OBDA for Temporal Querying and Streams

Christian Neuenstadt, Ralf Möller, and Özgür L. Özçep

Institute of Information Systems (Ifis)
University of Lübeck
Lübeck, Germany
{moeller,neuenstadt,oezcep}@ifis.uni-luebeck.de

Abstract. Data changes worldwide in size and over time and when new data arrives rapidly from different sources, an easy access to dynamic data becomes a keyfactor. Therefore, temporalizing and streamifying ontology-based data access (OBDA) is a very important topic today, where the industry still relies on algebraic queries. We contribute to the practical efforts in this field by showing how a specific ontology-based stream querying language can be transformed with respect to mappings into standard SQL queries. For that purpose we choose the stream and temporal reasoning query language STARQL. STARQL is motivated by industrial usecases and evaluated in the European research project Optique. It offers access to temporal and streaming data as well for reactive diagnosis or continous monitoring.

1 Introduction

The following work contributes to research in the field of OBDA[1] on temporal data [4,3] and especially on streaming data. We show how we transform an ontology based stream data query language into equivalent relational algebra queries in SQL for timetagged data tables. Our approach extends previous work[9,11,10,8] and can be applied to the paradigm of streaming data as well as temporal data itself.

Sensor data is playing an important role in the IP7 EU project Optique. Siemens, one of the industrial stakeholders, provides huge amounts of sensor data to the project [7]. Thus, in this project the idea of processing streams with a sliding window (e.g. in [2]) has been extended by defining a finite sequence of consistent ABoxes for each window, which represent a sequence of time and extends previous ideas of adopting OBDA to streams [6,5,12], where the window content was treated as a single ABox. The idea of a sliding window is used for reactive diagnostics on temporal sensor data and accessed by within the project. In addition, a query language for streams has been developed, which is called STARQL[2]. STARQL is a SPARQL-like query languange with added window definition and a special clause for additional time constraints and querying a

[1] Ontology-based Data Access
[2] Streaming and Temporal ontology Access with a Reasoning-based Query Language

sequence of ABoxes. It has been designed for advanced data access to work equally for stream and temporal reasoning.

The idea of an *ABox sequencing strategy* makes the required steps, rewriting and unfolding for Ontology Based Data Access a challenging task. In the next section, we analyze the rules for STARQL transformations into algebraic queries like SQL and show that the rewriting and unfolding steps are indeed possible.

2 Transformation Rules for Streaming Data with STARQL

As mentioned in the last section, rewriting and unfolding of the ABox sequencing strategy is the most challenging task. STARQL uses a clause especially designed for temporal data access called Having Clause. We will concentrate explicitly on transforming the STARQL Having-Clause into an algebraic query language in this paper. An example for the use of a Having Clause can be seen in Listing 1. Here we see the definition of a stream Sout (Line 1), its output is defined as sensors of the type *MonInc*, which stands for *monotonically increasing* (Line 2). In line three the input stream with sensor values and the sliding window operator is defined, which is of size three seconds and slides with exactly one second for each window evaluation. In the next part the sequence of ABoxes for each window is defined. *StdSeq* stands for a sequence where each data set, which is represented by another timestamp, is placed in different ABoxes by an increasing order. The important part to evaluate the content of each window is shown in lines 5 to 7. For STARQL a new so called *Having Clause* has been explored. Its syntax is defined in first order logic using quantors for binding variables, conjunctions, disjunctions or negations of atoms, which can be graph triples or logical and arithmetic expressions. In our specific example a filter for monotonically increasing sensor data is used. It can be read as follows: for all timepoints i smaller than j it is true that if in i a sensor has a value x and in j the same sensor has a value y, then x must be smaller than or equal i.

An expression of the Having Clause from Listing 1 in first order logic is shown in formula 1. For an analysis of the STARQL syntax and semantics we refer to [9,10].

$$\forall i, j, x, y((R(i, s, x) \land S(j, s, y) \land i < j) \rightarrow (x \leq y)) \tag{1}$$

We will take this query as a running example in the further paper.

```
1  CREATE STREAM S_out AS
2  CONSTRUCT  GRAPH NOW { ?s rdf:type MonInc }
3  FROM STREAM S_Msmt [NOW-2s, NOW]->"1S"^^xsd:duration
4  SEQUENCE BY StdSeq AS seq
5  HAVING FORALL ?i,?j IN seq,?x,?y:
6   IF (GRAPH ?i { ?s :val ?x }  AND GRAPH ?j { ?s  :val ?y } AND ?i < ?j)
7          THEN ?x <= ?y
```

Listing 1: Example query in STARQL

Classical OBDA without querying time or time sequences has been researched for many years. In the non-temporal case the query language SPARQL is commonly used. Query languages like SPARQL (or in our temporal case STARQL) that are transformed to relational algebra must be domain independent, which can be achieved by a syntactical criterion requiring that all variables have to be range restricted (see [1] for a definition of domain independence). An example for a non restricted variable would be in the STARQL case a Having Clause of "$?y > 1$" . In this case, the variable $?y$ could be evaluated to numbers between one and infinity. We can range restrict the variable $?y$ by another atom like "$?y > 1$ AND GRAPH i { ?sens :hasVal ?y }". Here the variable $?y$ is not range restricted for the atom "$?y > 1$", but for the complete clause, because it is restricted to the concrete elements that actually exist in ABox i or if we stick to the example, all values that have been measured by a sensor. Thus, we see the Having Clause can be range restricted, although parts of it are not restricted itself. STARQL requires the Having Clause to be range restricted over all, which is guaranteed by its formulation rules. Thus, we can assume that the Having Clause is domain independent for our cases and can be indeed transformed into relational algebra. For reasons of space restriction, we refer for more specific information on STARQL domain independence to [10].

For our transformation of the STARQL Having Clause, we make two more assumptions. First, our database offers a view that represents time tagged data with a time sequence representation of the sliding window in two additional columns, where the schema consists of *WindowID*, *ABoxID*, *Timestamp* and [Datacolumns] for every temporal table. An example view for a sliding window with measured data is shown in Table 1. It includes measurements for a single sensor and describes two windows with *WindowID 1* and *2*, where the window width is of three minutes. Every single ABox in the sequence can be accessed by the ABoxID in the second column.

Table 1: Example for a sliding window view in the measurement scenario

WindowID	*ABoxID*	*Timestamp*	Sensor	Value
1	1	00:00	sens1	90
1	2	00:01	sens1	91
1	3	00:02	sens1	93
2	1	00:01	sens1	91
2	2	00:02	sens1	93
2	3	00:03	sens1	94

For the second assumption we assume additionally that the unfolding for a single graph triple "GRAPH i { subject predicate value}", where i respresents the index variable of ABoxes, is trivial and can be unfolded with mappings to $\delta_{aid \rightarrow i} \pi_{wid,aid,x_1,\ldots,x_n}(R)$, where *wid* represents the evaluated window, *aid* the ABox, $x_1, ..., x_n$ the specific data columns and R the described data view in sliding window format. A projection of the WindowID is necessary for temporal queries, where we explore several time windows in parallel. For streaming queries, where we evaluate the data of a single window per timepoint only, an index variable for comparing the ABoxes in a sequence would be enough.

Making these assumptions, the Having Clause of STARQL can be transformed by following the algorithm below for transforming from Safe Range Normal Form into Relational Algebra Normal Form (for detailed informations and proofs, see [1]).

First, we normalize the Safe Range Form of the Having Clause by five steps:

1. **Variable substitution:** All variable names are substituted until every variable is bound only once and no variable occurs free and bound.
2. **Remove universal quantifier:** We replace all $\forall \boldsymbol{x} \psi$ by $\neg \exists \boldsymbol{x} \neg \psi$
3. **Remove implications:** We replace all $\psi \rightarrow \xi$ by $\neg \psi \vee \xi$
4. **Push negations:** We push every negation down into the formula until every child of a negation is either an atom or an exists quantifier
5. **Flattening:** Flatten every *and*, which is the child of an *and*, do the same for *or* and *exists* quantifiers

Second, we normalize again for the relational algebraic normal form by three additional steps:

1. **Push into or:** We push every *or* upwards until there is no *or* as a child of an *and*.
2. **Push into quantifier:** if ψ is of the form $\psi_1 \wedge ... \wedge \psi_n \wedge \exists \boldsymbol{x} \xi$ and not all free variables of ξ are range restricted, **push** a subset of $\psi_1 \wedge ... \wedge \psi_n$ as conjunct into ξ until all free variables of ξ are range restricted.
3. **Push into negated quantifier:** if ψ is of the form $\psi_1 \wedge ... \wedge \psi_n \wedge \neg \exists \boldsymbol{x} \xi$ and not all free variables of ξ are range restricted, **copy** a subset of $\psi_1 \wedge ... \wedge \psi_n$ as conjunct into ξ until all free variables of ξ are range restricted.

$$\neg \exists i, j, x, y (R(i, s, x) \wedge S(j, s, y) \wedge i < j \wedge x > y) \tag{2}$$

After both normalization steps the Having Clause example from Listing 1 can now be written in RANF as shown in Formula 2.

For transforming a formula ψ from Relational Algebraic Normal Form (RANF) into an algebraic query E_ψ, we adopt the modified algorithm 1 with added window and ABox information from [1].

Unfolding of the RANF formula yields to the algebraic query in Formula 3.

$$\{\langle\rangle\} - \pi_{wid,aid,s}(\sigma_{x>y}(\sigma_{i<j}(\pi_{wid,aid,s,x}(R) \bowtie \pi_{wid,aid,s,y}(S)))) \tag{3}$$

In the next step, we replace the left part of the subtraction with the projection of the right side without making use of any specific selection.

$$\begin{aligned} &\pi_{wid,aid,s}(\pi_{wid,aid,s,x}(R) \bowtie \pi_{wid,aid,s,y}(S)) \\ - &\pi_{wid,aid,s}(\sigma_{x>y}(\sigma_{i<j}(\pi_{wid,aid,s,x}(R) \bowtie \pi_{wid,aid,s,y}(S)))) \end{aligned} \tag{4}$$

Algorithm 1 (Translation from RANF to Algebra)
Input: a formula ψ in modified RANF
Output: an algebra query E_ψ equivalent to ψ

```
$R(\boldsymbol{e}) \longrightarrow \delta_f(\pi_{wid,aid,x_1,\ldots,x_n}(\sigma_F(R)))$
$x = a \longrightarrow \{\langle x : a\rangle\}$
$\psi \wedge \xi \longrightarrow$ if $\xi$ is $x = x$, then $E_\psi$
    if $\xi$ is $x = y$, (with x, y distinct) then
        $\sigma_{x=y}(E_\psi)$, if $\{x, y\} \subseteq free(\psi)$
        $\sigma_{x=y}(E_\psi \bowtie \delta_{x\to y}E_\psi)$, if $x \in free(\psi)$ and $y \notin free(\psi)$
        $\sigma_{x=y}(E_\psi \bowtie \delta_{y\to x}E_\psi)$, if $y \in free(\psi)$ and $x \notin free(\psi)$
    if $\xi$ is $x \neq y$, then $\sigma_{x\neq y}(E_\psi)$
    if $\xi = \neg\xi'$, then
        $E_\psi - (E_\psi \bowtie E_{\xi'})$, if $free(\xi') \subset free(\psi)$
        $E_\psi - E_{\xi'}$, if $free(\xi') = free(\psi)$
        otherwise, $E_\psi \bowtie E_\xi$
$\neg\psi \longrightarrow \{\langle\rangle\} - E_\psi$ (if $\neg\psi$ does not have "and" as parent)
$\psi_1 \vee \ldots \vee \psi_n \longrightarrow E_\psi \cup \ldots \cup E_\psi$
$\exists x_1,\ldots,x_n \psi(x_1,\ldots,x_n,\ y_1,\ldots,y_m) \longrightarrow \pi_{wid,aid,y_1,\ldots,y_n}(E_\psi)$
```

Finally, we arrive at an algebraic query transformed from the STARQL Having Clause in formula 4, which can now directly be written in SQL.

This algorithm has been implemented in the stream query answering prototype of the EU Optique project. A detailed description and evaluation of several examples will be presented in future work.

3 Conclusion

In this paper we have shown how we transform the Having Clause of the stream and temporal query language STARQL in general and described a specific example. The transformation is done by three steps. First, we guarantee the domain independence of STARQL by the Safe Range Normal Form. Second, we build the Relational Algebraic Normal Form and third, we transform the formula into an algebraic query. This algorithm has been implemented in the EU project Optique and will be evaluated and optimized in future work.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley (1995)
2. Arasu, A., Babu, S., Widom, J.: The CQL continuous query language: semantic foundations and query execution. The VLDB Journal 15, 121–142 (2006)
3. Artale, A., Kontchakov, R., Wolter, F., Zakharyaschev, M.: Temporal description logic for ontology-based data access. In: Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence. pp. 711–717. IJCAI'13 (2013)
4. Borgwardt, S., Lippmann, M., Thost, V.: Temporal query answering in the description logic DL-Lite. In: Fontaine, P., Ringeissen, C., Schmidt, R.A. (eds.) Frontiers of Combining Systems. LNCS, vol. 8152, pp. 165–180. Springer (2013)
5. Calbimonte, J.P., Jeung, H., Corcho, O., Aberer, K.: Enabling query technologies for the semantic sensor web. Int. J. Semant. Web Inf. Syst. 8(1), 43–63 (Jan 2012)
6. Della Valle, E., Ceri, S., Barbieri, D., Braga, D., Campi, A.: A first step towards stream reasoning. In: Domingue, J., Fensel, D., Traverso, P. (eds.) Future Internet – FIS 2008, LNCS, vol. 5468, pp. 72–81. Springer Berlin / Heidelberg (2009)
7. Kharlamov, E., Solomakhina, N., Özcep, Ö.L., Zheleznyakov, D., Hubauer, T., Lamparter, S., Roshchin, M., Soylu, A.: How semantic technologies can enhance data access at siemens energy. In: Proceedings of the 13th International Semantic Web Conference (ISWC 2014) (2014)
8. Özçep, Ö. L.., Möller, R.: Ontology based data access on temporal and streaming data. In: Koubarakis, M., Stamou, G., Stoilos, G., Horrocks, I., Kolaitis, P., Lausen, G., Weikum, G. (eds.) Reasoning Web. Reasoning and the Web in the Big Data Era. Lecture Notes in Computer Science, vol. 8714. (2014)
9. Özçep, Ö.L., Möller, R., Neuenstadt, C., Zheleznyakov, D., Kharlamov, E.: Deliverable D5.1 – A semantics for temporal and stream-based query answering in an OBDA context. Deliverable FP7-318338, EU (October 2013)
10. Özçep, Özgür.L., Möller, R., Neuenstadt, C.: A stream-temporal query language for ontology based data access. In: KI 2014. LNCS, vol. 8736, pp. 183–194. Springer International Publishing Switzerland (2014)
11. Özgür L. Özçep, Möller, R., Neuenstadt, C.: A stream-temporal query language for ontology based data access. In: Bienvenu, M., Ortiz, M., Rosati, R., Simkus, M. (eds.) Proceedings of the 7th International Workshop on Description Logics (DL-2014) (2014)
12. Phuoc, D.L., Dao-Tran, M., Parreira, J.X., Hauswirth, M.: A native and adaptive approach for unified processing of linked streams and linked data. In: International Semantic Web Conference (1). pp. 370–388 (2011)