

FROM STREAM   sensorMeasurements          [NOW - 1min, NOW]-> 1sec,
              referenceSensorMeasurements 1year <-[NOW - 1min, NOW]-> 1sec,
USING PULSE   examplePulse
SEQUENCE BY   StandardSequencing AS MergedSequenceOfMeasurementes
HAVING EXISTS i IN MergedSequenceOfMeasurementes
     (GRAPH i { ?sensor ex:hasValue ?y. ex:refSensor ex:hasValue ?z })
     HAVING PearsonCorrelation(?y, ?z) > 0.75
```

Figure 3: Running example query expressed in STARQL

the stream data are outputted. This global output time points are necessary as a STARQL query may refer to multiple streams with different slides.

`CREATE STREAM` clause declares the name of the output stream. In our example the output stream is called *StreamOfSensorsInCriticalMode*.

`SELECT/CONSTRUCT` clause defines how the output stream declared in the previous clause should be formed. STARQL allows for two types of output: the `SELECT` clause forms the output as simply the lists of variable bindings, while the `CONSTRUCT` clause defines the output as an RDF graph that further can be stored in an RDF store or sent as input to another STARQL query. In our example, we form the output as a set of data assertions of the form $A(b)$, thus making an RDF graph consisting of all sensors (i.e., `?sensor`) that function in a critical mode (i.e, `ex:InCriticalMode`) and are determined by the two sub-queries.

`FROM STATIC/STREAM` clause declares input static ontology and data and defines streaming data with window parameters using the start and end value, e.g., '`[NOW - 1min, NOW]`', as well as a slide parameter, e.g., '`->1sec`'. In our example, we have the static ontology `ex:sensorOntology` and data `DATA ex:sensorStaticData` and two streams: `sensorMeasurements` of live sensor measurements and also `referenceSensorMeasurements` of recorded measurements of the reference sensor. Note that the recorded sensor uses a set back time of one year, that is, values from one year ago are correlated to a live stream.

`USING` clause defines the periodic pulse for the input streams, given by an execution frequency, e.g., `1min` and its absolute start and/or end time, e.g., `NOW`. The pulse is a global clock that determines the output times points of the stream query. The main purpose of the pulse parameter is to align the different referenced streams which may have different (local) slide and range parameters.

`WHERE` clause declares a static conjunctive query expressed as a SPARQL graph pattern. The output variables of this query identify possible answers over the static data. In our example, the query is *Reliable*(x) where x corresponds to `?sensor` in the graph pattern '`?sensor a ex:Reliable`'.

`SEQUENCE BY` clause defines how the input streams should be merged into one and gives a name to the resulting merged stream. Using the built-in standard sequencing strategy results in a merged stream were all and only those stream data with the same timestamp are put into the same state (named RDF graph).

`HAVING` clause declares a streaming query. It can contain various constructs, including a conjunctive query expressed as a graph pattern, applied over all elements of the merged stream that have a specific timestamp identified by an index. In our example the query '`?sensor ex:hasValue ?y. ex:refSensor ex:hasValue ?z`' which is applied at the index point '`i`' of the merged stream and retrieves all measurements values of the candidate sensor (i.e., `?sensor`) and the reference sensor (i.e., `ex:refSensor`). In the `HAVING` clause one can do more than referring to specific time points: one can also compare them by evaluating graph patterns on each of the states or just return variables mentioned in the graph pattern, while restricting them by logical conditions or correlations. In our example, we verify that the live values `?y` of the candidate sensor are Pearson correlated with the archived values `?z` of the reference sensor with a degree greater than 0.75.

We also note that STARQL distinguishes between two kinds of variables that correspond to either points of time and their arrangement in the temporal sequence, or to the actual values defined by graph patterns of the `HAVING` or `WHERE` clause. Variables of different kinds cannot be mixed and points in time cannot be part of the output. Note that the state based relations of the `HAVING` clause are safe in the first-order logic sense and can be arranged by filter conditions on the state variables. This safety condition guarantees `HAVING` clauses are domain independent and thus can be smoothly transformed into domain independent queries in the languages of CQL [30] and SQL$^{\oplus}$, which is our extension of SQL for stream handling (see Sec. 6 for more details).

For other features of STARQL we refer the reader to [28, 19]. A comparison of STARQL with state-of-the-art RDF stream languages and engines is given the Sect. 9 on related work.

3.3. Semantics

Intuitively, the semantics of STARQL combines open and closed-world reasoning and extends snapshot semantics for window operators [30] with sequencing semantics that can handle integrity constraints such as functionality assertions. In particular, the window operator in combination with the sequencing operator provides a sequence of datasets on which temporal (state-based) reasoning can be applied. Every temporal dataset frequently produced by the window operator is converted to a sequence of (pure) datasets. The sequence strategy determines how the timestamped assertions are sequenced into datasets. In the case of the presented example in Figure 3, the chosen sequencing method is *standard sequencing*: assertions with the same timestamp are grouped into the same dataset. So, at every time point, one has a sequence of datasets on which temporal (state-based) reasoning can be applied. This is realised in STARQL by a sorted first-order logic template in which state stamped graph patterns are embedded. For evaluation of the time sequence, the graph patterns of the static `WHERE` clause are mixed into each state to join static and streaming data. Note that STARQL uses semantics with a real temporal dimension, where time is treated in a non-reified manner as an additional ontological dimension and not as ordinary attribute as, e.g., in SPARQL-Stream [9].

A formal denotational semantics of STARQL can be found in [37]. From the implementation point of view, an operational semantics is more helpful—at least it gives a different perspective on the intended semantics of the window. A full operational semantics along the lines of [38] is planned for future work. We illustrate the operational

```
CREATE STREAM S_{out}
...
FROM Sin [NOW-wr, NOW]->sl
USING PULSE WITH START = st, FREQUENCY = fr
...
```

Figure 4: Template query for illustration of operational semantics

semantics of the window in our terminology in order to make clear two points: Why is the snapshot-semantics of the window chosen in the way described in [37] and illustrated in the example before? Why dow we need a pulse declaration?

Consider the query template given in the listing of Figure 4. Let $timeExp_1$ = `NOW-wr` stand for the left end of the window, where `wr` is a constant denoting the window range, and $timeExp_2$ = `NOW` stand for the right end. We distinguish between a pulse time t_{pulse} and a stream time t_{str}. (For more than one stream one would have more local stream times.) The pulse time t_{pulse} evolves regularly according to the frequency specification,

$$t_{pulse} = st \longrightarrow st + fr \longrightarrow st + 2fr \longrightarrow \dots.$$

In contrast, the stream time t_{str} is jumping/sliding and is determined by the trace of endpoints of the sliding window. More concretely, the evolvement of t_{str}, which can be easily implemented, is specified as follows:

$$t_{str} \xrightarrow[\text{(for } m \in \mathbb{N} \text{ maximal)}]{\text{IF } t_{str} + m \times sl \leq t_{pulse}} t_{str} + m \times sl.$$

The window contents at t_{pulse} is given by:

$$\{triple\langle t\rangle \in S_{in} \mid t_{str} - wr \leq t \leq t_{str}\}.$$

Note that the following always holds: $t_{str} \leq t_{pulse}$. This is a crucial point since it enables STARQL to be used for both historical reasoning and stream reasoning. Indeed, having always $t_{str} \leq t_{pulse}$ guarantees that applying the window on real-time streams does not give different stream elements than when applying the window on a simulated stream from a DB with historical data. In other words, if $t_{str} > t_{pulse}$, then the window in a historical query would contain future elements from $[t_{pulse}, t_{str}]$ whereas in the real-time case the window cannot contain future elements from $[t_{pulse}, t_{str}]$.

We now illustrate t_{pulse} and t_{str} on our running example.

Example 5. *For the* STARQL *query in the listing of Figure 5 one gets the following evolvement of the pulse time and the streaming time:*

$$\begin{array}{rl} t_{pulse}: & 0s \to 2s \to 4s \to 6s \to 8s \to 10s \to 12s \to \\ t_{str}: & 0s \to 0s \to 3s \to 6s \to 6s \to 9s \to 12s \to \end{array}$$

```
CREATE STREAM Sout AS
...
FROM STREAM Sin:  [NOW-3s, NOW] -> 3s
USING PULSE WITH START =  0s, FREQUENCY = 2
    s
...
```

Figure 5: Query illustrating operational semantics on one stream

```
CREATE STREAM Sout AS
...
FROM  STREAM Sin1 [NOW-3s, NOW]->3s,
      STREAM Sin2 [NOW-3s, NOW]->2s
USING PULSE WITH START  = 0s, FREQUENCY = 2
    s
SEQUENCE BY StdSeq AS seq
...
```

Figure 6: Query illustrating operational semantics on two streams

The example query in the listing of Figure 6 refers to multiple streams and is intended to illustrate the synchronization effect of the pulse:

$$\begin{aligned} t_{pulse}: &\quad 0s \rightarrow 2s \rightarrow 4s \rightarrow 6s \rightarrow 8s \rightarrow 10s \rightarrow 12s \rightarrow \\ t_{str_1}: &\quad 0s \rightarrow 0s \rightarrow 3s \rightarrow 6s \rightarrow 6s \rightarrow 9s \rightarrow 12s \rightarrow \\ t_{str_2}: &\quad 0s \rightarrow 2s \rightarrow 4s \rightarrow 6s \rightarrow 8s \rightarrow 10s \rightarrow 12s \rightarrow \end{aligned}$$

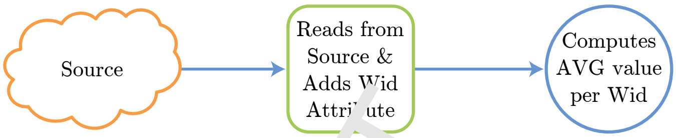
4. SQL$^{\oplus}$: An Analytics-aware Relational Query Language for Static and Streaming Data

We introduced SQL$^{\oplus}$ language as an extension of SQL with operators for handling streaming data and for combining streaming and static data. SQL$^{\oplus}$ contains a number of important pre-defined functions for data analysis and allows to introduce new such functions defined by users. SQL$^{\oplus}$ relies on the semantics of *Continuous Query Language* (CQL) [30], an expressive SQL-based declarative language for registering continuous queries against streams and updatable relations. Both SQL$^{\oplus}$ and CQL adopt specific operators for mapping streams of information to finite relations via a windowing mechanisms.

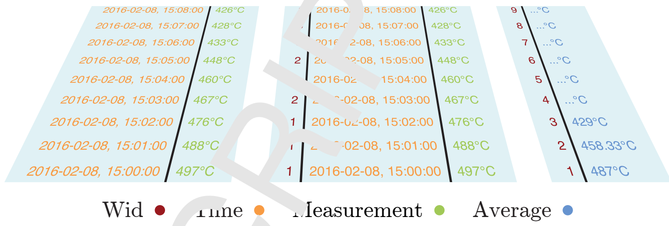
4.1. Data Model and Execution Architecture

We define our data model and execution architecture following the terminology that has been presented in the bibliography, e.g. Storm's data model and execution architecture [39] as well as the computational model presented in [40].

Within the SQL$^{\oplus}$ data model, a *topology* describes the flow of streaming and static records between *computational*



(a) Topology



(b) Dataflow

```
CREATE STREAM read_from_source AS
SELECT *
FROM (TIMESLIDINGWINDOW timewindow:3 frequency:3
 SELECT *
 FROM (http 'http://ip:port/stream1'));

CREATE STREAM avg_value AS
SELECT wid, AVG(value)
FROM read_from_source
GROUP BY wid;

SELECT * FROM avg_value;
```

(c) Syntactical representation

Figure 7: A simple SQL$^{\oplus}$ topology, its corresponding dataflow, and its syntactical representation

nodes. Computational nodes are logical processing units that have one or more live-stream or static-data inputs and one output. They execute a set of *operations* on their input to produce the corresponding output. Computational nodes can be classified as either having exclusively *live-stream inputs*, exclusively *static-data inputs*, and *hybrid inputs*. Similarly they can be classified to being *streaming* or *static*, based on the form of their output.

A special type of computational nodes are those responsible for communicating external sources to our topology, similar to Storm's spouts. These input nodes:

(i) access external sources, e.g. access live streams from *OPC* and *HTTP servers*

(ii) associate each external source to a *time-sliding window mechanism*, i.e. a mechanism of forming (possibly overlapping) *sub-sequences of tuples* (*windows*) at pre-determined time instances;

(iii) associate each record accessed from some external source to a temporal identifier and window identifiers.

Example 6. *Figure 7a shows a simple topology. The input node receives information from a stream of temperature measurements acquired from a single sensor on some power* *generating turbine. The initial data contain the temperature measurement in Celsius degrees and the time that this measurement was acquired. The input node processes the records arriving from the source, acknowledges the temporal identifier indicated by the source, and relates each measurement to a time-sliding window mechanism that assumes a window of size* 10 *sec is produced every* 10 *sec. Then a second computational node calculates the average temperature value grouped by windows. The result is stored in the table as in Figure 7b.*

4.2. A Declarative Language for Computations

ExaStream takes advantage of existing Database Management technologies and optimisations by providing a declarative language, namely SQL$^{\oplus}$, extending the SQL syntax and semantics for querying live streams and relations. In contrast to popular distributed DSMSs, such as Storm[1], Flink[2], Kafka[3], Heron[4], and Spark Streaming[5] that offer an API that allows the user to submit dataflows of user defined operators, the user can define complex dataflows using a declarative language. The system's *query planner* is responsible for choosing an optimal plan depending on the query, the available stream/static data sources, and the execution environment. It should be noted that several state-of-the-art systems for Big Data processing are adopting a similar approach, providing for declarative SQL-like languages for data processing. Apache Spark allows to query structured data inside Spark programs using SQL queries, while KSQL is a streaming SQL engine that enables real-time data processing against Apache Kafka. The query optimiser makes it possible to process SQL$^{\oplus}$ queries that blend streams with static and historical data (e.g., archived streams).

In order to incorporate the algorithmic logic for extending SQL into SQL$^{\oplus}$ several operators and statements have been implemented:

`CreateStream` : The *create stream* statement allows to add a new computational node to our topology that outputs a live stream. The *create stream* statement always contains a *Select* subquery that determines the operations that are performed on the input records. Input records are identified in the *From* clause of the subquery.

`TimeSlidingWindow` : This specific operator is implemented as a user defined function, groups tuples from the same time window and associates them with a unique window identifier corresponding to the `Wid` attribute. The *timeSlidingWindow* operator produces results in the order of `Wid` and `Time` attributes. The operator is used by input computational nodes to create the corresponding window identifier.

`WCache` : *WCache* is an SQL$^{\oplus}$ operator that when applied between two streams it is translated to an equality join between the two streams on their corresponding `Wid` attribute. WCache also creates the indexing structures for answering efficiently equality constraints on the `Wid` and `Time` attributes when processing infinite streams. The WCache operator, its related indexes and corresponding optimisations are presented in Section 7.1.1.

It should be noted that the aforementioned SQL$^{\oplus}$ operators are based on the semantics of the CQL language [30].

Example 7. *In Figure 7c we see an example of the* SQL$^{\oplus}$ *language. The presented query correspond to the topology shown in Figure 7a. The* `create stream` *statement creates the two different computational nodes responsible for reading from the data source (*`read_from_source`*) and computing the average value per window (*`avg_value`*). As we see the* `read_from_source` *computational node uses two user defined functions:* `http` *reads the stream data that are pushed from an HTTP server; and* `timeslidingwindow` *is responsible for creating the windows based on the windowing mechanism expressed by the* `timewindow` *and* `frequency` *parameters. The frequency attribute defines that a window will be created every* 3 *secs and the timewindow defines that the length of the window is* 3 *secs. The* `avg_value` *computational node has* `read_from_source` *as its input and outputs a new stream that contains the average value per window. Finally the select query is the one that shows the results of the* `avg_value` *stream.*

5. Bridging STARQL over *DL-Lite*$_{\mathcal{A}}^{\mathsf{agg}}$ and SQL$^{\oplus}$: Mapping Language and Query Transformation

In this section we explain how to bridge STARQL and SQL$^{\oplus}$. To this end we start in Section 5.1 by reviewing the classical OBDA approach to bridge ontological and data oriented queries with the help of mappings (we give their syntax and semantics) and a two-stage query transformation procedure (we also review correctness of this procedure). Then, in Section 5.2 we explain how we extend the classical mappings and the query transformation procedure to account for the features of STARQL queries $Q_{\mathsf{starql}} \approx Q_{\mathsf{StatCQ}} \wedge Q_{\mathsf{Stream}}$ (recall Equation (3)) and aggregate concepts of *DL-Lite*$_{\mathcal{A}}^{\mathsf{agg}}$. Subsequently, we give an example-driven but formal explanation of the query transformation procedure for static queries Q_{StatCQ} in Section 5.3 and of streaming queries Q_{Stream} in Section 5.4. Afterwards, in Section 5.5 we discuss correctness of the query transformation procedures. Finally, in Section 5.6 we discuss practical advantages of aggregate concepts.

[1] Apache Storm. `http://storm.apache.org`
[2] Apache Flink. `http://flink.apache.org`
[3] Apache Kafka. `https://kafka.apache.org`
[4] Twitter Heron. `https://apache.github.io/incubator-heron`
[5] Spark Streaming. `https://spark.apache.org/streaming`

5.1. Background on OBDA

We now present notions from traditional OBDA and refer the reader to [1, 41] for further details. A *database schema* $\mathcal{S}$ is a finite set of relational symbols P with associated arities and associated attribute domains given by $ar(P)$ and $dom_P(i), i \in [1, ar(P)]$, respectively. For simplicity, we assume that $\mathcal{S}$ is fixed and that the only attribute domains are the set of individuals Γ and the set of data values D introduced in Section 2. A *database instance* $\mathcal{B}$ is a finite set of assertions of the form $P(d_1, \ldots, d_{ar(P)})$, where P is a relation symbol in $\mathcal{S}$ and each d_i is from $dom_P(i)$, $i \in [1, ar(P)]$. We view a SQL query sql of arity k as a function that assigns to every database instance $\mathcal{B}$ a finite subset $\mathsf{ans}(\mathsf{sql}, \mathcal{B})$ of $(\Gamma \cup D)^k$.

Let $\mathcal{L}$ be an *ontology language* and $\mathcal{O}$ and ontology from $\mathcal{L}$. Following the practice of OBDA we rely on the so-called *global-as-view* (GAV) mappings [1] that relate each (atomic) ontological term from $\mathcal{O}$ (i.e., concept, relation, or attribute) to a query over $\mathcal{S}$. Formally, a GAV mapping is of the form

$$S(\vec{x}) \leftarrow \mathsf{sql}(\vec{x}), \tag{4}$$

where S is an atomic concept, an atomic role, or an atomic attribute, sql is a SQL query over relation symbols in $\mathcal{S}$ with appropriate arity and attribute domains, and $\vec{x}$ is tuple of variables with no repetitions. We denote with $\mathcal{M}$ a *set of* GAV *mappings*.

An *OBDA setting* is a triple of the form $(\mathcal{B}, \mathcal{M}, \mathcal{O})$ where $\mathcal{B}$ is a database instance, $\mathcal{M}$ is a set of GAV mappings, and $\mathcal{O}$ is an ontology from $\mathcal{L}$. The semantics of an OBDA setting is defined on the basis of first-order interpretations. An interpretation $\mathcal{I}$ is a *model* of $(\mathcal{B}, \mathcal{M}, \mathcal{O})$ if $\mathcal{I} \models \mathcal{O}$ and for every mapping $S(\vec{x}) \leftarrow \mathsf{sql}(\vec{x})$ in $\mathcal{M}$ and every tuple $\vec{t}$ of elements from $\Gamma \cup D$, if $\vec{t} \in \mathsf{ans}(\mathsf{sql}, \mathcal{B})$, then $\vec{t}^{\mathcal{I}} \in S^{\mathcal{I}}$.

The semantics of query answering in OBDA is based on the notion of certain answers. Let $q(\vec{x}) \leftarrow \mathsf{conj}(\vec{x})$ be a conjunctive query of arity k over the vocabulary of $\mathcal{O}$. The set of certain answers to q over an OBDA setting $(\mathcal{B}, \mathcal{M}, \mathcal{O})$ is defined as

$$\begin{aligned}\mathsf{cert}(q, (\mathcal{B}, \mathcal{M}, \mathcal{O})) = \{&\vec{t} \in (\Gamma \cup D)^k \mid \mathcal{I} \models \mathsf{conj}(\vec{t}) \\ &\text{for each model } \mathcal{I} \text{ of } (\mathcal{B}, \mathcal{M}, \mathcal{O})\}.\end{aligned}$$

Query answering in OBDA is realised by a two-stage transformation procedure that reformulates the input query q to a query $\hat{q}$ so that the answers to the latter over $\mathcal{B}$ coincides with the certain answers to q over $(\mathcal{B}, \mathcal{M}, \mathcal{O})$. This transformation is graphically depicted below.

$$q \xrightarrow[\mathcal{O}]{\mathsf{rewrite}} \bar{q} \xrightarrow[\mathcal{M}]{\mathsf{unfold}} \hat{q} \tag{5}$$

In the first stage of the transformation, query q is reformulated using the '$\mathsf{rewrite}$' procedure to a query $\bar{q}$ over $\mathcal{O}$ that incorporates the knowledge expressed in $\mathcal{O}$; in the second stage, $\bar{q}$ is further reformulated using the 'unfold' procedure to a query $\hat{q}$ over $\mathcal{B}$ that additionally incorporates the mappings $\mathcal{M}$. The correctness of such a reformulation is usually shown on the basis of the *virtual dataset* $\mathcal{D}_{\mathcal{M},\mathcal{B}}$, which is the dataset obtained from $\mathcal{B}$ and $\mathcal{M}$ by materialising the answers $\vec{t}$ in $\mathsf{ans}(\mathsf{sql}, \mathcal{B})$ as assertions $S(\vec{t})$, for each mapping $S(\vec{x}) \leftarrow \mathsf{sql}(\vec{x})$ in $\mathcal{M}$. The virtual dataset allows to cast the problem of computing the certain answers to q over $(\mathcal{B}, \mathcal{M}, \mathcal{O})$ as the problem of computing the certain answers to q over the ontology defined by the union of $\mathcal{O}$ and $\mathcal{D}_{\mathcal{M},\mathcal{B}}$, that is $\mathsf{cert}(q, (\mathcal{B}, \mathcal{M}, \mathcal{O})) = \mathsf{cert}(q, \mathcal{O}, \mathcal{D}_{\mathcal{M},\mathcal{B}})$. Then, to show correctness of the reformulation procedure depicted in Equation (5), one shows that the answers to the rewriting $\bar{q}$ over the dataset $\mathcal{D}_{\mathcal{M},\mathcal{B}}$ coincide with $\mathsf{cert}(q, \mathcal{O}, \mathcal{D}_{\mathcal{M},\mathcal{B}})$ and, subsequently, that these answers coincide with the answers to the unfolding $\hat{q}$ over the database instance $\mathcal{B}$. This is summarised symbolically in the following equations:

$$\begin{aligned}\mathsf{cert}(q, (\mathcal{B}, \mathcal{M}, \mathcal{O})) &= \mathsf{cert}(q, \mathcal{O}, \mathcal{D}_{\mathcal{M},\mathcal{B}}) \\ &= \mathsf{ans}(\bar{q}, \mathcal{D}_{\mathcal{M},\mathcal{B}}) \\ &= \mathsf{ans}(\hat{q}, \mathcal{B}).\end{aligned} \tag{6}$$

In [1] it was shown that the query transformation procedure described above for conjunctive queries is correct when $\mathcal{L}$ is *DL-Lite*$_{\mathcal{A}}$. In the following we show how we extend this result to *DL-Lite*$_{\mathcal{A}}^{\mathsf{agg}}$ and STARQL queries.

5.2. Extending OBDA for *DL-Lite*$_{\mathcal{A}}^{\mathsf{agg}}$ and STARQL

We now discuss how we extend mappings and give a high level overview of an extended two stage transformation procedure.

Mappings. STARQL queries are defined over *DL-Lite*$_{\mathcal{A}}^{\mathsf{agg}}$ ontologies and have complex constructs related to stream processing. Thus, the classical mappings should be extended to account for these features and we consider two types of mappings:

- *schema-mappings*: from atomic concepts, roles, attributes, as well as from aggregate concepts to SQL queries over relational schemas of static, streaming, or historical data, and
- *construct-mappings*: from the constructs of the streaming queries of STARQL into SQL$^{\oplus}$ queries over streaming and historical data. These are built on the basis of schema-mappings by compiling in the pulse, slide, and the sequencing constructs into them.

For the syntax of construct-mappings we refer the reader to [42, 28], while here we will exemplify them as follows and sketch how they are compiled n the basis of schema mappings in Section 5.4.

Example 8.

$\mathsf{GRAPH}\ i\ \{?sensor\ ex{:}hasVal\ ?y\} \leftarrow$
$\quad \mathsf{SELECT}\ sid\ as\ ?sensor,\ sval\ as\ ?y, wid\ as\ i$
$\quad \mathsf{FROM}$
$\quad\quad [\ \mathsf{SELECT} * \mathsf{FROM}$
$\quad\quad\quad (\ \mathsf{TIMESLIDINGWINDOW}$
$\quad\quad\quad\quad timewindow : r$
$\quad\quad\quad\quad frequency : sl$
$\quad\quad\quad\quad \mathsf{SELECT}\ *\ \mathsf{FROM}(\mathsf{http}\ ip{-}of{-}Msmt)$
$\quad\quad\quad)$
$\quad\quad];$

In this example, a named graph template is mapped to an SQL$^\oplus$ *query. The mapping relies on parameters r and s from* STARQL *queries to accomplish the correct mapping of sates* `i` *to time points in* SQL$^\oplus$.

The syntax of schema-mappings is the same as the syntax of GAV mappings given in Equation (4) with the additional restriction that query sql in Equation (4) mentions a top-level $\mathsf{DISTINCT}$ specifier whenever S is a concept or a role. The reason for imposing this restriction stems from the fact that *DL-Lite*$^{\mathsf{agg}}_{\mathcal{A}}$ interprets concepts and roles as sets, while it interprets attributes as bags. In the following we describe how the syntax and semantics of OBDA need to be extended to account for bags.

Semantics of Extended OBDA. A *bag database instance* $\mathcal{B}$ is a finite bag over the set of assertions of the form $P(d_1, \ldots, d_{ar(P)})$, where P is a relation symbol in $\mathcal{S}$ and each d_i is from $dom_P(i)$, $i \in [1, ar(P)]$. We view a SQL query sql of arity k as a function assigning to every bag database instance $\mathcal{B}$ a finite bag $\mathsf{ans}(\mathsf{sql}, \mathcal{B})$ over the set of tuples in $(\Gamma \cup D)^k$. An *extended OBDA setting* is now a triple $(\mathcal{B}, \mathcal{M}, \mathcal{O})$, where $\mathcal{B}$ is a bag database instance, $\mathcal{M}$ is a set of schema-mappings and of construct-mappings, and $\mathcal{O}$ is a *DL-Lite*$^{\mathsf{agg}}_{\mathcal{A}}$ ontology.

We now define the semantics of extended OBDA settings for schema-mappings and refer the reader to [43, 28] for the semantics of construct-mappings. Let $(\mathcal{B}, \mathcal{M}, \mathcal{O})$ be an extended OBDA setting where $\mathcal{M}$ is a set of schema-mappings. We say that a *DL-Lite*$^{\mathsf{agg}}_{\mathcal{A}}$ interpretation $\mathcal{I}$ is a model of $(\mathcal{B}, \mathcal{M}, \mathcal{O})$ if $\mathcal{I} \models \mathcal{O}$ and $\mathcal{I}$ satisfies the following two conditions, where S ranges over atomic concepts and atomic roles, and F ranges over atomic attributes:

1. For every $S(\vec{x}) \leftarrow \mathsf{sql}(\vec{x})$ in $\mathcal{M}$ and every $\vec{t}$ over Γ, if $\mathsf{ans}(\mathsf{sql}, \mathcal{B})(\vec{t}) \geq 1$, then $\vec{t}^{\mathcal{I}} \in S^{\mathcal{I}}$;
2. For every $\vec{t}$ in $\Gamma \times D$ it holds that $F^{\mathcal{I}}(\vec{t}^{\mathcal{I}}) \geq \sum_{F(\vec{x}) \leftarrow \mathsf{sql}(\vec{x}) \in \mathcal{M}} \mathsf{ans}(\mathsf{sql}, \mathcal{B})(\vec{t})$.

Let us clarify now the above definition. Recall that when $\mathcal{B}$ is a bag database instance, $\mathsf{ans}(\mathsf{sql}, \mathcal{B})$ is defined as a bag of tuples; thus expression $\mathsf{ans}(\mathsf{sql}, \mathcal{B})(\vec{t})$ denotes the multiplicity of $\vec{t}$ in bag $\mathsf{ans}(\mathsf{sql}, \mathcal{B})$. Condition 1 above then stipulates that if $\mathcal{M}$ contains a mapping $S(\vec{x}) \leftarrow \mathsf{sql}(\vec{x})$ and tuple $\vec{t}$ appears in the answers to query sql over $\mathcal{B}$, then (the interpretation of) $\vec{t}$ must also appear in the extension of S under $\mathcal{I}$. Therefore, this condition together with the requirement that $\mathcal{I}$ must be a model of $\mathcal{O}$ constitute only a reformulation of the definition of models in standard OBDA settings. The difference in the two definitions stems from Condition 2, which stipulates that the multiplicity of (the interpretation of) a tuple $\vec{t}$ in the extension of an attribute F under $\mathcal{I}$ must be at least as large as the sum of the multiplicities of $\vec{t}$ in the bags $\mathsf{ans}(\mathsf{sql}_1, \mathcal{B}), \ldots, \mathsf{ans}(\mathsf{sql}_n, \mathcal{B})$, where $F(\vec{x}) \leftarrow \mathsf{sql}_1(\vec{x}), \ldots, F(\vec{x}) \leftarrow \mathsf{sql}_n(\vec{x})$ are all mappings in $\mathcal{M}$ populating attribute F. The intuition behind this definition is to simulate the semantics of SQL according to which the multiplicity of a tuple in the result of a query corresponds to the number of different proofs for that tuple.

Given the definition of models above, the definition of certain answers for conjunctive queries over extended OBDA settings coincide with the one over standard OBDA settings modulo the notion of (virtual) datasets. We now extend the notion of virtual datasets to extended OBDA settings. The virtual dataset $\mathcal{D}_{\mathcal{M},\mathcal{B}}$ corresponding to an extended OBDA setting $(\mathcal{B}, \mathcal{M}, \mathcal{O})$ is defined as the bag satisfying the following two conditions, where $\vec{t}$ ranges over tuples of elements in $\Gamma \cup D$, S ranges over atomic concepts and roles, and F ranges over attributes:

$$\mathcal{D}_{\mathcal{M},\mathcal{B}}(S(\vec{t})) = \max_{S(\vec{x}) \leftarrow \mathsf{sql}(\vec{x}) \in \mathcal{M}} \{\mathsf{ans}(\mathsf{sql}, \mathcal{B})(\vec{t})\},$$
$$\mathcal{D}_{\mathcal{M},\mathcal{B}}(F(\vec{t})) = \sum_{F(\vec{x}) \leftarrow \mathsf{sql}(\vec{x}) \in \mathcal{M}} \mathsf{ans}(\mathsf{sql}, \mathcal{B})(\vec{t}).$$

Given the similarity in the definitions of models and virtual datasets for extended OBDA settings, it is straightforward to show that Equation (6) holds for extended OBDA settings or, in other words, that the certain answers to conjunctive queries q over $(\mathcal{B}, \mathcal{M}, \mathcal{O})$ coincides with the certain answers to q over the union of the *DL-Lite*$^{\mathsf{agg}}_{\mathcal{A}}$ ontology $\mathcal{O}$ and the virtual dataset $\mathcal{D}_{\mathcal{M},\mathcal{B}}$.

Proposition 3. *For any extended OBDA setting* $(\mathcal{B}, \mathcal{M}, \mathcal{O})$ *and any conjunctive query* q, *we have* $\mathsf{cert}(q, (\mathcal{B}, \mathcal{M}, \mathcal{O})) = \mathsf{cert}(q, \mathcal{O}, \mathcal{D}_{\mathcal{M},\mathcal{B}})$.

We now give an example illustrating query answering over extended OBDA settings.

Example 9. *Let* $\mathcal{S}$ *be a database schema comprising the relations* $\mathsf{S}(\mathsf{TRB}, \mathsf{SNS}, \mathsf{OP}, \mathsf{TMP})$ *and* $\mathsf{T}(\mathsf{SNS}, \mathsf{RT})$, *where* S *records the operational temperature of sensors and* T *records the fraction of measurements the system has received from a sensor. Thus, an assertion* $\mathsf{S}(t, s, 1, 50)$ *means that sensor* s*, which is attached to the turbine* t*, is operational and has temperature* $50\,^\circ\mathrm{C}$ *at some time point, whereas an assertion* $\mathsf{T}(s, 0.3)$ *means that only* 30% *of the total number of measurements sensor* s *transmitted over a* *predefined period of time were eventually recorded in the system. Let $\mathcal{B}$ be the bag database instance over $\mathcal{S}$*

$$\begin{aligned}\mathcal{B} = \{\!|\,&\mathsf{S}(t_0, s_0, 0, 0) : 1, \mathsf{S}(t_1, s_1, 1, 50) : 1,\\ &\mathsf{S}(t_2, s_2, 1, 25) : 1, \mathsf{S}(t_3, s_3, 1, 50) : 1,\\ &\mathsf{T}(s_0, 0) : 1, \mathsf{T}(s_1, 0.9) : 1, \mathsf{T}(s_2, 0.98) : 1, \mathsf{T}(s_3, 0.9) : 1\,|\!\}.\end{aligned}$$

Let also $\mathcal{M}$ comprise the mappings

$$\begin{aligned}\mathit{Reliable}(x) &\leftarrow \mathsf{sql}_1(x),\\ \mathit{testScore}(x, y) &\leftarrow \mathsf{sql}_2(x, y),\\ \mathit{testScore}(x, y) &\leftarrow \mathsf{sql}_3(x, y),\end{aligned}$$

where the SQL queries $\mathsf{sql}_1, \mathsf{sql}_2, \mathsf{sql}_3$ are defined as

$\mathsf{sql}_1(x)$: SELECT DISTINCT SNS AS x FROM S WHERE OP $= 0$,
$\mathsf{sql}_2(x, y)$: SELECT SNS AS x, $(1 -$ TEMP/500$)$ AS y FROM S WHERE OP $= 1$,
$\mathsf{sql}_3(x, y)$: SELECT SNS AS x, RT AS y FROM T WHERE RT > 0.

Last, let $\mathcal{O}$ be the DL-Lite$^{\mathsf{agg}}_{\mathcal{A}}$ ontology given in Equation (1) of Example 2. Then, the triple $(\mathcal{B}, \mathcal{M}, \mathcal{O})$ defines an extended OBDA setting that populates the role Reliable with non-operational sensors and populates attribute testScore with operational sensors assigned a score that either denotes how far the temperature of the turbine, as measured by the sensor, is from its maximum operational temperature (currently assigned to 500 °C) or the fraction of the measurements of the sensor successfully recorded in the system.

We next employ the correspondence between the OBDA setting $(\mathcal{B}, \mathcal{M}, \mathcal{O})$ and the virtual dataset $\mathcal{D}_{\mathcal{M},\mathcal{B}}$ in computing the certain answers to query $q(x)$:- Reliable(x) (see Proposition 3). Observe that $\mathsf{ans}(\mathsf{sql}_1, \mathcal{B}) = \{\!|s_0 : 1|\!\}$, $\mathsf{ans}(\mathsf{sql}_2, \mathcal{B}) = \{\!|(s_1, 0.9) : 1, (s_2, 0.95) : 1, (s_3, 0.9) : 1|\!\}$, and $\mathsf{ans}(\mathsf{sql}_3, \mathcal{B}) = \{\!|(s_1, 0.9) : 1, (s_2, 0.98) : 1, (s_3, 0.9) : 1|\!\}$, thus, by definition of virtual datasets, $\mathcal{D}_{\mathcal{M},\mathcal{B}}$ corresponds to the dataset defined in Example 2. By Example 4, we have that $\mathsf{cert}(q, \mathcal{O}, \mathcal{D}_{\mathcal{M},\mathcal{B}}) = \{s_0, s_1, s_2\}$, thus, we derive that $\mathsf{cert}(q, (\mathcal{B}, \mathcal{M}, \mathcal{O})) = \{s_0, s_1, s_2\}$.

Query Transformation Procedure: Overview. Due to the separation property (Equation (3)) of STARQL queries we can define a transformation procedure for STARQL queries as follows:

$$\begin{aligned}Q_{\mathsf{starql}} \approx Q_{\mathsf{StatCQ}} \wedge Q_{\mathsf{Stream}} &\xrightarrow[\mathcal{O}]{\mathsf{rewrite}} Q'_{\mathsf{StatUCQ}} \wedge Q'_{\mathsf{Stream}}\\ &\xrightarrow[\mathcal{M}]{\mathsf{unfold}} Q''_{\mathsf{AggSQL}} \wedge Q''_{\mathsf{Stream}} \approx Q_{\mathsf{sql}^\oplus}.\end{aligned} \tag{7}$$

During the transformation process the static conjunctive Q_{StatCQ} and streaming Q_{Stream} parts of Q_{starql}, are first independently *rewritten* using the 'rewrite' procedure that relies on the input ontology $\mathcal{O}$ into the union of static conjunctive queries Q'_{StatUCQ} and a new streaming query Q'_{Stream}, and then *unfolded* using the 'unfold' procedure that relies on the input mappings $\mathcal{M}$ into an aggregate SQL query Q''_{AggSQL} and a streaming SQL$^\oplus$ query Q''_{Stream} that together give an SQL$^\oplus$ query $Q_{\mathsf{sql}^\oplus}$, i.e., $Q_{\mathsf{sql}^\oplus} = \mathsf{unfold}(\mathsf{rewrite}(Q_{\mathsf{starql}}))$. In this transformation procedure we rely on the rewriting procedure of [1] while unfolding is different in that it relies on the two new types of mappings.

In what follows we exemplify the transformation procedures for static and streaming queries, discuss their correctness and also discuss practical benefits of aggregate concepts.

5.3. Transformation of Static Queries

In realising the first stage of the query transformation, we rely on the rewriting procedure of [1], called PerfectRef, for which we assume familiarity. As a reminder, recall that PerfectRef takes as input a conjunctive query q and a DL-Lite$_{\mathcal{A}}$ ontology and outputs a union of conjunctive queries $\bar{q}$ satisfying $\mathsf{cert}(q, \mathcal{O}, \mathcal{D}) = \mathsf{ans}(\bar{q}, \mathcal{D})$, for every dataset $\mathcal{D}$. Each conjunctive query in $\bar{q}$ is derived from q by applying to q a series of *(i)* rewriting or *(ii)* unification steps according to which *(i)* either an atom α_1 is replaced by an atom α_2 whenever there is an inclusion axiom $C_2 \sqsubseteq C_1$ in $\mathcal{O}$ such that C_i unifies with α_i or *(ii)* two atoms are unified into one with the goal of enabling a rewriting step that would otherwise not be applicable. For conjunctive queries over *DL-Lite*$^{\mathsf{agg}}_{\mathcal{A}}$ both of these steps are required and are indeed performed in the same fashion. The only exception is the treatment of atoms based on aggregate concepts and of attributes for which *DL-Lite*$^{\mathsf{agg}}_{\mathcal{A}}$ adopts a closed-world semantics, and thus, PerfectRef must leave them intact. Indeed, due to the imposed syntactic restrictions on *DL-Lite*$^{\mathsf{agg}}_{\mathcal{A}}$, such constructs can occur only on the left-hand side of inclusion axioms, hence, the rewriting step is never applicable, whereas the unification step, which can be only applied to two atoms mentioning an attribute, does not enable further applications of a rewriting step either.

To illustrate the above discussion, we apply PerfectRef to the example ontology in (1) and the query $q(x)$:- *Reliable*(x) to obtain query

$$\bar{q}(x) = \mathit{Reliable}(x) \vee (\geq_{0.9} (\mathsf{min}\ \mathit{testScore}))(x). \tag{8}$$

Before stating the correctness of the rewriting, we introduce the class of unions of conjunctive queries of arity k as the set of all queries of the form $q(\vec{x}) = q_1(\vec{x}) \vee \cdots \vee q_n(\vec{x})$ where each q_i is a conjunctive query of arity k $q_i(\vec{x})$:- $\mathsf{conj}_i(\vec{x})$. We define the answers to q over a dataset $\mathcal{D}$ as the set

$$\mathsf{ans}(q, \mathcal{D}) = \{\vec{t} \in (\Gamma \cup D)^k \mid \mathcal{I} \models \bigvee_{i=1}^{n} \mathsf{conj}_i(\vec{t}) \text{ for all } \mathit{DL\text{-}Lite}^{\mathsf{agg}}_{\mathcal{A}} \text{ models } \mathcal{I} \text{ of } \mathcal{D}\}.$$

Proposition 4. *For any DL-Lite$^{\mathsf{agg}}_{\mathcal{A}}$ ontology $\mathcal{O}$, any dataset $\mathcal{D}$, and any conjunctive query q, where $\bar{q}$ is the output of* PerfectRef *on inputs q and $\mathcal{O}$, we have* $\mathsf{cert}(q, \mathcal{O}, \mathcal{D}) = \mathsf{ans}(\bar{q}, \mathcal{D})$.

In realising the second stage of the query transformation, namely, the unfolding of $\bar{q}$, we define the output of procedure unfold on query atoms $S(\vec{t})$ and $F(t, s)$, where S is an atomic concept or role and F is an atomic attribute, and then extend it to atoms of the form $(\circ_r(\mathsf{agg}\ F))(t)$ and to (unions of) conjunctive queries.

For a fixed set of schema-mappings $\mathcal{M}$ and any atom $T(\vec{t})$ with $T \in \mathbf{C} \cup \mathbf{R} \cup \mathbf{A}$, we define

$$\mathsf{unfold}(T(\vec{t})) = \underset{T(\vec{y}) \leftarrow \mathsf{sql}(\vec{y}) \in \mathcal{M}}{\mathsf{op}} \mathsf{sql}(\theta(\vec{y})), \tag{9}$$

where $\mathsf{op} = \mathsf{UNION}$ if $T \in \mathbf{C} \cup \mathbf{R}$ and $\mathsf{op} = \mathsf{UNION\ ALL}$ if $T \in \mathbf{A}$, and θ is a substitution unifying atom $T(\vec{t})$ with the atoms $T(\vec{y})$ appearing in the left-hand side of mappings in $\mathcal{M}$. Given an atom $(\circ_r(\mathsf{agg}\ F))(t)$, we define

$$\mathsf{unfold}\big((\circ_r(\mathsf{agg}\ F))(t)\big) = \mathsf{sql}_{\circ_r(\mathsf{agg}\ \mathsf{unfold}(F(t,y)))}(t), \tag{10}$$

where y is a fresh variable and expression $\mathsf{sql}_{\circ_r(\mathsf{agg}\ \star)}(t)$ is the query defined in (2). Last, given a conjunctive query $q(\vec{x})$:- $\mathsf{conj}(\vec{x})$, we define $\mathsf{unfold}(q(\vec{x}))$ to be the query obtained from q by replacing every atom α in $\mathsf{conj}(\vec{x})$ with $\mathsf{unfold}(\alpha)$, while for a union of conjunctive queries $q(\vec{x}) = q_1(\vec{x}) \vee \cdots \vee q_n(\vec{x})$, we define

$$\mathsf{unfold}(q(\vec{x})) = \mathsf{unfold}(q_1(\vec{x}))\ \mathsf{UNION} \cdots \mathsf{UNION}\ \mathsf{unfold}(q_n(\vec{x})). \tag{11}$$

To illustrate the case of unfolding of an aggregate atom, consider the set of mappings $\mathcal{M}$ given in Example 9 and the atom $(\geq_{0.9}\ (\mathsf{min}\ \mathit{testScore}))(x)$. By (10), to obtain $\mathsf{unfold}\big((\geq_{0.9}\ (\mathsf{min}\ \mathit{testScore}))(x)\big)$, we first need to obtain $\mathsf{unfold}(\mathit{testScore}(x, y))$, where y is a fresh variable. By (9), this latter expression corresponds to the union of the SQL queries in $\mathcal{M}$ defining $\mathit{testScore}$, that is,

$$\mathsf{unfold}(\mathit{testScore}(x, y)) = \mathsf{sql}_2(x, y)\ \mathsf{UNION\ ALL}\ \mathsf{sql}_3(x, y).$$

Letting now $E = \geq_{0.9}(\mathsf{min}\ \mathsf{unfold}(\mathit{testScore}(x, y)))(x)$, we obtain the unfolding $\mathsf{unfold}\big((\geq_{0.9}\ (\mathsf{min}\ \mathit{testScore}))(x)\big)$ as the SQL query $\mathsf{sql}_E(x)$ defined in (2):

$$\begin{aligned} \mathsf{sql}_E(x) = \ & \mathsf{SELECT}\ \ x\ \ \mathsf{FROM} \\ & \quad (\mathsf{sql}_2(x, y)\ \mathsf{UNION\ ALL}\ \mathsf{sql}_3(x, y)) \\ & \mathsf{GROUP\ BY}\ \ x\ \ \mathsf{HAVING}\ \ \mathsf{min}(y) \geq 0.9. \end{aligned}$$

Finally, the reformulation of query $q(x)$:- $\mathit{Reliable}(x)$ over the database schema defined with respect to the ontology $\mathcal{O}$ and mappings $\mathcal{M}$ specified in Example 9 corresponds to query $\hat{q}(x)$ below that is obtained from $q(x)$ by unfolding its rewriting $\bar{q}(x)$ specified in (8):

$$\begin{aligned} \hat{q}(x) &= \mathsf{unfold}(\bar{q}(x)) \\ &= \mathsf{unfold}(\mathit{Reliable}(x))\ \mathsf{UNION} \\ &\quad\ \mathsf{unfold}\big((\geq_{0.9}\ (\mathsf{min}\ \mathit{testScore}))(x)\big) \\ &= \mathsf{sql}_1(x)\ \mathsf{UNION}\ \mathsf{sql}_E(x). \end{aligned}$$

Let us now stress the distinction between the SQL operators UNION and UNION ALL. The former computes the set union of its operands and removes duplicate tuples. The latter computes the so-called arithmetic union of its operands resulting in a bag that assigns to each tuple a multiplicity corresponding to the sum of the multiplicities that this tuple has in the bag operands. Given that we care for aggregating over attributes, the use of operator UNION ALL is crucial in unfolding an attribute. On the other hand, the operator of UNION is more appropriate for interpreting the connective of disjunction appearing in rewritings of queries, where the semantics is set-based.

The following example verifies the correctness of the transformation described above.

Example 10. *Recall the extended OBDA setting $(\mathcal{B}, \mathcal{M}, \mathcal{O})$ specified in Example 9 and the certain answers to query $q(x)$:- Reliable(x) over $(\mathcal{B}, \mathcal{M}, \mathcal{O})$. We next compute the answers to $\hat{q}$ over the bag database instance $\mathcal{B}$. Recall that the answers to queries sql_1, sql_2, and sql_3 over $\mathcal{B}$ have already been computed in Example 9. We next compute the answers to the subquery* $\mathsf{sql}_2(x, y)$ UNION ALL $\mathsf{sql}_3(x, y)$ *mentioned in the* FROM *clause of query sql_E. These correspond to the bag $\{\!|(s_1, 0.9) : 2, (s_2, 0.95) : 1, (s_2, 0.98) : 1, (s_3, 0.5) : 1, (s_3, 0.9) : 1|\!\}$; thus sql_E evaluates to bag $\{\!|s_1 : 1, s_2 : 1|\!\}$. Combining the above results, the answers to $\hat{q}$ over $\mathcal{B}$ are given by the bag $\{\!|s_0 : 1, s_1 : 1, s_2 : 1|\!\}$.*

We are now ready to prove correctness of the reformulation procedure for conjunctive queries over extended OBDA settings.

Proposition 5. *For any extended OBDA setting $(\mathcal{B}, \mathcal{M}, \mathcal{O})$, any conjunctive query q of arity k, and any tuple $\vec{t}$ from $(\Gamma \cup D)^k$, where $\bar{q}$ is the output of* PerfectRef *on inputs q and $\mathcal{O}$ while $\hat{q}$ is the result of unfolding $\bar{q}$ with $\mathcal{M}$, we have that $\vec{t} \in \mathsf{cert}(q, (\mathcal{B}, \mathcal{M}, \mathcal{O}))$ if and only if $\mathsf{ans}(\hat{q}, \mathcal{B})(\vec{t}) = 1$.*

Proof. (Sketch) By Propositions 3 and 4, we have $\mathsf{cert}(q, (\mathcal{B}, \mathcal{M}, \mathcal{O})) = \mathsf{cert}(q, \mathcal{O}, \mathcal{D}_{\mathcal{M},\mathcal{B}}) = \mathsf{ans}(\bar{q}, \mathcal{D}_{\mathcal{M},\mathcal{B}})$, thus, it suffices to show that $\vec{t} \in \mathsf{ans}(\bar{q}, \mathcal{D}_{\mathcal{M},\mathcal{B}})$ if and only if $\mathsf{ans}(\hat{q}, \mathcal{B})(\vec{t}) = 1$, for every $\vec{t}$ in $(\Gamma \cup D)^k$. Given the one-to-one correspondence between the conjunctive queries in q_i in $\bar{q}$ and the SQL queries sql_i in the union of SQL queries in $\hat{q}$ as well as the one-to-one correspondence between an atom α in q_i and its unfolding $\mathsf{unfold}(\alpha)$ in sql_i, it suffices to show that $\vec{t} \in \mathsf{ans}(\alpha, \mathcal{D}_{\mathcal{M},\mathcal{B}})$ if and only if $\mathsf{ans}(\mathsf{unfold}(\alpha), \mathcal{B})(\vec{t}) = 1$, for each such atom. This can be shown easily by contrasting the definition of virtual datasets with Equations (9), (10), and (2). □

5.4. Transformation of Streaming Queries

The streaming part of a STARQL query may involve 'static' concepts and roles such as *Rotor* and *testRotor*, that is, concepts and roles that are mapped into static data, and 'dynamic' ones such as *hasValue* that are mapped into streaming data.[6] Mappings for the static ontological vocabulary are classical and discussed above. Mappings for the dynamic vocabulary are composed from the mappings for attributes and the mapping schemata for STARQL query clauses and constructs. The mapping schemata rely on user defined functions of $\text{SQL}^{\oplus}$ and involve windows and sequencing parameters specified in a given STARQL query which make them dependent on time-based relations and temporal states. Note that the latter kind of mappings is not supported by traditional OBDA systems.

For instance, a mapping schema for the '`GRAPH i`' STARQL construct (see Line 16, Figure 3) can be defined based on the following classical mapping that relates a dynamic attribute *ex*:*hasVal* to the table *Msmt* about measurements that among others has attributes *sid* and *sval* for storing sensor IDs and measurement values:

$$ex{:}hasVal(Msmt.sid, Msmt.sval) \leftarrow \\ \textsf{SELECT}\ Msmt.sid, Msmt.sval\ \ \textsf{FROM}\ Msmt.$$

The actual mapping schema for '`GRAPH i`' extends this mapping as follows:

$$\textsf{GRAPH}\ i\ \{?sensor\ \ ex{:}hasVal\ ?y\} \leftarrow \\ \textsf{SELECT}\ sid\ as\ ?sensor,\ \ sval\ as\ ?y \\ \textsf{FROM}\ \textsf{Slice}(Msmt, i, r, sl, st),$$

where the left part of the schema contains an indexed graph triple pattern and the right part extends the mapping for *ex*:*hasVal* by applying a macro function *Slice* that describes the relevant finite slice of the stream *Msmt* from which the triples in the i^{th} RDF graph in the sequence are produced and uses the parameters such as the window range r, the slide sl, the sequencing strategy st and the index i. (See [43] for further details.) Due to the various possible sequencing strategies st, the representation of the r.h.s. in a closed pure $\text{SQL}^{\oplus}$ form (not using any macro function) would become bulky. However, if the sequencing strategy is standard sequencing, then the neat representation as given in Example 8 results. Note that now the mapping has a pure $\text{SQL}^{\oplus}$ r.h.s.

```
GRAPH i {?sensor ex:hasVal ?y} ←
    SELECT sid as ?sensor, sval as ?y, wid as i
    FROM
      [ SELECT * FROM
        ( TIMESLIDINGWINDOW
            time window r
            frequency sl
            SELECT * FROM(http ip−of−Msmt)
        )
      ];
```

[6]Note that we refer here to elements of ontological vocabulary as 'static' and 'dynamic' in order to emphasise that it is mapped to static or dynamic data.

More details on the whole transformation process can be found in our paper [28] which concerns the completeness and correctness of the rewriting step, and in [43], which describes the unfolding and implementation step.

5.5. Correctness of Query Transformation Procedures

Due to the separation property (Equation (3)) of STARQL queries $Q_{\textsf{starql}} \approx Q_{\textsf{StatCQ}} \wedge Q_{\textsf{Stream}}$ we define semantics of STARQL queries over *DL-Lite*$_{\mathcal{A}}^{\textsf{agg}}$ queries over OBDA settings by separately defining the semantics of first static queries $Q_{\textsf{StatCQ}}$ then of streaming queries $Q_{\textsf{Stream}}$ and then combining them in the epistemic fashion by making the join between the certain answers. Note that the epistemic approach has been already considered for classical OBDA settings [44] when one defines semantics of query answering for queries that are more expressive than the class of conjunctive queries.

Therefore, in order to show correctness of the query transformation procedure in Equation (7) it is enough to show correctness of two transformations: of the query $Q_{\textsf{StatCQ}}$ and of $Q_{\textsf{Stream}}$. Correctness of the transformation for $Q_{\textsf{StatCQ}}$ follows from Proposition 5. The main reason for correctness of the rewriting process for $Q_{\textsf{Stream}}$ relies in the semantics the HAVING clause: The GRAPH triples in the internal state, which are constructed in the window, are answered independently. This guarantees the local rewriting in each state. In this aspect of separated consideration of states, STARQL is quite similar to the language of TCQs in [29]. Moreover, in [28] it is shown that the additional step of abstraction induced by the sequencing step poses no problem in the rewriting process. Considering the unfolding process, [43] argues for its completeness and correctness using the fact that the HAVING clauses of STARQL implement a safe fragment of first-order logic, as shown in [37], and hence enable a translation into SQL.

5.6. Discussion: Practicality of Aggregate Concepts

Despite the fact that one can encode aggregate concepts as atomic with the help of mappings as discussed above, we argue, that this encoding has practical disadvantages compared to aggregate concepts.

Indeed, in the case of aggregate concepts, the SQL query $\textsf{sql}_{\circ_r(\textsf{agg unfold}(F))}(x)$ that maps $E = \circ_r(\textsf{agg}\ F)$ to data is computed on the fly during query transformation by 'composing' the mapping for the unfolded attribute F and the query for the 'aggregate context' of F, $\circ_r(\textsf{agg}\ \star)$, in E. Thus, $\textsf{sql}_{\circ_r(\textsf{agg unfold}(F))}(x)$ is not actually stored by the query transformation system as it depends on the definition of F in the ontology and some relevant mappings and may change when the ontology or mappings are modified. At the same time, if one encodes E with a fresh concept A_E and a mapping $A_E(x) \leftarrow \textsf{sql}_{\circ_r(\textsf{agg unfold}(F))}(x)$ and stores them, then one would have to ensure that each further modification in the ontology and mappings relevant to F are propagated in $\textsf{sql}_{\circ_r(\textsf{agg unfold}(F))}(x)$.

Another benefit of using aggregate concepts instead of aggregate queries in mappings is that the former approach offers more flexibility in terms of modelling. Indeed, consider a data property *HasTemperature*. One can map it to data sources with potentially many non-aggregate mappings and then a knowledge engineer can define various aggregate concepts required by applications (i.e., with `avg` or `max` temperatures) over this property using only ontological terms. This approach does not require to write mappings with complex SQL queries for each new aggregation required by applications.

6. System

In this section we discuss our system that implements the OBDA extensions proposed in Section 3. In Figure 8 (Left), we present the overall architecture of our system. On the application level one can formulate STARQL queries over analytics-aware ontologies and pass them to the query compilation module that performs query rewriting, unfolding, and optimisation. Query compilation components can access relevant information in the ontology for query rewriting, mappings for query unfolding, and source specifications for optimisation of data queries. Compiled data queries are sent to a query execution layer that performs distributed query evaluation over streaming and static data, post-processes query answers, and sends them back to applications. In the following we will discuss two main components of the system, namely, our dedicated STARQL2SQL$^\oplus$ translator that turns STARQL queries to SQL$^\oplus$ queries, and our native data-stream management system ExaStream that is in charge of data query optimisation and distributed query evaluation.

6.1. STARQL *to* SQL$^\oplus$ *Translator*

Our translator consists of several modules for transformation of various query components and we now give some highlights on how it works. The translator starts by turning the window operator of the input STARQL query and this results in a *slidingWindowView* on the backend system that consists of columns for defining *windowID* (as in Figure 10) and *dataGraphID* based on the incoming data tuples. Our underlying data-stream management system ExaStream already provides *user defined functions* (UDFs) that automatically create the desired streaming views, e.g., the *timeSlidingWindow* function as discussed below in the ExaStream part of the section.

The second important transformation step that we implemented is the transformation of the STARQL `HAVING` clause. In particular, we normalise the `HAVING` clause into a relational algebra normal form (RANF) and apply the described slicing technique illustrated in Section 5, where we unfold each state of the temporal sequence into slices of the *slidingWindowView*. For the rewriting and unfolding of each slice, we make use of available tools using the OBDA paradigm in the static case, i.e., the Ontop framework [6]. After unfolding, we join all states together based on their temporal relations given in the `HAVING` sequence.

6.2. ExaStream *Data-Stream Management System*

Data queries produced by the STARQL2SQL$^\oplus$ translation, are handled by ExaStream a *Data Stream Management System* (DSMS), which is embedded in Exareme[7], a system for elastic large-scale dataflow processing in the cloud [20, 21].

ExaStream is built as a streaming extension of the SQLite database engine, taking advantage of existing Database Management technologies and optimisations. It provides the declarative language SQL$^\oplus$ (Section 4) for querying data streams and relations. The user can define complex dataflows in SQL$^\oplus$ and the system's *query planner* is responsible for choosing an optimal plan depending on the query, the available stream/static data sources, and the execution environment. ExaStream's optimiser makes it possible to process SQL$^\oplus$ queries that blend streams with static and historical data (e.g., archived streams).

ExaStream's processing engine is built as a streaming extension of SQLite being able to execute relational operations on worker nodes. SQLite has some distinctive features that fit our objectives [45, 46]: *(i) Manifest typing:* instead of static attribute typing, SQLite allows to manifest typing where the datatype is a property of the value itself. This is the most beneficial for the stream processing case, since we cannot know a priori a stream's datatype. *(ii) Single Database File* and *Variable-length records:* an SQLite database stores data in ordinary disk files that can be located anywhere in the directory hierarchy. These files can be easily shared in a distributed environment. Also the fact that SQLite allows for variable-length records, which results in smaller database files, makes the database run faster and allows to minimise data transfer between ExaStream's worker nodes. *(iii) The APSW Python wrapper*[8] allows to easily extend the SQLite database engine with UDFs implemented in python. We are able to use python to implement *virtual tables*, *aggregate* and *row* functions. *(iv) Compactness:* the whole SQLite library with everything enabled is less than 500 KB in size. This feature facilitates the elastic model of ExaStream by allowing to initialise new VMs running SQLite with minimum data transfer.

ExaStream supports *parallelism* by allocating processing across different workers in a distributed environment. Its architecture is shown in Figure 8(Right). Queries are registered through the Gateway Server. Each registered query passes through the ExaStream *Parser* and then is send to its *Query Planner*. The Query planner decides for an efficient order to execute SQL operators, i.e. optimal query plan, and feeds it to the *Scheduler* module. The Scheduler places data and compute operators (including UDFs and relational plans) on workers nodes based on each worker's load. These operators are executed by an

[7] `http://madgik.github.io/exareme/`
[8] https://github.com/rogerbinns/apsw

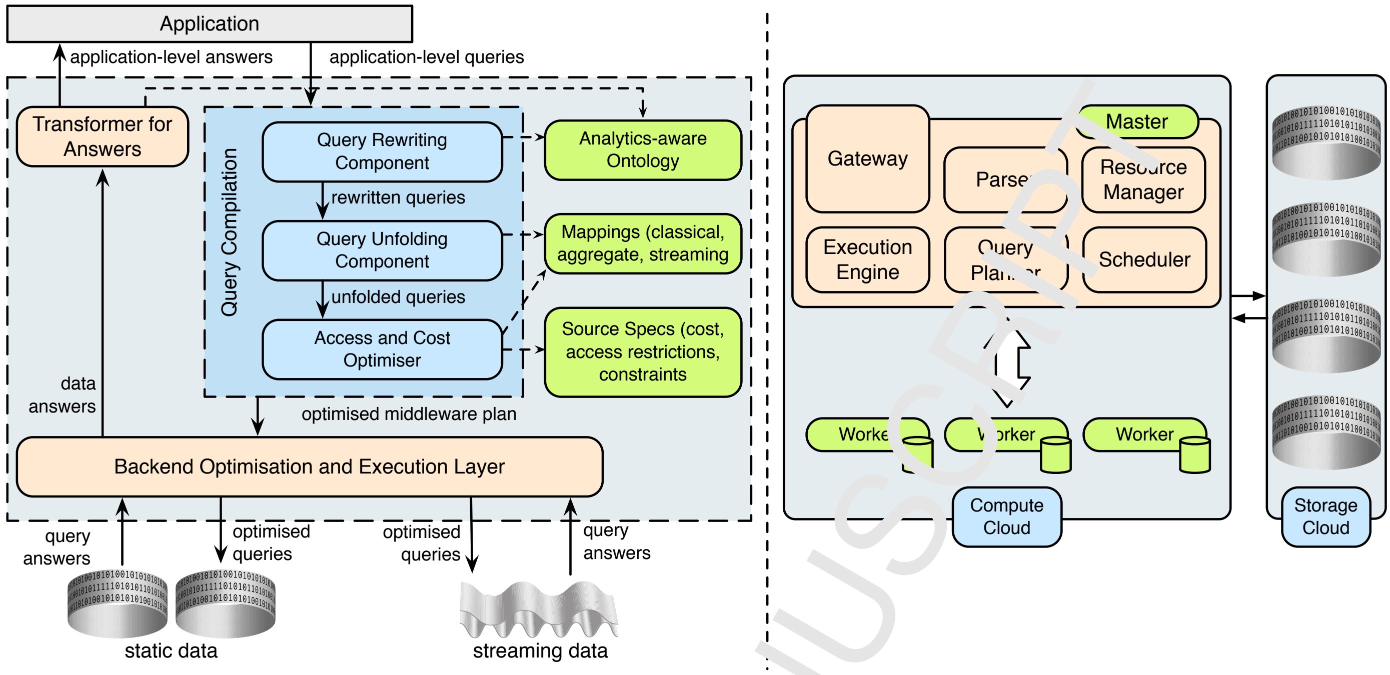


Figure 8: (Left) Overall architecture. (Right) Distributed stream engine of ExaStream

SQLite[9] database engine instance running on each worker.

ExaStream offers different types of parallelism depending on the type of operations performed within a query. Inter-query parallelism is supported for queries with an exclusively streaming input. This means that all the operations of a single query are executed on the same worker, while parallelism is achieved by distributing queries across workers. For example, for a set of queries $q_1, \ldots, q_n$ on streaming input and a set of workers $w_1, \ldots, w_k$, the query planner assigns each query to a specific worker. For computational nodes with a static input, ExaStream provides intra-query parallelism. This means that each operation of a query is distributed on multiple workers. E.g., for an hybrid operation that refers to an analytical task involving live-stream and static data: *(i)* the query planner will have the static data distributed across workers; *(ii)* each consecutive window of the live-stream will be sent to all workers; *(iii)* the operation will be executed on each worker for a different part of the static information and latter combined to form the final answer.

ExaStream offers *query planners* that allow to efficiently execute queries in a declarative language, such as SQL, without any concern for low-level execution details. Our query planner extends the one provided by SQLite in order to handle stream processing continuous queries. It should be noted that the *stream query planner* is responsible for handling local node computations.

SQLite computes joins adopting nested loops, using one loop for each table in the join. One or more indices might be used on the inner loops to accelerate the search, or a loop might be a *full-table scan* that reads every row in the table. Thus, query planning decomposes into two main subtasks: picking the nested order of the various loops; choosing good indices for each loop.

When a query accesses streaming data, SQLite should not make a full scan over an inner stream, or build a *B*-tree index on it. This is because streams are a relational representation of infinite records and therefore the two previous operations would never end, making the resulting plans non-terminating. Therefore we always push streams to the top of query plan trees, i.e. when joining one stream with a static table, the static table is forced to be in the inner loop.

The indexing structures and optimisations presented in Section 7.1.1 are integrated to the ExaStream's query planner.

7. Backend Query Optimisations for SQL$^{\oplus}$

Since a STARQL query consists of analytical static and streaming parts, the result of its transformation by the rewrite and unfold procedures is an analytical data query that also consists of two parts and accesses information from both live streams and static data sources. A special form of static data are archived-streams that, though static in nature, accommodate temporal information that represents the evolution of a stream in time. Therefore, our analytical operations can be classified as:

(i) *live-stream operations* that refer to analytical tasks involving exclusively live streams;

(ii) *static-data operations* that refer to analytical tasks involving exclusively static information;

[9]https://www.sqlite.org

(iii) hybrid operations that refer to analytical tasks involving live-streams and static data that usually originate from archived stream measurements.

For static-data operations we rely on standard database optimisation techniques for aggregate functions. For live-stream and hybrid operations we developed a number of optimisation techniques and execution strategies. These have been incorporated in the ExaStream system described in Section 6. In Section 7.1 we present optimisations regarding live streams; while in Section 7.2 we focus on the system's optimisations for hybrid queries.

7.1. Query Optimisations on Live Streams

SQL$^{\oplus}$ queries access information from both live streams and static data sources. For static-data operations we rely on standard database optimisation techniques. This paragraph focuses on the live-stream optimisations we have developed.

7.1.1. Indexing Structures

Considering the particularities of live-streams with infinite records, we have developed hybrid in-memory indexing structures and algorithms dedicated to accelerating stream-processing. For visualisation purposes, we will assume a *3D space* describing each stream and corresponding to the attributes (`Wid`, `Time`, `Measurement`). The corresponding structures can be applied for higher dimensional spaces.

Our technique considers two levels of indexing: *(i)* the first level, namely $\texttt{WCacheL}_1$, is for performing fast equality operations on the `Wid` attribute based on an hybrid merge/hash-join algorithm *(ii)* the second level, namely $\texttt{WCacheL}_2$, is for accelerating operations on the rest of the attributes, i.e. `Time` and `Measurement` for our description, and is based on a KD-tree structure [47]. KD-trees are in-memory data structures that are very useful for join, range, and nearest neighbour searches. The specific indexing structures were proved to be the most beneficial for the Siemens scenarios that assume join and range operation on non-overlapping windows. For other use cases, different indexing structures can combined with the *Adaptive Indexing Technique* that is presented in Section 7.1.2. We now discuss the indexing structures in more detail.

$\texttt{WCacheL}_1$ *Index.* The $\texttt{WCacheL}_1$ index related to a stream is used for efficiently answering equality constraints on its `Wid` attribute. In particular, we use the $\texttt{WCacheL}_1$ in-memory hash-index with `Wid` as key and the list of tuples that belongs to that specific `Wid` as values. Each bucket on $\texttt{WCacheL}_1$ stores `Wids` in a sorted order, while records on the live stream also appear sorted on the `Wid` attribute —this property of live streams is credited to the `timeslidingWindow` operator.

Example 11. *The left hand side of Figure 9 shows the* $\texttt{WCacheL}_1$ *level of indexing. Bucket 0 contains in sorted order all the wids that have appeared till now and are mapped to the value of* 0*, as we can see both wids in buckets and in the actual stream, are sorted on the* `Wid` *attribute.*

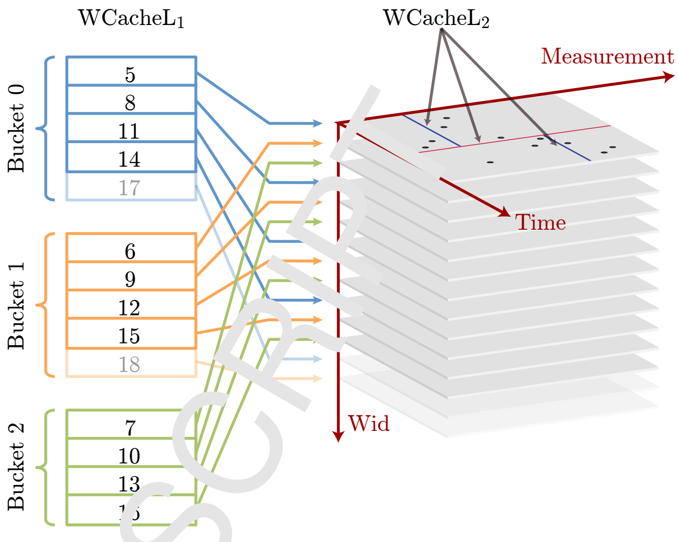


Figure 9: The $\texttt{WCacheL}_1$ and $\texttt{WCacheL}_2$ index structures

Because a stream is infinite, we need a mechanism to ensure that our hash-structure *moves* forward in time. This mechanism adds wids to the $\texttt{WCacheL}_1$ index, as soon as they appear in the stream. Since live streams arrive sorted on the `Wid` attribute, the $\texttt{WCacheL}_1$ related to it can be easily updated by inserting each new *wid* to the bottom of its corresponding hash-bucket.

Example 12. *In Figure 9 the wids* 17 *and* 18 *are added to the* 0 *and* 1 *buckets, as soon as they appear as records into our stream.*

We will demonstrate how our algorithm exploits the $\texttt{WCacheL}_1$ structure for a simple equi-join on two streams. The outer stream of the join operation makes a scan to its data and visits the $\texttt{WCacheL}_1$ of the inner one. If the outer stream scans the wid w and $\texttt{WCacheL}_1$ contains the finite set of wids denoted with $\mathcal{W}$ the following cases may occur:

(i) $w \leq \mathsf{max}(\mathcal{W})$ and $w \notin \mathcal{W}$: In that case w does not appear as a value in the $\texttt{WCacheL}_1$-index and consequently in the `Wid` attribute of $\texttt{Stream}_{\texttt{inner}}$. Since values in $\texttt{Stream}_{\texttt{inner}}$ are ordered in `Wid`, we can safely assume that the window w will never appear as part of the inner stream and therefore the joining condition will never be satisfied for the w window.

(ii) $w \in \mathcal{W}$: In that case we search the corresponding bucket of $\texttt{WCacheL}_1$ that contains the value of w. Since windows are stored in a sorted order per bucket, the algorithm searches for w using a merge-join algorithm. When w is found, our algorithm will return all the tuples in $\texttt{Stream}_{\texttt{inner}}$ that belong to the specific window.

(iii) $\mathsf{max}(\mathcal{W}) < w$: In that case our algorithm will pull more tuples from the inner stream until we get a wid

that is greater than the outer tuple's wid and then operate as in one of the previous cases.

It should be noted that the joining algorithm on window identifiers is hybrid hash/merge-join since it takes advantage of a hash-index and the ordering of elements per hash-bucket.

Example 13. *Suppose that two streams contain a* `Wid`*, a* `Time`*, and a* `Measurement` *attribute and an equi-join is performed between the measurement attributes. Let's also assume that the record of the outer stream that is being examined has a* `Wid` *value of* 9 *and a* `Measurement` *value of* $450°C$*. In order to find if the same temperature appears within the 9th window of the inner stream, the value of the window id 9 is hashed to the Bucket 2 in Fig. 9. Since the value appears in the Bucket 2 of the inner stream, we examine if the corresponding temperature appears in the second level of storage, i.e.* $\texttt{WCacheL}_2$*, that hold all the information about* `Wid` *9 within a KD-tree structure. Using the KD-tree we can decided if the latter is the case.*

$\texttt{WCacheL}_2$ *Index.* The second level of indexing ensures the acceleration of data retrieval operations for attributes other than `Wid`. This index is nested on each window and we have adopted a KD-tree structure [47] for indexing in the rest of the dimensions that participate in a join between two streams. Each level of a KD-tree partitions the space into two subspaces. The partitioning is done along one dimension at the node at the top level of the tree, along another dimension in nodes at the next level, and so on, cycling through the dimensions. The partitioning proceeds in such a way that, at each node, approximately one-half of the points stored in the subtree fall on one side and one-half fall on the other. Partitioning stops when a node has less than a given maximum number of points. Since KD-trees are linear in the size of the data, the memory consumption will also be linear in the size of the incoming information.

Example 14. *The right part of Figure 9 shows how a two level KD-tree partitions the (Time, Measurement) space. The red line performs a data partitioning on the* `Time`*-axis, each partition containing 6 records. Then the blue lines perform data partitioning on the* `Measurement`*-axis, each partition containing exactly 3 records.*

It should be noted that the second level of indexing is dynamically created based on the *Adaptive Stream Indexing* technique that is described next.

7.1.2. Adaptive Stream Indexing

The *Adaptive Stream Indexing* technique is responsible for creating on the fly the appropriate $\texttt{WCacheL}_2$ structures that will accelerate execution of live-stream operations. This means that a KD-tree structure will only be created if the system's optimiser decides it beneficial for the query execution on the specific window of a stream. Formally, let's assume a set of stream-join operations that all have stream s as the inner relation of the join computation:

$$\bigcup_{i=1}^{\nu}\{s_i \bowtie_{\theta_i} s\}.$$

Moreover each join condition θ_i contains the conjunct $\texttt{Wid}_{s_i} = \texttt{Wid}_s$ [10]. Our problem constitutes in finding whether it is beneficial for the query execution speed to build a secondary level of KD-tree index on the attributes of s that appear in all θ_i conditions.

The *adaptive indexing* algorithm operates in two steps:

Step 1. With each new window w appearing in stream s, our algorithm first estimates the number of records that have a `Wid` of value w for all streams under consideration. The function $recs(t, w)$ that makes the estimation takes as input a stream t and the wid w. If all the records of stream t with a wid of w have already appeared, i.e. a record with a wid $w+1$ exists, our algorithm returns the actual number of records in window w. Otherwise, the number of records during the wth window is estimated based on what happened during the last n windows (where n has a default value of 10 but can be altered depending on the use case).

Step 2. The second step of the algorithm estimates whether it is beneficial to build a KD-tree index on the new window of stream s. If we assume that *(i)* the cost of computing the join operation between s_i and s on the wth window without any KD-tree index is denoted with $cost(s_i \bowtie_{\theta_i} s)$, *(ii)* the cost of performing the join operation on the wth window when having a KD-tree structure is denoted with $cost_{KD}(s_i \bowtie_{\theta_i} s)$, *(iii)* and the cost of building the actual KD-tree on the wth window of stream s is denoted with $cost_{KD}(s)$, then the algorithm decides that creating a KD-tree index is beneficial whenever:

$$\sum_{i=1}^{\nu} cost(s_i \bowtie_{\theta_i} s) > \sum_{i=1}^{\nu} cost_{KD}(s_i \bowtie_{\theta_i} s) + cost_{KD}(s).$$

With k the dimensionality of the s stream, n_i the number of tuples within the wth window of stream s_i and n the number of tuples within the wth window of stream s, the cost of building the KD-tree is $\mathcal{O}(k \cdot n \cdot log(n))$, while the cost of performing a join operation using a multidimensional KD-tree index is $\mathcal{O}(n_i \cdot k \cdot n^{1-\frac{1}{k}})$. Details on KD-trees and their corresponding cost functions can be found in [47].

7.2. Query Optimisations on Archived Information

This section focuses on optimisations we have developed on hybrid operations between streaming and static data.

[10] Our algorithm also works for $\texttt{Wid}_{s_i} = \texttt{Wid}_s + d_i$ conditions.

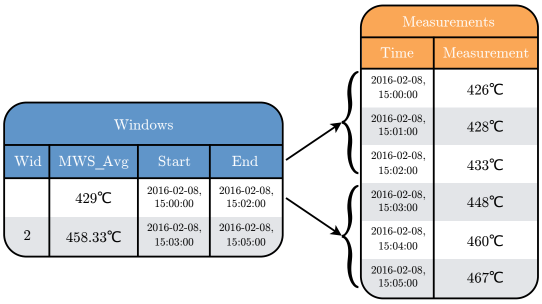


Figure 10: Schema for storing archived streams and MWSs

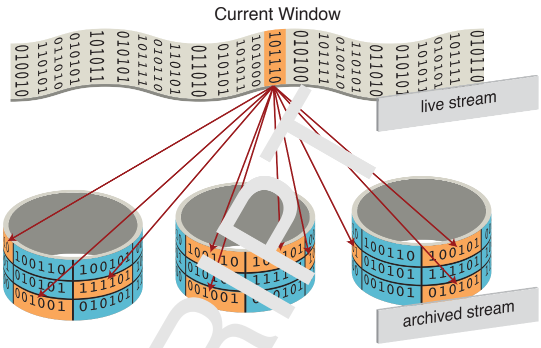


Figure 11: Pearson Correlation coefficient between live and archived streams with embedding the LSH technique in MWSs

7.2.1. Efficient Storage of Archived Streams for Hybrid Operations

Our approach for storing archived streams and performing hybrid operations on them, separates the actual stream from the windowing mechanisms that are applied on it. Consider the relational schema depicted in Figure 10 for storing archived streams and performing hybrid operations on them. The relational table `Measurements` represents the archived part of the stream and stores the temporal identifier (`Time`) of each measurement and the actual values (attribute `Measurement`). The relational table `Windows` identifies the windows that have appeared up till now based on the existing window-mechanism. It contains a unique identifier for each window (`Wid`) and the attributes that determine its starting and ending points (`Window_Start`, `Window_End`). The necessary indices that will facilitate the complex analytic computations are materialised.

Example 15. *In Figure 10 for six measurements we created two windows and for each of them we computed the average of the corresponding measurements.*

The schema that we proposed and illustrated in the example: *(i)* is flexible to query changes since it separates the windowing mechanism —which is query dependent— from the actual measurements; *(ii)* permits execution of multiple queries on the same dataset without the need to replicate the archived streaming data between different windows. Indeed, the flexibility is guaranteed sine each time that the windowing mechanism changes only the `Window` table will be updated and not the, much larger, `Archived_Stream` table that stores the actual stream Moreover, if we have n queries over the same dataset we need $\texttt{Window}_1, \ldots, \texttt{Window}_n$ to identify the window mechanism of each query. Ideally, if $\texttt{Window}_i$, $\texttt{Window}_j$ share the same windowing mechanism, we only need to keep one of the tables.

7.2.2. Materialised Window Signatures

In order to accelerate analytical tasks that include hybrid operations over archived streams, we facilitate precomputation of frequently requested aggregates on each archived window. We name these precomputed summarisations as *Materialised Window Signatures* (MWS). These MWSs are calculated when past windows are stored in the backend and are later utilised while performing complex calculations between these windows and a live stream. The summarisation values are determined by the analytics under consideration. E.g., for the computation of the Pearson correlation, we precompute the *avg* value and *standard deviation* on each archived window measurements; for the cosine similarity, we precompute the *Euclidean norm* of each archived window; for finding the absolute difference between the average values of the current and the archived windows, we precompute the *average* value, etc.

The selected MWSs are stored in the Windows relation with the use of additional columns. In Fig. 10 we see the MWS summary for the `avg` aggregate function being included in the relation as an attribute termed `MWS_Avg`. The application can easily modify the schema of this relation in order to add or drop MWS, depending on the analytical workload.

When performing hybrid operations between the current and archived windows, some analytic operations can be directly computed based on their MWS values with no need to access the actual archived measurements. This provides significant benefits as it removes the need to perform a costly join operation between the live stream and the, potentially very large, `Measurements` relation. On the opposite, for calculations such as the Pearson correlation coefficient and the cosine similarity measures, we need to perform calculations that require the archived measurements as well, e.g., for computing cross-correlations or inner-products. Nevertheless, the MWS approach allows us to avoid recomputing some of the information on each archived window such as its `avg` value and `deviation` for the Pearson correlation coefficient, and the Euclidean norm of each archived window for the cosine similarity measure. Moreover, in case when there is a selective additional filter on the query (such as the `avg` value exceeds a threshold), by creating an index on the MWS attributes, we can often exclude large portions of the archived measurements from consideration, by taking advantage of the underlying index.

Locality Sensitive Hashing. For more complex similarity measures such as the *Pearson correlation coefficient* and the *cosine similarity*, the problem of finding relationship between a live and several archived streams cannot be efficiently solved with the plain use of MWSs . That concern motivates the use of the *locality-sensitive hashing* (LSH) technique and the embedding of LSH information into MWSs.

The premise of the LSH technique is that in many cases it is not necessary to insist on the exact answer; instead, determining an approximate answer with strong accuracy bounds should suffice. The above argument relies on the assumption that approximate similarity search can be performed much faster than the exact one. The key idea is to hash the streams using several hash functions which are chosen so as to ensure that, for each function, the probability of collision is much higher for streams which are similar to each other than for those which are far apart. Then, one can determine similar streams by hashing the query point and retrieving elements stored in buckets containing that point. The LSH technique [48, 49] was introduced for the purposes of devising main memory algorithms for nearest neighbor search. Detailed studies of LSH for live streams and its extensions have been presented in the literature [50, 51].

The combination of MWS and the LSH technique allows to build a smaller summary on what happened during a specific period of time. This summary needs to be build only once for each archived window, while it can be used to compute the similarity between the archived and the current window without the need to access the actual information of the archived data stream. This accelerates similarity operations several orders of magnitude.

We extend MWSs to incorporate LSH information as it is illustrated in Figure 10. For complex similarity measures, the table `Windows` of Figure 10 will be extended to incorporate information related to the LSH hash-values of archived windows by adding the attribute `MWS_LSH`. For each new window arriving from the live-stream the same information is calculated and the live window is only compared to the archived ones that fall into the same bucket, i.e. that are most possible to be similar.

Example 16. *Figure 11 illustrates a correlation example between the current window of a live stream and several archived windows. The LSH algorithm hashes archived windows into two different buckets illustrated with the orange and cyan colours. Since the current window of the stream falls under the orange bucket, there is a high probability to correlate with archived window measurements that are hashed under the same bucket and a low probability to correlate with all other window measurements. Therefore, it will only be correlated with the archived measurements that are hashed to the orange bucket.*

```
PREFIX ex : <http://www.siemens.com/onto/gasturbine/
CREATE PULSE pulse WITH START = NOW, FREQUENCY = 1sec
CREATE STREAM pearsonStream AS
SELECT  pearsonCorrelation(?y,?z), NOW
FROM STREAM
  measurementA [NOW -100sec,NOW]->1sec,
  measurementB [NOW -100sec,NOW ]->1sec
USING PULSE pulse SEQUENCE BY StdSeq AS SEQ1
HAVING EXISTS i in SEQ1 (
   GRAPH i { ex:sensorA :hasValue ?y .
             ex:sensorB :hasValue ?z } )
```

Figure 12: Query V expressed in STARQL

8. Experimental Evaluation of the Backend

The aim of our evaluation is to study how our optimisation techniques and query distribution to multiple workers accelerate the overall execution time of different analytic queries that involve live-stream and hybrid operations.

8.1. Evaluation Setting

We deployed our system to the Okeanos Cloud Infrastructure[11] and used up to 16 virtual machines (VMs) each having a 2.100 GHz processor with two cores and 4 GB of main memory. We used streaming and static data that contain measurements produced by 100,000 thermocouple sensors installed in 950 Siemens power generating turbines.

8.2. Test Queries

For the experimental evaluation, the following queries were adopted:

Query I: The first query computes an equality join on the `Wid` and `Time` attributes between two live-streams.

Query II: This query computes the Pearson correlation of a live stream with a varying number of archived streams.

Queries III & **IV**: These two queries are variations of Query II but, instead of the Pearson correlation, they compute similarity based on either the *average* or the *minimum* values within a window.

We defined such similarities between vectors (of measurements) $\vec{w}$ and $\vec{v}$ as follows: $|\text{avg}(\vec{w}) - \text{avg}(\vec{v})| < 10°C$ and $|\min(\vec{w}) - \min(\vec{v})| < 10°C$. The archived stream windows are stored in the `Measurements` relation, against which the current stream is compared.

Query V: This query calculates the Pearson correlation between two live streams. The STARQL formulation of this query is given in Figure 12.

[11] www.okeanos.grnet.gr/

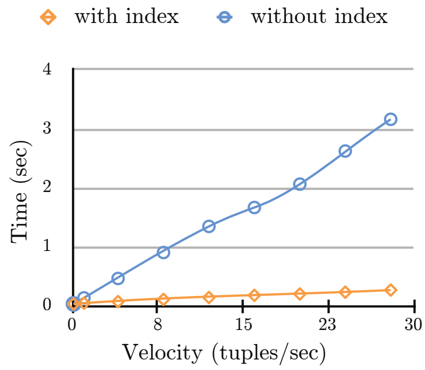


Figure 13: Effect of adaptive indexing

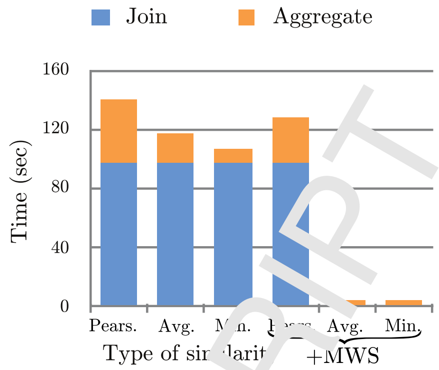


Figure 14: Effect of MWS optimisation

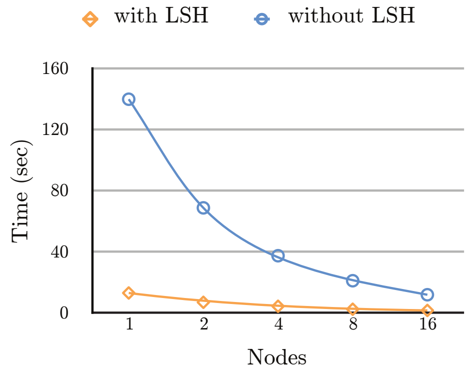


Figure 15: Effect of intra-query parallelism and the LSH technique

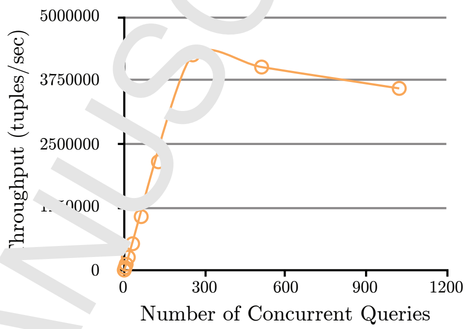


Figure 16: Effect of inter-query parallelism on live-stream

In the remaining part of the section we present the results of our experimental evaluation for each of our optimisations techniques: Adaptive indexing optimisation, MWS Optimisation, Parallelism between live and archived streams, and Parallelism between live streams.

8.3. Adaptive Indexing Optimisation

This experiment is devised to show how the adaptive indexing optimisation and the related indexing structures affect query-response times. We execute *Query I* as follows:

- *(i)* on a single VM-worker;
- *(ii)* processing is performed on windows of 100 secs;
- *(iii)* the evaluation is performed on the live streams A and B (A being the inner relation of the join operation), building an index on stream A whenever appropriate;
- *(iv)* stream A has a velocity of 10 tuples/sec, while we vary the velocity of stream B from 1 tuple/sec to 28 tuples/sec.

In Figure 13, we measured the processing time for computing the join between a pair of windows of stream A and B with and without enabling the adaptive indexing technique that creates the necessary $\texttt{WCacheL}_2$ structures. The horizontal axis displays the velocity of stream B and the vertical axis the window processing time measured as the average of 100 consecutive live-stream execution cycles. We observe that for high throughput, the adaptive indexing techniques performs substantially better then simple join, i.e. in our experiment the adaptive indexing technique performs 12 times faster for a 28 tuples/sec throughput.

For the Adaptive Indexing optimisation, we did not perform an experiment dedicated to the size of the corresponding window, since, increasing the window size has a similar effect to changing the velocity of each stream.

8.4. MWS Optimisation

This set of experiments is devised to show how the MWS optimisation affects the query's response time. We executed test *Queries II*, *III*, and *IV*:

- *(i)* on a single VM-worker;
- *(ii)* for a fixed live-stream velocity of 1 tuple/min;
- *(iii)* for a fixed window size of 1 hour which corresponds to 60 tuples of measurements per window;
- *(iv)* and the current live stream window was measured against 100,000 archived ones.

We measured the window processing time with and without the MWS optimization. In Figure 14 we present the results of our experiments. The reported time is the average of 15 consecutive live-stream execution cycles. The horizontal axis displays the three test queries with and without the MWS optimisation, while the vertical axis measures the time it takes to process 1 live-stream window against all the archived ones. This time is divided to the time it takes to join the live stream and the `Measurements` relation and the time it takes to perform the actual computations. Observe that the MWS optimisation reduces the time for the Pearson query by 8.18%. This is attributed to the fact that some computations (such as the avg and standard deviation values) are already available in the `Windows` relation and are, thus, omitted. Nevertheless, the join operation between the live stream and the very large `Measurements` relation that takes 69.58% of the overall query execution time can not be avoided. For the other two queries, we not only reduce the CPU overhead of the query, but the optimiser further prunes this join from the query plan as it is no longer necessary. Thus, for these queries, the benefits of the MWS technique are substantial.

It should be noted that for hybrid operations the effect of the MWS optimisation becomes more substantial for larger window sizes. Therefore, increasing the size of the window would further improve the contribution of the MWS technique on hybrid operations, especially for the cases when the archived streams are not accessed, e.g. when computing the minimum or average aggregate functions or when using the LSH technique to compute similarity measures (see Subsection 8.7 for the corresponding experiments using LSH optimisations).

8.5. Parallelism Between Live and Archived Streams

Since the MWS optimisation substantially accelerates query execution for the two test queries that rely on average and minimum similarities, query distribution would not offer significant benefit, and thus these queries were not used in the third experiment. For complex analytics such as the Pearson correlation that necessitates access to the archived windows, the ExaStream backend permits us to accelerate queries by distributing the load among multiple worker nodes. In the third experiment we use the same setting as before for the Pearson computation without the MWS technique, but we vary this time the number of available workers from 1 to 16. In Figure 15, one can observe a significant decrease in the overall query execution time as the number of VM-workers increases. ExaStream distributes the `Measurements` relation between different worker nodes. Each node computes the Pearson coefficient between its subset of archived measurements and the live stream. As the number of archived windows is much greater than the number of available workers, intra-query parallelism results in significant decrease of the time required to perform the join operation.

8.6. Parallelism Between Live Streams

This experiment focuses on the effect of accelerating live-stream operations by distributing the load to multiple worker nodes via inter-query parallelism. We executed *Query V* (Pearson correlation)

(i) for a varying number of 1 to 1024 of concurrent queries between different pairs of live streams;

(ii) for a fixed window size of 60 tuples;

(iii) on non-overlapping windows;

(iv) using 128 ExaStream worker nodes.

We measured the window throughput, as the number of stream tuples that are processed per sec. Recall that each node is equipped with a two-core processor. We can see from Figure 16 that initially, the overall throughput of the system increases linearly with the number of queries. This is because ExaStream utilizes the available workers and distributes the load evenly among them. When the number of queries reaches the number of cores available (256), we observe the maximum throughput of 4,250,226 $tuples/sec$. From that point onward, the additional queries injected in Exareme result in multiple queries sharing the same core and, as a result, the cumulative throughput decreases. It should be noted that the Adaptive Indexing Technique creates the corresponding indexing structures whenever it is beneficial for the aforementioned operations. For a larger number of concurrent queries/streams, we can obtain even better performance by utilizing the LSH technique, discussed next.

8.7. LSH *Optimisation*

Our final experiment focuses on the LSH technique and how the intermix of MWSs, LSH buckets, and parallelism accelerates the computation of complex similarity measures between live and archived streams. We perform the same experiment as in Section 8.5 for *parallelism between live & archived streams*, only this time we employ the LSH variation of MWSs. For the interested reader in the LSH parameterisation we used a combination of 7 AND-constructors and 6 OR-constructors. The results of this experiment are also displayed in Figure 15 that compares performance with and without our optimisation. One can observe a significant decrease in the overall query execution time when we adopt the combination of the MWS and LSH techniques for computing correlation between live and archived streams. The price we have to pay for this increase in performance is 3% of false negative results for finding all Pearson correlations with an equality degree above 0.7.

9. Related Work

OBDA System. Our proposed approach extends existing OBDA systems since they either assume that data is in (static) relational DBs, e.g [12, 6], or streaming, e.g., [9, 10], but not of both kinds. Moreover, we are different from existing solutions for unified processing of streaming and static semantic data, e.g. [52], since they assume that data is natively in RDF while we assume that the data is relational and mapped to RDF. An extension of OBDA tailored towards equipment diagnostics has been recently presented in [53, 54]. They rely on the standard OWL 2 QL ontologies and define a rule-based language over them that has a sort of fixed-point semantics. In contrast, we propose an analytics-aware ontology language $DL\text{-}Lite_{\mathcal{A}}^{\mathsf{agg}}$ and a query language STARQL that has a different expressive power and semantics. Finally, we focus on backend optimisations while they rely on the standard backend solutions for evaluation of diagnostic programs.

Ontology language. The semantic similarities of $DL\text{-}Lite_{\mathcal{A}}^{\mathsf{agg}}$ to other works have been covered in Sec. 3. Syntactically, the aggregate concepts of $DL\text{-}Lite_{\mathcal{A}}^{\mathsf{agg}}$ have counterpart concepts, named local range restrictions (denoted by $\forall F.T$) in $DL\text{-}Lite_{\mathcal{A}}$ [55, 56, 57]. However, for purposes of rewritability, these concepts are not allowed on the left-hand side of inclusion axioms as we have done for $DL\text{-}Lite_{\mathcal{A}}^{\mathsf{agg}}$, but only in a very restrictive semantic/syntactic way. Consequently, most of the results of [55, 56, 57] regarding rewritability of ontology satisfiability and query answering are very relevant for $DL\text{-}Lite_{\mathcal{A}}^{\mathsf{agg}}$ as well.

The semantics of $DL\text{-}Lite_{\mathcal{A}}^{\mathsf{agg}}$ for aggregate concepts is very similar to the epistemic semantics proposed in [23] for evaluating conjunctive queries involving aggregate functions. A different and more intuitive semantics for evaluating conjunctive queries with aggregate functions has been considered in [24] based on minimal models relative to a query, but query answering has been shown to be intractable, while it covers only the aggregate functions count and countd. Interpretations assigning a bag extension to predicates has been considered recently in the context of OBDA [25] and data exchange [26]. In both of these works, the motivation is based on the need for performing aggregation over the integrated database for which duplicates influence the answers and must be retained. The semantics of $DL\text{-}Lite_{\mathcal{A}}^{\mathsf{agg}}$ follows this spirit, but only for the predicates corresponding to attributes, over which aggregation may be performed as a result of the definition of an aggregate concept, which, nonetheless, is given a set extension. In contrast to [25], where satisfaction of TBox axioms is defined based on an extension of the subset relation to bags, $DL\text{-}Lite_{\mathcal{A}}^{\mathsf{agg}}$ retains the more standard, set-based semantics for satisfaction. In this respect, $DL\text{-}Lite_{\mathcal{A}}^{\mathsf{agg}}$ is closer to [24], which adopts standard set-based semantics for TBox axioms.

Last, query answering in $DL\text{-}Lite_{\mathcal{A}}^{\mathsf{agg}}$ is closer to that in $DL\text{-}Lite_{\mathcal{A}}$ rather than the ontology languages in [25, 24]. This is because the latter works are concerned about the computation of the minimum number of matches of the query across all models of the ontology, whereas in $DL\text{-}Lite_{\mathcal{A}}^{\mathsf{agg}}$ we care only for the existence of a match. Closing the discussion on $DL\text{-}Lite_{\mathcal{A}}^{\mathsf{agg}}$, concepts based on aggregates functions were considered in [58] for the description logics $\mathcal{ALC}$ and $\mathcal{EL}$ equipped with concrete domains, but the problem of query answering was not studied there.

Query language. While already several languages and engines for RDF stream reasoning exist, e.g., C-SPARQL [66], RSP-QL [38], SPARQLStream [9], or CQELS [67], only SPARQLStream supports an ontology based data access approach in the classical sense: It uses (pure) query rewriting of the queries in a preprocessing process w.r.t. a DL-Lite TBox—without knowledge of the input data (static data and streaming data). The system described in [68] also exploits rewriting of queries, but uses a different DL language, namely $\mathcal{ELHIO}$. In general, FOL rewritability is not guaranteed for this DL, but the authors consider rewriting for the non-recursive fragment of $\mathcal{ELHIO}$. Unfolding is not relevant for the approach in [68] as the authors consider materialized RDF streams. In comparison to these *OBDA* approaches, STARQL offers more advanced user defined functions from the backend system like Pearson correlation. ([9] at least uses a native inclusion of aggregation functions).

In Tables 1 and 2 we use the setting of features of [x] in order to compare STARQL with the state-of-the-art RDF stream query languages, namely, Streaming SPARQL [59], C-SPARQL [60, 61, 60], CQELS [52], SPARQLStream [9, 62, 63], EP-SPARQL [64], TEF-SPARQL [65], and RSP-QL [38]. Observe that except for *Property Paths*, a new feature of SPARQL 1.1, and *Triple Windows*, STARQL supports all constructors of the languages reported in the tables. In particular, STARQL supports the basic operators such as *Union*, *Join*, *Optional*, and *Filter* that are supported by all other languages in the tables. STARQL also supports the *If Expression*, an SPARQL 1.1 function form that evaluates some boolean condition and outputs one or other expression depending on the outcome of testing the boolean condition. This is supported by C-SPARQL, SPARQLStream, and RSP-QL only. Also, STARQL supports value *Aggregation* and *Time Windowing* as most of the other systems reported in the tables. STARQL supports *W-to-S Operator* on RStreams, that is, it outputs the whole content of the window. Moreover, STARQL allows to declare *Named Streams*, that is, it is possible to define a new stream by a STARQL query that can be referenced by other STARQL queries. This feature is important for our diagnostics use case, because named streams enable a pipe-lined query building methodology which is required to handle in a modularized manner those aspects of various streams that are relevant for diagnostics. Note that among the languages reported in the tables, only C-SPARQL, EP-SPARQL and RSP-QL support named streams.

Observe that STARQL supports a rear feature of *Intra window time* (which is supported only by C-SPARQL, SPARQLStream, and EP-SPARQL), that is, the users can distinguish between different states within a window and order them. This adds the useful abstraction of state-based reasoning on the window contents. Another rear fea-

Table 1: Comparison of RDF-stream query languages (Part 1)
(*) See explanation in main text

Name	Data Model	Union, Join, Optional, Filter	IF Expression	Aggregate	Property Paths	Time Windows	Triple Windows
STREAMING SPARQL [59]	RDF streams	Yes	No	No	No	Yes	Yes
C-SPARQL [60, 61, 60]	RDF streams	Yes	Yes	Yes	Yes	Yes	Yes
CQELS [52]	RDF streams	Yes	No	Yes	No	Yes	No
SPARQLSTREAM [9, 62, 63]	(virtual) RDF streams	Yes	Yes	Yes	Yes	Yes	No
EP-SPARQL [64]	RDF streams	Yes	No	Yes	No	No	No
TEF-SPARQL [65]	RDF streams	Yes	No	Yes	No	Yes	Yes
RSP-QL [38]	RDF streams	Yes	Yes	Yes	Yes(*)	Yes	No (*)
STARQL [37, 19, 28, 18]	(virtual) RDF streams	Yes	Yes	Yes	No	Yes	No

Table 2: Comparison of RDF-stream query languages (Part 2)
(*) See explanation in main text

Name	W-to-S Operator	Named Streams	Intra window time	Sequencing	Pulse
STREAMING SPARQL	RStream	No	No	No	No
C-SPARQL	RStream	Yes	Yes	No	Yes
CQELS	IStream	No	No	No	No
SPARQLSTREAM	RStream, IStream, DStream	No	Yes	No	No
EP-SPARQL	RStream	Yes	Yes	Yes	No
TEF-SPARQL	RStream	No	No	Yes	No
RSP-QL [38]	RStream, IStream, DStream	Yes	Yes	No	No(*)
STARQL	RStream	Yes	Yes	Yes	Yes

ture of STARQL is *Sequencing*, that is, a user can build a sequence of stream elements within a window, which is also supported by EP-SPARQL and TEF-SPARQL. Finally, the last rare feature of STARQL is a *pulse* declaration which handles the synchronization of outputs from multiple streams. C-SPARQL is the only other query language which offers a pulse declaration—using the keyword `EVERY`.

RSP-QL [38] is the most recent suggestion for an RDF query language on streams. It defines an operational semantics for a streaming extension of SPARQL. As such, in principle, it also supports property paths of SPARQL 1.1. But as the language is not explicitly stated in [38] and property paths are not discussed there, the "yes" entry holds under the condition that the add-on stream semantics is separable from the semantics of property paths for ordinary (non-streaming) RDF graphs. Triple windows are not explicitly discussed by [38] and hence we wrote "No" for this feature slot, though a slight adaptation of RSP-QL should also cover these. Regarding the pulse declaration, we add the remark that there is no explicit construct for specifying a pulse in RSP-QL. At the same time, they discuss a different construct to handle the synchronization of different sliding windows: they describe the semantics using an evaluation policy and w.r.t. a starting time t^0 not specified by the query designer but by the implementing system.

EP-SPARQL [64] plays a unique role under the RDF stream languages as it relies on the paradigms of event processing and logic programming. The sequence operator is quite different from that of STARQL. EP-SPARQL uses the sequence operator to identify a sequence pattern in the event stream, whereas in STARQL it is used to build a sequence of RDF graphs from a stream of timestamped RDF elements.

We described with an example the operational semantics of the window operator of STARQL. A full operational model for the STARQL query language and a comparison with the SECRET model described in [69] or with the model of RSP-QL of [38] is saved for future work.

Data Stream Management System. One of the leading edges in database management systems is to extend the relational model to support for continuous queries based on declarative languages analogous to SQL. Following this approach, systems such as TelegraphCQ [70], STREAM [71], and Aurora [72] take advantage of existing Database Management technologies, optimisations, and implementations developed over 30 years of research. In the era of big data and cloud computing, a different class of DSMS has emerged. Systems such as Storm[12], Flink[13], Kafka[14], Heron[15], and Spark Streaming[16] offer an API that allows the user to submit dataflows of user defined operators. EXASTREAM unifies these two different approaches by allowing to describe in a declarative way complex dataflows of (possibly user-defined) operators. It should be noted that several state-of-the-art systems for Big Data processing are adopting a similar approach, providing for declarative

[12] Apache Storm. `http://storm.apache.org`
[13] Apache Flink. `http://flink.apache.org`
[14] Apache Kafka. `https://kafka.apache.org`
[15] Twitter Heron. `https://apache.github.io/incubator-heron`
[16] Spark Streaming. `https://spark.apache.org/streaming`

SQL-like languages for data processing. Apache Spark allows to query structured data inside Spark programs using SQL queries, while KSQL is a streaming SQL engine that enables real-time data processing against Apache Kafka. In Section 10 we explain how to take advantage of recent advances in Big Data processing systems.

In Section 7.1 we have adapted existing indexing structures to accelerate query processing in actual industrial diagnostics and monitoring of equipment in Siemens. We have additionally presented the Adaptive Indexing technique that creates on the fly the appropriate structures for indexing. The specific indexing structures were proved to be the most beneficial for the Siemens scenarios that assume join and range operation on non-overlapping windows. We chose KD-trees [47] because they are in-memory data structures that are very useful for join, range, and nearest neighbour searches. Additionally building KD-tree indexes is much faster compared to other multidimensional indexes such as R-trees [73] and their variations. For scenarios that these conditions do not apply, other indexing structures can be examined in combination with the Adaptive Indexing Technique. Index materialisation strategies have been examined in the current bibliography, e.g. in [74] a methodology for automatically selecting an appropriate set of materialised views and indexes is presented. Our Adaptive Indexing Technique, contrary to other indexing strategies that are focus on static data processing, takes advantage of what happened in the latest windows of a stream in order to decide when to build the corresponding *KD*-tree index. A similar methodology for a different problem has been presented in [75]. In [75] a query processing mechanism reorders operators in a query plan as it runs.

The Materialised Window Signature summarisation, implemented in ExaStream, is inspired from data warehousing techniques for maintaining selected aggregates on stored datasets [76, 77]. Though the idea of Materialise Window Signatures (MWS) appears to be intuitive, the only similar methodology that we found in the bibliography is presented by the state of the art *Data Canopy* system [78]. The Data Canopy system stores basic aggregates within an in-memory data structure and reuses them for overlapping data parts and for various statistical measures. Consider that the work on the Data Canopy was presented subsequently to the our introduction of Materialise Window Signatures [13].

10. Conclusion and Future Work

We see our work as a first step towards the development of a solid theory and new full-fledged systems in the space of analytics-aware ontology-based access to data that is stored in different formats such as static relational, streaming, etc. To this end we proposed ontology, query, and mapping languages that are not only capable of supporting analytical tasks common for Siemens turbine diagnostics, but also we believe to other industrial settings. Moreover, we developed a number of backend optimisation techniques that allow such tasks to be accomplished in reasonable time as we have demonstrated on large scale Siemens data.

We believe that our work will be interesting for a wide range of researchers and practitioners in the area of data integration, semantic data access, and Internet of Things. We also believe that our results will be inspiring for the Semantic Web community in developing new fundamental research as well as efficient algorithms for light-weight ontology languages enhanced with analytical operators. Finally, we believe that the next generation Semantic Systems such as OBDA-based should be in a tight integration with analytics and our work contributes in this direction.

Finally, there is a number of important further research and practical directions that we plan to explore.

From the practical perspective, we plan to extend our OBDA system with several important modules. First, in order to facilitate ontology and mapping development and maintenance we plan to work on novel ontology and mapping bootstrapping [79, 80, 81, 82, 83, 84] and rule learning techniques [85, 86]. Second, in order to facilitate formulation of analytical queries over analytics-aware ontologies we plan to work on end-user oriented query formulation interfaces, e.g., visual query systems [87, 88, 89, 90, 91] or faceted search query interfaces [92, 93, 94, 95, 96, 97, 98, 99]. Third, we plan to investigate access control policies for analytics-aware OBDA [100, 101]

On the side of analytics-aware ontologies, since bag semantics is natural and important in analytical tasks, we see a need in exploring bag instead of set semantics for ontologies as it has been considered recently in OBDA and data exchange [25, 26]. Besides, we plan to study how the semantics and results of [56, 55, 58] and queries of [23] can be adapted to our setting.

On the side of analytics-aware queries, an important further direction is to align them with the terminology of the W3C RDF Data Cube Vocabulary[17] and to provide additional optimisations after the alignment. This direction is important since this will improve the integration of analytical data, produced by other queries with analytical and non-analytical data stemming from further streams or repositories. Moreover, we plan to conduct empirical evaluations to compare STARQL with other such languages. Finally, we plan to investigate extensions of analytics-aware queries with recursion, e.g., following the approach of semantic diagnostic languages [102, 103, 53, 104, 105]

For backend optimisations, our future work involves the adaptive adjustment of ExaStream's topology into the cloud's demands. The rate of input streams may change drastically from time to time. ExaStream's future goal is to keep the utilisation of the cloud always to hight percentages using only the resources that are needed. This affects both the data distribution and ExaStream's stream processing engine. For example our optimiser must support

[17] https://www.w3.org/TR/2014/REC-vocab-data-cube-20140116/

stream join reordering on the fly. The optimiser must take into account the rate of the input tuples and change the order without damaging the adaptive indexing technique and the creation of the related structures. Another interesting backend optimisation relates to the pre-computation of the appropriate structures that will accelerate the aggregate-query execution, e.g. materialised views and database indexes. We intend to examine refined optimisation techniques that combine information on the *OBDA* layer with building of the appropriate structures on our DSMS (or database engine). With the recent advances in stream processing engines and the adoption of declarative languages from several Big Data frameworks, we intend to examine Polystore architectures [106] for data integration of streaming and static information via OBDA solutions.

Acknowledgements

This work was partially funded by the EU project Optique (FP7-ICT-318338) and the EPSRC projects MaSI^3 (EP/K00607X/1), DBOnto (EP/L012138/1), and ED^3 (EP/N014359/1). It was also partially funded by the SIRIUS Centre, Norwegian Research Council project number 237898.

References

[1] D. Calvanese, G. Giacomo, D. Lembo, Ontologies and Databases: The DL-Lite Approach, in: Reas. Web, 2009.

[2] I. Horrocks, M. Giese, E. Kharlamov, A. Waaler, Using semantic technology to tame the data variety challenge, IEEE Internet Computing 20 (6) (2016) 62–66.

[3] C. Bizer, A. Seaborne, D2RQ-Treating Non-RDF Databases as Virtual RDF Graphs, in: ISWC, 2004.

[4] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodriguez-Muro, R. Rosati, M. Ruzzi, D. F. Savo, The MASTRO System for Ontology-Based Data Access., Semantic Web 2 (1) (2011) 43–53.

[5] F. Priyatna, O. Corcho, J. Sequeda, Formalisation and Experiences of R2RML-Based SPARQL to SQL Query Translation Using Morph, in: WWW, 2014, pp. 479–490.

[6] M. Rodriguez-Muro, R. Kontchakov, M. Zakharyaschev, Ontology-Based Data Access: Ontop of Databases., in: ISWC, 2013, pp. 558–573.

[7] K. Munir, M. Odeh, R. McClatchey, Ontology-Driven Relational Query Formulation Using the Semantic and Assertional Capabilities of OWL-DL., Knowl.-Based Syst. 35 (2012) 144–159.

[8] J. Sequeda, D. P. Miranker, Ultrawrap: SPARQL Execution on Relational Data., JWS 22 (2013) 19–39.

[9] J. Calbimonte, Ó. Corcho, A. J. G. Gray, Enabling ontology-based access to streaming data sources, in: ISWC, 2010, pp. 96–111.

[10] L. Fischer, T. Scharrenbach, A. Bernstein, Scalable linked data stream processing via network-aware workload scheduling, in: SSWKBS@ISWC, 2013, pp. 81–96.

[11] D. Calvanese, P. Liuzzo, A. Mosca, J. Remesal, M. Rezk, G. Rull, Ontology-based Data Integration in EPNet: Production and Distribution of Food During the Roman Empire, Eng. Appl. of AI 51 (2016) 212–229.

[12] C. Civili, M. Console, G. De Giacomo, D. Lembo, M. Lenzerini, L. Lepore, R. Mancini, A. Poggi, R. Rosati, M. Ruzzi, V. Santarelli, D. F. Savo, MASTRO STUDIO: managing ontology-based data access applications, PVLDB 6 (12) (2013) 1314–1317.

[13] E. Kharlamov, D. Hovland, E. Jiménez-Ruiz, D. Lanti, H. Lie, C. Pinkel, M. Rezk, M. G. Skjæveland, E. Thorstensen, G. Xiao, D. Zheleznyakov, I. Horrocks, Ontology based access to exploration data at statoil, in: ISWC, 2015, pp. 93–112.

[14] E. Kharlamov, D. Hovland, M. G. Skjæveland, D. Bilidas, E. Jiménez-Ruiz, G. Xiao, A. Soylu, D. Lanti, M. Rezk, D. Zheleznyakov, M. Giese, H. Lie, Y. E. Ioannidis, Y. Kotidis, M. Koubarakis, A. Waaler, Ontology based data access in statoil, J. Web Sem. 44 (2017) 3–36.

[15] E. Kharlamov, N. Solomakhina, Ö. L. Özçep, D. Zheleznyakov, T. Hubauer, S. Lamparter, M. Roshchin, A. Soylu, S. Watson, How Semantic Technologies Can Enhance Data Access at Siemens Energy, in: ISWC, 2014.

[16] E. Kharlamov, M. G. Skjæveland, T. Mailis, E. Jiménez-Ruiz, G. Xiao, A. Soylu, H. Lie, A. Waaler, Finding Data Should be Easier than Finding Oil, in: BigData, 2018.

[17] M. Rodriguez-Muro, D. Calvanese, High Performance Query Answering over DL-Lite Ontologies, in: KR, 2012.

[18] E. Kharlamov, Y. Kotidis, T. Mailis, C. Neuenstadt, C. Nikolaou, Ö. L. Özçep, C. Svingos, D. Zheleznyakov, S. Brandt, I. Horrocks, Y. E. Ioannidis, S. Lamparter, R. Möller, Towards Analytics Aware Ontology Based Access to Static and Streaming Data, in: ISWC, 2016, pp. 344–362.

[19] Ö. L.. Özçep, R. Möller, Ontology based data access on temporal and streaming data, in: M. Koubarakis, G. Stamou, G. Stoilos, I. Horrocks, P. Kolaitis, G. Lausen, G. Weikum (Eds.), Reasoning Web. Reasoning and the Web in the Big Data Era, Vol. 8714. of Lecture Notes in Computer Science, 2014.

[20] M. M. Tsangaris, G. Kakaletris, H. Kllapi, G. Papanikos, F. Pentaris, P. Polydoras, E. Sitaridi, V. Stoumpos, Y. E. Ioannidis, Dataflow Processing and Optimization on Grid and Cloud Infrastructures, IEEE Data Eng. Bull. 32 (1) (2009) 67–74.

[21] H. Kllapi, P. Sakkos, A. Delis, D. Gunopulos, Y. Ioannidis, Elastic Processing of Analytical Query Workloads on IaaS Clouds, in: arXiv, 2015.

[22] C. Lutz, I. Seylan, F. Wolter, Ontology-based data access with closed predicates is inherently intractable(sometimes), in: IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013, 2013, pp. 1024–1030.

[23] D. Calvanese, E. Kharlamov, W. Nutt, C. Thorne, Aggregate Queries Over Ontologies, in: ONISW, 2008, pp. 97–104.

[24] E. V. Kostylev, J. L. Reutter, Complexity of Answering Counting Aggregate Queries Over DL-Lite, J. of Web Sem. 33 (2015) 94–111.

[25] C. Nikolaou, E. V. Kostylev, G. Konstantinidis, M. Kaminski, B. Cuenca Grau, I. Horrocks, The bag semantics of ontology-based data access, in: IJCAI, 2017, pp. 1224–1230.

[26] A. Hernich, P. G. Kolaitis, Foundations of information integration under bag semantics, in: LICS, 2017, pp. 1–12.

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

[28] Özgür. L. Özçep, R. Möller, C. Neuenstadt, Stream-query compilation with ontologies, in: B. Pfahringer, J. Renz (Eds.), Poceedings of the 28th Australasian Joint Conference on Artificial Intelligence 2015 (AI 2015), Vol. 9457 of LNAI, Springer International Publishing, 2015.

[29] S. Borgwardt, M. Lippmann, V. Thost, Temporal query answering in the description logic dl-lite, in: FroCoS, 2013, pp. 165–180.

[30] A. Arasu, S. Babu, J. Widom, The cql continuous query language: Semantic foundations and query execution, VLDBJ 15 (2) (2006) 121–142.

[31] E. Kharlamov, S. Brandt, M. Giese, E. Jiménez-Ruiz, Y. Kotidis, S. Lamparter, T. Mailis, C. Neuenstadt, Ö. L. Özçep, C. Pinkel, A. Soylu, C. Svingos, D. Zheleznyakov, I. Horrocks,

Y. E. Ioannidis, R. Möller, A. Waaler, Enabling semantic access to static and streaming distributed data with optique: demo, in: DEBS Demo, 2016, pp. 350–353.

[32] E. Kharlamov, S. Brandt, E. Jimenez-Ruiz, Y. Kotidis, S. Lamparter, T. Mailis, C. Neuenstadt, O. Özçep, C. Pinkel, C. Svingos, D. Zheleznyakov, I. Horrocks, Y. Ioannidis, R. Möller, Ontology-Based Integration of Streaming and Static Relational Data with Optique, SIGMOD demo.

[33] E. Kharlamov, S. Brandt, M. Giese, E. Jiménez-Ruiz, S. Lamparter, C. Neuenstadt, Ö. L. Özçep, C. Pinkel, A. Soylu, D. Zheleznyakov, M. Roshchin, S. Watson, I. Horrocks, Semantic Access to Siemens Streaming Data: the Optique Way, in: ISWC, 2015.

[34] E. Kharlamov, E. Jiménez-Ruiz, C. Pinkel, M. Rezk, M. G. Skjæveland, A. Soylu, G. Xiao, D. Zheleznyakov, M. Giese, I. Horrocks, A. Waaler, Optique: Ontology-Based Data Access Platform, in: ISWC P&D, 2015.

[35] E. Kharlamov, E. Jiménez-Ruiz, D. Zheleznyakov, D. Bilidas, M. Giese, P. Haase, I. Horrocks, H. Kllapi, M. Koubarakis, Ö. L. Özçep, M. Rodriguez-Muro, R. Rosati, M. Schmidt, R. Schlatte, A. Soylu, A. Waaler, Optique: Towards OBDA Systems for Industry, in: ESWC (Selected Papers), 2013, pp. 125–140.

[36] E. Kharlamov, M. Giese, E. Jiménez-Ruiz, M. G. Skjæveland, A. Soylu, D. Zheleznyakov, T. Bagosi, M. Console, P. Haase, I. Horrocks, S. Marciuska, C. Pinkel, M. Rodriguez-Muro, M. Ruzzi, V. Santarelli, D. F. Savo, K. Sengupta, M. Schmidt, E. Thorstensen, J. Trame, A. Waaler, Optique 1.0: Semantic Access to Big Data: The Case of Norwegian Petroleum Directorate FactPages, in: ISWC Posters & Demos, 2013.

[37] Özgür. Özçep, R. Möller, C. Neuenstadt, A Stream-Temporal Query Language for Ontology Based Data Access, in: Proceedings of the 37th German Conference on AI (KI 2014), 2014, pp. 183–194.

[38] D. Dell'Aglio, E. D. Valle, J. Calbimonte, Ó. Corcho, RSP-QL semantics: A unifying query model to explain heterogeneity of RDF stream processing systems, Int. J. Semantic Web Inf. Syst. 10 (4) (2014) 17–44.

[39] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham, et al., Storm@ twitter, in: Proceedings of the 2014 ACM SIGMOD international conference on Management of data, ACM, 2014, pp. 147–156.

[40] T. Akidau, R. Bradshaw, C. Chambers, S. Chernyak, R. J. Fernández-Moctezuma, R. Lax, S. McVeety, D. Mills, F. Perry, E. Schmidt, et al., The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing, Proceedings of the VLDB Endowment 8 (12) (2015) 1792–1803.

[41] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, R. Rosati, Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family, JAR 39 (3).

[42] C. Neuenstadt, R. Möller, Ö. L. Özçep, OBDA for temporal querying and streams, in: D. Nicklas, Ö. L. Özçep (Eds.), Proceedings of the 1st Workshop on High-Level Declarative Stream Processing co-located with the 38th German AI conference (KI 2015), Dresden, Germany, September 22, 2015., Vol. 1447 of CEUR Workshop Proceedings, CEUR-WS.org, 2015, pp. 70–75.
URL `http://ceur-ws.org/Vol-1447/paper6.pdf`

[43] C. Neuenstadt, R. Möller, Özgür. L. Özçep, OBDA for Temporal Querying and Streams with STARQL, in: D. Nicklas, Özgür. L. Özçep (Eds.), Proceedings of the 1st Workshop on High-Level Declarative Stream Processing co-located with the 38th German AI Conference (KI 2015), Dresden, Germany, September 22, 2015, HiDeSt'15, Vol. 1447 of CEUR Proceedings, 2015.

[44] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodriguez-Muro, R. Rosati, M. Ruzzi, D. F. Savo, The mastro system for ontology-based data access, Semantic Web 2 (1) (2011) 43–53.

[45] M. Owens, Embedding an sql database with sqlite, Linux Journal 2003 (110) (2003) 2.

[46] M. Owens, G. Allen, SQLite, Springer, 2010.

[47] J. L. Bentley, Multidimensional binary search trees used for associative searching, Communications of the ACM 18 (9) (1975) 509–517.

[48] P. Indyk, R. Motwani, Approximate nearest neighbors: towards removing the curse of dimensionality, in: Proceedings of the thirtieth annual ACM symposium on Theory of computing, ACM, 1998, pp. 604–613.

[49] A. Gionis, P. Indyk, R. Motwani, et al., Similarity search in high dimensions via hashing, in: VLDB, Vol. 99, 1999, pp. 518–529.

[50] K. Georgoulas, Y. Kotidis, Distributed similarity estimation using derived dimensions, VLDB J. 21 (1) (2012) 25–50.

[51] N. Giatrakos, Y. Kotidis, A. Deligiannakis, V. Vassalos, Y. Theodoridis, In-network approximate computation of outliers with quality guarantees, Information Systems 38 (8) (2013) 1285–1308.

[52] D. L. Phuoc, M. Dao-Tran, J. X. Parreira, M. Hauswirth, A Native and Adaptive Approach for Unified Processing of Linked Streams and Linked Data, in: ISWC, 2011, pp. 370–388.

[53] G. Mehdi, E. Kharlamov, O. Savkovic, G. Xiao, E. G. Kalaycı, S. Brandt, I. Horrocks, M. Roshchin, T. A. Runkler, Semantic Rule-Based Equipment Diagnostics, in: ISWC, 2017, pp. 314–333.

[54] E. Kharlamov, O. Savkovic, G. Xiao, R. Penaloza, G. Mehdi, I. Horrocks, M. Roshchin, Semantic Rules for Machine Diagnostics: Execution and Management, in: CIKM, 2017.

[55] O. Savkovic, D. Calvanese, Introducing Datatypes in DL-Lite, in: ECAI, 2012, pp. 720–725.

[56] A. Artale, V. Ryzhikov, R. Kontchakov, DL-Lite with Attributes and Datatypes, in: ECAI, 2012, pp. 61–66.

[57] F. Baader, S. Borgwardt, M. Lippmann, Query rewriting for dl-lite with n-ary concrete domains, in: IJCAI, 2017, pp. 786–792.

[58] F. Baader, U. Sattler, Description logics with aggregates and concrete domains, IS 28 (8) (2003) 979–1004.

[59] A. Bolles, M. Grawunder, J. Jacobi, Streaming sparql extending sparql to process data streams, in: Proceedings of the 5th European semantic web conference on The semantic web: research and applications, Springer-Verlag, 2008, pp. 448–462.

[60] D. F. Barbieri, D. Braga, S. Ceri, E. Della Valle, M. Grossniklaus, C-sparql: Sparql for continuous querying, in: Proceedings of the 18th international conference on World wide web, ACM, 2009, pp. 1061–1062.

[61] D. F. Barbieri, D. Braga, S. Ceri, M. Grossniklaus, An execution environment for c-sparql queries, in: Proceedings of the 13th International Conference on Extending Database Technology, ACM, 2010, pp. 441–452.

[62] J.-P. Calbimonte, H. Jeung, O. Corcho, K. Aberer, Enabling query technologies for the semantic sensor web, Int. J. Semant. Web Inf. Syst. 8 (1) (2012) 43–63.

[63] J.-P. Calbimonte, Ontology-based access to sensor data streams, dissertation, Universidad Politecninca de Madrid (2013).
URL `http://oa.upm.es/15320/1/JEAN_PAUL_CALBIMONTE.pdf`

[64] D. Anicic, P. Fodor, S. Rudolph, N. Stojanovic, Ep-sparql: a unified language for event processing and stream reasoning, in: WWW, 2011, pp. 635–644.

[65] J. Kietz, T. Scharrenbach, L. Fischer, A. Bernstein, K. Nguyen, Tef-sparql: The ddis query-language for time annotated event and fact triple-streams, Tech. rep., Technical report, University of Zurich, Department of Informatics (2013).

[66] D. F. Barbieri, D. Braga, S. Ceri, E. D. Valle, M. Grossniklaus, C-SPARQL: A Continuous Query Language for RDF Data Streams, Int. J. Sem. Computing 4 (1) (2010) 3–25.

[67] D. Le-Phuoc, M. Dao-Tran, M.-D. Pham, P. Boncz, T. Eiter, M. Fink, Linked Stream Data Processing Engines: Facts and Figures, in: ISWC, 2012, pp. 300–312.

[68] J.-P. Calbimonte, J. Mora, O. Corcho, Query rewriting in rdf

stream processing, in: Proceedings of the 13th International Conference on The Semantic Web. Latest Advances and New Domains - Volume 9678, Springer-Verlag New York, Inc., New York, NY, USA, 2016, pp. 486–502.

[69] N. Dindar, N. Tatbul, R. J. Miller, L. M. Haas, I. Botan, Modeling the execution semantics of stream processing engines with secret, The VLDB Journal 22 (4) (2013) 421–446.

[70] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, F. Reiss, M. A. Shah, TelegraphCQ: Continuous Dataflow Processing, in: SIGMOD, 2003, pp. 668–668.

[71] A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, I. Nishizawa, J. Rosenstein, J. Widom, STREAM: the stanford stream data manager, in: SIGMOD, 2003, p. 665.

[72] D. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, C. Erwin, E. Galvez, M. Hatoun, A. Maskey, A. Rasin, et al., Aurora: A Data Stream Management System, in: SIGMOD, 2003, pp. 666–666.

[73] A. Guttman, R-trees: A dynamic index structure for spatial searching, in: SIGMOD'84, Proceedings of Annual Meeting, Boston, Massachusetts, USA, June 18-21, 1984, 1984, pp. 47–57.

[74] S. Agrawal, S. Chaudhuri, V. R. Narasayya, Automated selection of materialized views and indexes in sql databases, in: VLDB, Vol. 2000, 2000, pp. 496–505.

[75] R. Avnur, J. M. Hellerstein, Eddies: Continuously adaptive query processing, ACM sigmod record 29 (2) (2000) 261–272.

[76] Y. Kotidis, N. Roussopoulos, DynaMat: A Dynamic View Management System for Data Warehouses, in: SIGMOD, 1999, pp. 371–382.

[77] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, H. Pirahesh, Data Cube: A Relational Aggregation Operator Generalizing Group-by, Cross-tab, and Sub-totals, Data mining and knowl. discovery 1 (1) (1997) 29–53.

[78] A. Wasay, X. Wei, N. Dayan, S. Idreos, Data canopy: Accelerating exploratory statistical analysis, in: Proceedings of the 2017 ACM International Conference on Management of Data, ACM, 2017, pp. 557–572.

[79] E. Jiménez-Ruiz, E. Kharlamov, D. Zheleznyakov, I. Horrocks, C. Pinkel, M. G. S. veland, E. Thorstensen, J. Mora, BootOX: Practical Mapping of RDBs to OWL 2, in: ISWC, 2015.

[80] C. Pinkel, C. Binnig, E. Jiménez-Ruiz, E. Kharlamov, W. May, A. Nikolov, A. S. Bastinos, M. G. Skjæveland, A. Solimando, M. Taheriyan, C. Heupel, I. Horrocks, RODI: Benchmarking relational-to-ontology mapping generation quality, Semantic Web 9 (1) (2018) 25–52.

[81] C. Pinkel, C. Binnig, E. Jiménez-Ruiz, E. Kharlamov, A. Nikolov, A. Schwarte, C. Heupel, T. Kraska, IncMap: A Journey towards Ontology-based Data Integration, in: BTW, 2017, pp. 145–164.

[82] E. Jiménez-Ruiz, E. Kharlamov, D. Zheleznyakov, I. Horrocks, C. Pinkel, M. G. Skjæveland, E. Thorstensen, J. Mora, BootOX: Bootstrapping OWL 2 Ontologies and R2RML Mappings from Relational Databases, in: ISWC Posters & Demonstrations Track, 2015.

[83] C. Pinkel, C. Binnig, E. Jiménez-Ruiz, W. May, D. Ritze, M. G. Skjæveland, A. Solimando, E. Kharlamov, RODI: A benchmark for automatic mapping generation in relational-to-ontology data integration, in: ESWC, 2015, pp. 21–37.

[84] C. Pinkel, C. Binnig, E. Kharlamov, P. Haase, Pay as you go matching of relational schemata to OWL ontologies with incmap, in: ISWC Posters & Demonstrations, 2013, pp. 225–228.

[85] V. T. Ho, D. Stepanova, M. H. Gad-Elrab, E. Kharlamov, G. Weikum, Rule Learning from Knowledge Graphs Guided by Embedding Models, in: ISWC, 2018, pp. 72–90.

[86] V. T. Ho, D. Stepanova, M. H. Gad-Elrab, E. Kharlamov, G. Weikum, Learning rules from incomplete kgs using embeddings, in: ISWC Posters & Demonstrations, 2018.

[87] A. Soylu, M. Giese, E. Jiménez-Ruiz, E. Kharlamov, D. Zheleznyakov, I. Horrocks, Ontology-based end-user visual query formulation: Why, what, who, how, and which?, Universal Access in the Information Society 16 (2) (2017) 435–467.

[88] A. Soylu, M. Giese, R. Schlatte, E. Jiménez-Ruiz, E. Kharlamov, Ö. L. Özçep, C. Neuenstadt, S. Brandt, Querying industrial stream-temporal data: An ontology-based visual approach, JAISE 9 (1) (2017) 77–95.

[89] A. Soylu, E. Kharlamov, D. Zheleznyakov, E. Jiménez-Ruiz, M. Giese, M. G. Skjæveland, D. Hovland, R. Schlatte, S. Brandt, H. Lie, I. Horrocks, Optiquevqs: A visual query system over ontologies for industry, Semantic Web 9 (5) (2018) 627–660.

[90] A. Soylu, E. Kharlamov, D. Zheleznyakov, E. Jiménez-Ruiz, M. Giese, I. Horrocks, Ontology-based visual query formulation: An industry experience, in: ISVC, 2015, pp. 842–854.

[91] A. Soylu, M. Giese, E. Jiménez-Ruiz, E. Kharlamov, D. Zheleznyakov, I. Horrocks, Why not simply google?, in: NordiCHI, 2014, pp. 1039–1042.

[92] E. Kharlamov, L. Giacomelli, E. Sherkhonov, B. C. Grau, E. V. Kostylev, I. Horrocks, Semfacet: Making hard faceted search easier, in: CIKM, 2017, pp. 2475–2478.

[93] E. Kharlamov, L. Giacomelli, E. Sherkhonov, B. C. Grau, E. V. Kostylev, I. Horrocks, Ranking, aggregation, and reachability in faceted search with semfacet, in: ISWC Posters & Demonstrations, 2017.

[94] E. Sherkhonov, B. C. Grau, E. Kharlamov, E. V. Kostylev, Semantic faceted search with aggregation and recursion, in: ISWC, 2017, pp. 594–610.

[95] M. Arenas, B. C. Grau, E. Kharlamov, S. Marciuska, D. Zheleznyakov, Faceted search over rdf-based knowledge graphs, J. Web Sem. 37-38 (2016) 55–74.

[96] B. C. Grau, E. Kharlamov, S. Marciuska, D. Zheleznyakov, M. Arenas, Semfacet: Faceted search over ontology enhanced knowledge graphs, in: ISWC Posters & Demonstrations, 2016.

[97] M. Arenas, B. C. Grau, E. Kharlamov, S. Marciuska, D. Zheleznyakov, Faceted search over ontology-enhanced RDF data, in: CIKM, 2014, pp. 939–948.

[98] M. Arenas, B. C. Grau, E. Kharlamov, S. Marciuska, D. Zheleznyakov, E. Jiménez-Ruiz, Semfacet: semantic faceted search over yago, in: WWW, 2014, pp. 123–126.

[99] M. Arenas, B. C. Grau, E. Kharlamov, S. Marciuska, D. Zheleznyakov, Towards semantic faceted search, in: WWW, 2014, pp. 219–220.

[100] B. C. Grau, E. Kharlamov, E. V. Kostylev, D. Zheleznyakov, Controlled query evaluation for datalog and OWL 2 profile ontologies, in: IJCAI, 2015, pp. 2883–2889.

[101] B. C. Grau, E. Kharlamov, E. V. Kostylev, D. Zheleznyakov, Controlled query evaluation over OWL 2 RL ontologies, in: ISWC, 2013, pp. 49–65.

[102] E. Kharlamov, O. Savkovic, G. Xiao, R. Peñaloza, G. Mehdi, M. Roshchin, I. Horrocks, Semantic rules for machine diagnostics: Execution and management, in: CIKM, 2017, pp. 2131–2134.

[103] G. Mehdi, E. Kharlamov, O. Savkovic, G. Xiao, E. G. Kalayci, S. Brandt, I. Horrocks, M. Roshchin, T. A. Runkler, Semdia: Semantic rule-based equipment diagnostics tool, in: CIKM, 2017, pp. 2507–2510.

[104] G. Mehdi, E. Kharlamov, O. Savkovic, G. Xiao, E. G. Kalayci, S. Brandt, I. Horrocks, M. Roshchin, T. A. Runkler, Semantic rule-based equipment diagnostic, in: ISWC Posters & Demonstrations, 2017.

[105] E. Kharlamov, O. Savkovic, M. Ringsquandl, G. Xiao, G. Mehdi, E. G. Kalayci, W. Nutt, M. Roshchin, I. Horrocks, T. A. Runkler, Diagnostics of trains with semantic diagnostics rules, in: ILP, 2018, pp. 54–71.

[106] J. Duggan, A. J. Elmore, M. Stonebraker, M. Balazinska, B. Howe, J. Kepner, S. Madden, D. Maier, T. Mattson, S. Zdonik, The bigdawg polystore system, ACM Sigmod Record 44 (2) (2015) 11–16.

Appendix A. Data Analysis Example

In Table A.3 we illustrate the shape of $\text{SQL}^{\oplus}$ queries that were used during our experimental evaluation. The query corresponding to the one in the table computes the Pearson correlation of a live stream with a varying number of archived streams. Each new stream record provides information related to a temperature sensor such as: (i) the time when the measurement was made: the `timestamp` attribute; (ii) the id of the sensor that took the measurement: the `sensor` attribute; (iii) the feature that was measured: the `feature` attribute; (iv) the value of the measurement: the `value` attribute. Archived streams also contain one additional attribute next to each of their records corresponding to the window id `Wid` of the measurement.

Table A.3: SQL$^{\oplus}$ Generated Query

```
/*
 * Compares the last window of live stream with all archived windowsand returns the number of archived
 * windows for which the pearson correlation between it and the streaming one are above 0.6
 */

ATTACH DATABASE '/home/optique/demo/tc255_8.db' AS tc255;


-- CREATE a static table with all appropriate windows
CREATE TEMP TABLE static_wids AS
SELECT wid
FROM tc255_8
GROUP BY widgroup BY wid

-- Get the stream for sensor TC260
CREATE TEMP VIEW stream AS WCACHE
SELECT *
FROM (newtimeslidingwindow timewindow:10 frequency:10 granularity:1 equivalence:floor
      SELECT cast(strftime('%s', timestamp) as long) as epoch, *
      FROM (file dialect:json 'http://optique-ubuntu-04:8989/uniondataset'
      where sensor = 'TC260');

-- Get the last window statistics
CREATE TEMP VIEW stream_wids AS WCACHE
SELECT wid, max(timestamp) AS window_time
FROM stream
GROUP BY wid;

-- We need to add the one window "next" to another. So we make a join between the wids and aboxes.
-- This stream tells for the current window with wich wids must compared.
CREATE TEMP VIEW matches AS WCACHE
SELECT stream_wids.wid AS stream_wid,
       stream_wids.window_time AS window_time,
       static_wids.wid AS static_wid
FROM stream_wids, static_wids;

-- Add the one window "next" to archived ones
CREATE TEMP VIEW final AS ORDERED
SELECT matches.stream_wid AS stream_wid,
       matches.window_time AS window_time,
       matches.static_wid AS static_wid,
       stream.value AS stream_value,
       tc255_8.value AS static_value
FROM matches, stream, tc255_8
where stream.wid = matches.stream_wid and
      tc255_8.wid = matches.static_wid and
      tc255_8.abox = stream.abox;


-- Take the final results
CREATE TEMP VIEW all_with_all AS ORDERED
SELECT *, latency((cast(strftime('%s', window_time) as integer))) as latency
FROM (
        SELECT stream_wid, window_time, static_wid, pearson(stream_value, static_value) AS pearson
        FROM final
        GROUP BY stream_wid, static_wid
        HAVING pearson > 0.6
);

-- Get statistics
SELECT window_time AS timestamp,
       'TC260' AS stream_sensor,
       count(*) AS sum_windows_matched,
       (SELECT count(*) FROM (SELECT * FROM tc255_8 GROUP BY wid)) AS sum_static_windows,
       latency
FROM all_with_all
GROUP BY stream_wid;
```