Web Semantics: Science, Services and Agents on the World Wide Web ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Web Semantics: Science, Services and Agents on the World Wide Web

journal homepage: www.elsevier.com/locate/websem

Semantic access to streaming and static data at Siemens

Evgeny Kharlamov [a,*], Theofilos Mailis [c], Gulnar Mehdi [b], Christian Neuenstadt [d], Özgür Özçep [d], Mikhail Roshchin [b], Nina Solomakhina [b], Ahmet Soylu [f], Christoforos Svingos [c], Sebastian Brandt [b], Martin Giese [e], Yannis Ioannidis [c], Steffen Lamparter [b], Ralf Möller [d], Yannis Kotidis [g], Arild Waaler [e]

[a] University of Oxford, Department of Computer Science, Wolfson Building, Parks Road, OX1 3QD, Oxford, UK
[b] Siemens Corporate Technology, Siemens AG, Otto-Hahn-Ring 6, 81739, Munich, Germany
[c] National and Kapodistrian University of Athens, Panepistimiopolis, Ilissia, 15784, Athens, Greece
[d] University of Luebeck, Ratzeburger Allee 160, 23562, Lübeck, Germany
[e] Department of Informatics, University of Oslo, Blindern, 0316, Oslo, Norway
[f] NTNU –Norwegian University of Science and Technology, Teknologiveien 22, 2815, Gjøvik, Norway
[g] Athens University of Economics and Business, 76 Patission Street, 10434, Athens, Greece

ARTICLE INFO

Article history:
Received 26 October 2016
Accepted 8 February 2017
Available online xxxx

Keywords:
Ontology based data access
Data integration
Streaming data
Static data
Predictive and reactive analytics
Optimisations
Siemens

ABSTRACT

We present a description and analysis of the data access challenge in Siemens Energy. We advocate Ontology Based Data Access (OBDA) as a suitable Semantic Web driven technology to address the challenge. We derive requirements for applying OBDA in Siemens, review existing OBDA systems and discuss their limitations with respect to the Siemens requirements. We then introduce the Optique platform as a suitable OBDA solution for Siemens. The platform is based on a number of novel techniques and components including a deployment module, BootOX for ontology and mapping bootstrapping, a query language STARQL that allows for a uniform querying of both streaming and static data, a highly optimised backend, ExaStream, for processing such data, and a query formulation interface, OptiqueVQS, that allows to formulate STARQL queries without prior knowledge of its formal syntax. Finally, we describe our installation and evaluation of the platform in Siemens.

1. Introduction

The growth of available information in enterprises requires new efficient methods for data access by domain experts whose ability to analyse data is at the core of making business decisions. Current centralised approaches, where an IT expert translates the requirements of domain experts into *Extract–Transform–Load* (ETL) processes to integrate the data and to apply predefined analytical reporting tools, are too heavy-weight and inflexible [1]. In order to support interactive data exploration, domain experts therefore want to access and analyse available data sources *directly*, without IT experts being involved.

This direct data access is particularly important for Siemens Energy[1] that runs several service centres for power plants. The main task of a service centre is remote *monitoring and diagnostics* of many thousand appliances, such as gas and steam turbines, generators, and compressors installed in plants. Monitoring and diagnostics are performed by service engineers and are typically conducted in four steps: (i) engineers receive a notification about a potential or detected issue with an appliance, (ii) they gather data relevant to the case, (iii) analyse the data, and finally (iv) report about ways to address the issue to the appliance owner. Currently, Step ii of the process is the bottleneck consuming up to 80% of the overall time needed by the engineer to accomplish the task. The main reason for this is the exploratory nature of data gathering that often cannot be accomplished by predefined ETL procedures and requires new such procedures which can only be done by highly qualified IT experts. Defining new ETL procedures over non-trivial data is time consuming and it slows down the diagnostic

* Corresponding author.
E-mail addresses: evgeny.kharlamov@cs.ox.ac.uk (E. Kharlamov), theofilos@image.ntua.gr (T. Mailis), gulnar.mehdi@siemens.com (G. Mehdi), neuenstadt@ifis.uni-luebeck.de (C. Neuenstadt), oezcep@ifis.uni-luebeck.de (Ö. Özçep), mikhail.roshchin@siemens.com (M. Roshchin), nina.solomakhina@siemens.com (N. Solomakhina), ahmet.soylu@ntnu.no (A. Soylu), c.svingos@di.uoa.gr (C. Svingos), sebastian-philipp.brandt@siemens.com (S. Brandt), martingi@ifi.uio.no (M. Giese), yannis@di.uoa.gr (Y. Ioannidis), steffen.lamparter@siemens.com (S. Lamparter), moeller@ifis.uni-luebeck.de (R. Möller), kotidis@aueb.gr (Y. Kotidis), arild@ifi.uio.no (A. Waaler).

[1] http://www.energy.siemens.com/.

http://dx.doi.org/10.1016/j.websem.2017.02.001

process for multiple reasons including the complexity of this task, the overload of IT experts, and even miscommunication between them and the engineers—they simply speak different languages and think in different perspectives.

Enabling direct data access for engineers in Siemens is a challenging task, primarily due to the Big Data dimensions as well as the conceptual mismatch between the language and structures that the engineers use to describe the data, and the way the data is actually expressed and structured in databases. The data accessible from Siemens service centres naturally reflects the variety, volume, and velocity dimensions of Big Data: it is stored in several thousand databases possibly under many different schemata; its size is in the order of hundreds of terabytes; and it currently grows at an average rate of 30 GB per day. Regarding the conceptual mismatch, it occurs because industrial schemata are often integrated from independently evolving databases adapted by different organisational units over years. The result is usually shaped by the IT aspect of the integration effort rather than the needs of the domain experts ultimately using the schema. Only IT experts fully understand this evolving structure of databases and thus currently only they can write queries over these databases in order to extract information relevant for engineers.

Ontology Based Data Access (OBDA) [2] has been recently proposed as a means to enhance end-user direct data access. The key idea behind OBDA is to use *ontologies*, i.e., semantically rich conceptual domain models, to mediate between users and data. Ontologies describe the domain of interest on a higher level of abstraction and in terms that are clear for domain experts.[2] In OBDA users formulate their information needs as queries using terms defined in the ontology and ontological queries are then *automatically translated* into SQL or some other database query languages and *executed* over the data, without an IT expert's intervention. To this end a set of *mappings* is maintained that describe the relationship between the ontological vocabulary and the schema of the data. Note that OBDA follows the classical data integration paradigm that requires the creation of a common 'global' schema that consolidates 'local' schemata of the integrated data sources and mappings that define how the local and global schemata are related [1].

The main benefit of OBDA is that the combination of ontologies and mappings allows 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. This allows us to formulate each Siemens diagnostic task via only one ontological query instead of a collection of hundreds data queries that today have to be written or configured by IT specialists. Note that this collection of queries does not disappear: the automatic translation by an OBDA system will compute it from the high-level ontological query. Another important benefit of OBDA is *modularity* and *compositionality* of its assets: each mapping relates one ontological term to the data, which allows the mappings to be constructed independently and on demand; and the same ontological term can be used in different queries, so defining mappings for even a few terms enables the evaluation of many different ontological queries.

Existing OBDA techniques and systems are tailored towards queries over static relational data. At the same time, monitoring and diagnostic routines at Siemens require *hybrid queries* that combine live data streams, historical time-stamped information, and static relational data. In order to meet Siemens requirements, we had to extend the traditional OBDA to handle hybrid queries and we shall refer to our approach as *Ontology-Based Stream-Static Data Integration* (OBSSDI).

[2] Ontologies have become a common and successful mechanism to describe application domains in, e.g., biology, medicine, and the (Semantic) Web [3]. This success is partially due to a number of available formal languages for describing ontologies, including the Web Ontology Language (OWL) standardised by W3C [4].

The first contribution of this work that required a huge effort is:

(i) Assessing Siemens data access needs, and consolidation of corresponding system requirements.

On the technical side our two main contributions are:

(ii) A query language STARQL that natively supports OBSSDI hybrid queries and
(iii) A highly efficient and scalable OBSSDI backend ExaStream to process such queries.

STARQL embeds (a fragment of) SPARQL and has a dedicated safe temporal first-order logic which allows to formulate many of the relevant monitoring patterns that arise in industrial use cases as that of Siemens. In contrast to most of the RDF stream approaches that extend SPARQL with stream operators, STARQL does not rely on a simple snap-shot semantics – which has its limits when handling functionality constraints – but instead provides a timestamp-preserving window semantics and on top of it a useful sequencing abstraction which allows for a flexible specification of window-internal groupings of timestamped RDF tuples. Despite this additional layer, STARQL queries are feasible (at least in the case of standard sequencing, see Section 4.1) because, within the transformation process to queries in the language of the backend system ExaStream, the abstraction layer can be eliminated without causing significant blowup.

ExaStream is a novel distributed Data Stream Management System that meets the data processing requirements of Siemens. ExaStream provides low latency answers to queries on high-velocity live streams and high-volume static data sources. It is built as a streaming extension to the SQLite database engine. As a result, it takes advantage of existing database optimisations that are blended with several novel optimisation techniques for efficiently processing analytical queries on streaming and static information. ExaStream has several important features such as: the ability to run in a distributed environment that can scale up in order to meet user demands; a declarative language, extending the SQL syntax for querying live streams and relations; native support of *user defined functions* with arbitrary user code for executing complex analytical workflows; and native support for streaming and static data integration.

Next important contributions of this work are

(iv) a system OptiqueVQS that allows end-users to construct STARQL queries without any prior knowledge of semantic technologies and the formal syntax of the language,
(v) a full-fledged implementation of our techniques in the Optique platform and
(vi) both end-user and performance evaluation of the platform at Siemens.

The paper is organised as follows. In Section 2, we analyse reactive and predictive diagnostics at Siemens and derive six Siemens direct data access requirements. In Section 3, we introduce classical OBDA, show that it conceptually satisfies the Siemens requirements, while the existing OBDA systems are not mature enough to fulfil the aforementioned six requirements and thus cannot be used in Siemens. In Section 4, we introduce our OBSSDI components: syntax and semantics of STARQL queries; how these queries can be transformed into SQL-like data queries of our backend ExaStream; and how ExaStream processes data queries. In Section 5, we introduce the Optique platform developed as a part of a large European project [5–9]. Optique supports STARQL queries, has ExaStream as one of its backends, offers a native SPARQL query builder OptiqueVQS, and offers a number of dashboards that we developed for turbine diagnostics. In Section 6, we present our deployment of the Optique platform over Siemens data and a user

Please cite this article in press as: E. Kharlamov, et al., Semantic access to streaming and static data at Siemens, Web Semantics: Science, Services and Agents on the World Wide Web (2017), http://dx.doi.org/10.1016/j.websem.2017.02.001.

evaluation. In Section 7, we present lessons learned, conclusions, and discuss future work.

Delta from previous publications. This paper extends our previous publications in several important ways. First, based on our evaluation experience with Siemens, we realised that the stream-only processing support that we initially developed was not sufficient, and thus we extended our use-case analysis of [10] by introducing Requirement 4 (see Section 3.1) about Stream-Static Data Processing; and 5 about a Data Stream Management System; moreover, in this submission we give more use-case related explanations and examples. Second, the section about STARQL extends previously presented material [10,11] with an in-depth comparison with existing approaches and systems, as well as with more details and explanations of the language. Third, most of the ExaStream techniques and the evaluation over the Siemens data are presented in this paper for the first time and were not reported in previous papers [12,13]. Fourth, the OptiqueVQS section is significantly extended compared to our earlier paper [10] since the system became much more mature over time. Finally, the Optique implementation has become much more mature and in particular the dashboards that we present in this paper have not been published before.

2. Siemens monitoring and diagnostic service

Siemens produces a variety of rotating appliances, including gas and steam turbines, generators, and compressors. These appliances are complex machines and typically used in different critical processes including power generation where each hour of downtime may cost thousands of euros. Thus, these appliances should be under constant monitoring that requires an in-depth knowledge of their components and setup. Siemens provides such monitoring via service centres and operates over fifty such centres worldwide, where each centre is responsible for several thousand appliances. Typical monitoring tasks of a service centre include: (i) *reactive and preventive diagnostics* of turbines which is about offline data analysis applied after a malfunction or an abnormal behaviour such as vibration, temperature or pressure increase, unexpected events, or even unexpected shutdowns, of a unit is detected; (ii) *predictive analysis* of turbine conditions which is about real-time data analysis of data streams received from appliances. We now discuss these monitoring tasks in detail and present requirements to enhance them.

2.1. Reactive and preventive diagnostics

Reactive diagnostics is usually applied after a malfunction of a unit has occurred, e.g., the abnormal shutdown of a turbine. Complementarily, the preventive diagnostic task is performed before a malfunction of a unit, when its abnormal behaviour is detected, e.g., high vibration or temperature increase. Diagnostic tasks are triggered either when a customer sends a service ticket claiming assistance or an automated diagnostic system creates such a ticket. Fig. 1 depicts a general process triggered when a service ticket arrives. We now discuss each step of the process in detail.

Arrival of a service ticket. A service ticket typically contains information on when a problem occurred and its frequency. In some cases the ticket isolates the location of the problem in the appliance and its cause, but often it has no or few details.

Example 1. An example of a reactive monitoring request from a customer is:

> *Figure out why the turbine failed to start during the last five hours, with the goal of checking that there will be no fault of the turbine.* ■
>
> A typical preventive monitoring request could be
>
> *Will there be a failure of the turbine after the observed temperature increase?* ■

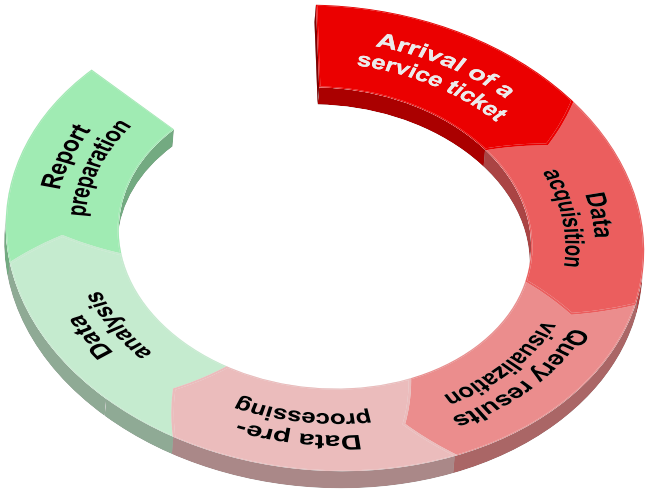

Fig. 1. High-level view on the turbine service process.

Data acquisition. Service engineers gather relevant data by querying databases that are updated every hour, or on demand, and contain sensor and event data. In order to support data gathering, Siemens equips service centres with more than 4000 predefined queries and query patterns of different complexity. Engineers use the queries by setting parameters such as time periods, names of events or sensors, sensor types, etc.

Example 2. Based on the service ticket of Example 1, the engineer formulates the following information need and has to find appropriate queries to cover it:

> *Return the most frequent start failure and warning messages of the gas turbine T01 during the last week. Moreover, find analogous cases of failures for turbines of the same type as T01 in the last three months.* ■

Query result visualisation. Sensor data is visualised with the use of standard diagrams, and event messages are presented as a list, i.e., as an Excel spreadsheet, with timestamps and additional attributes.

Data preprocessing. The queried data is preprocessed using generic procedures such as sensor check (i.e., whether sensor data quality is appropriate), threshold and trend analysis. Independent from the concrete ticket, these preprocessing steps are done manually, e.g., over the visualised Excel spreadsheets, or using specialised analytic tools.

Data analysis. The engineer uses sophisticated diagnostic models and tools for complex analysis, e.g., Principal Component Analysis or other statistical methods, to detect and isolate the given problem based on the preprocessed data. Typically, analytical tasks are executed individually for each ticket. The gathering and analysis steps are often carried out iteratively, i.e., the results from one iteration are used to pose additional queries.

Report preparation. This process terminates when an explanation for the problem in the service ticket is established. In this case the engineer provides the customer with a report aggregating the result of the analysis and describing possible further actions.

2.2. Predictive analysis

In predictive analysis, in contrast to the diagnostic process described above, appliances are continuously monitored, i.e., without prior service tickets, using online processing of the incoming sensor data. The other process steps of predictive analysis are

Please cite this article in press as: E. Kharlamov, et al., Semantic access to streaming and static data at Siemens, Web Semantics: Science, Services and Agents on the World Wide Web (2017), http://dx.doi.org/10.1016/j.websem.2017.02.001.

similar to the ones described in the previous section, but have to be applied online to streaming data with minimal user intervention. The purpose here is to analyse the current condition of an appliance by combining operating information, system data, specifications of concrete product lines, and temporal phases of operating regimes. This information allows to predict whether some parts of an appliance should be repaired soon, assess risks related to the use of these parts, and adjust maintenance intervals for each part by automatically integrating this information into service scheduling, thus, minimising maintenance cost.

Example 3. For predictive analysis of turbines, the diagnostic engineer may want to be automatically notified when a turbine shows repetitive start failures combined with increased vibration values during its operating time. This can be formulated as follows:

Notify me if a turbine that had more than three start failures in the last two weeks additionally shows abnormal vibration values in operative phases. ■

2.3. Siemens requirements

The main bottleneck for diagnostics is the data gathering part, which takes up to 80% of the overall diagnostic time. The main reason is that finding the *right* data for analytics is very hard due to limitations of predefined queries, complexity of data, complexity of query formulation, and limitation to explicitly stated information. In Fig. 2 we schematically depict the complex process of data access that requires to determine the right DB location, then the right schemata, and the corresponding data collectors and controllers deployed in turbines. Moreover, often diagnostic tasks involve up to dozens of turbines and thus this process should be done for each of them. Based on these observations we now derive concrete requirements that a system for diagnostic processing should fulfil.

R1: Integrated Data Access. Siemens data over which the queries are formulated naturally reflects the variety, volume, and velocity dimensions of Big Data. The data is stored in so-called data centres, each responsible for several thousand appliances such as turbines, where a typical turbine has about 2000 sensors constantly producing measurements. This data can be roughly grouped into three categories: (i) sensor and event data from appliances; (ii) analytical data obtained as results of monitoring tasks conducted by service centres for the last several years; and (iii) miscellaneous data, typically stored in XML, containing technical description of appliances, types of configurations for appliances, indicates in which databases information from sensors is stored, history of weather forecasts, etc. All in all the data is stored in several thousand databases having a variety of different schemata. The size of the data is in the order of hundreds of terabytes, e.g., there are about 15 GB of data associated to a single turbine, and they currently grow with the average rate of 30 GB per day. At the moment there is no unified access point to the Siemens data and it is required.

R2: Flexible Definition of Queries. Existing predefined queries in the Siemens query catalogue, about 4000 queries, are often not sufficient to cover information needs as they are often either too general, thus yielding an overload of irrelevant information, or too specific, thus not providing enough relevant data. For gathering relevant data, service engineers often have to use several queries and combine their results. When this is not sufficient, existing queries have to be modified or new queries should be created. To this end the engineer contacts an IT expert and this leads to a complex and time-consuming interaction that takes up to weeks. The reason why it takes so long is miscommunication, high workload of IT personnel, complexity of query formulation, and long query execution times. In average up to 35 queries require modification every month, and up to 10% of queries are changed throughout a year. Moreover, several new queries are developed monthly. Therefore, flexible modification and definition of queries is one of the strong requirements for the improvement of the diagnostic process.

Example 4. Continuing with the query in Example 2, in order to answer it first about the turbine T01, the engineer spent two days and found three queries Q_1, Q_2, and Q_3 which, taken together, partially answer his information need. All three queries ask about start failures of some, but not all components. Moreover, Q_1 and Q_2 ask about T01, while Q_3 asks about a cluster of turbines, e.g., T01 and T02 located in the same factory. By tuning parameters of these queries, the engineer manages to retrieve 70% of relevant answers with Q_1 (since it asks about some parts of the turbine only), and an overlapping 40% with Q_2, giving him a coverage of only 90% of relevant answers in total. The cluster-based query Q_3 covers only 50% of relevant answers for T01 and additionally returns irrelevant answers about T02. Even though the union of answers from Q_1, Q_2, and Q_3 gives all the relevant answers, the engineer has to request the IT staff to produce extra queries: He either needs a query that retrieves the remaining 10% of answers for the combination of Q_1 and Q_2, or a query to filter out the irrelevant answers from the combination of all three queries.

Even in this relatively simple scenario, the engineer will typically not have enough information on the many DBs involved, despite the fact that he has a conceptual understanding of the missing 10% of answers and also understands what to filter out in the combination of three queries. By contrast, IT support has a clear picture of all DBs available, but lacks domain knowledge for expressing the engineer's information needs. This leads to an iterative process and query development times in the region of days. ■

R3: Utilising implicit information. In databases it is typically assumed that only explicit data matters, i.e., the data which is stored in the system. From a formal perspective, the so-called *closed-world* semantics is adopted, meaning that exactly the information stated is true, and anything not stated is false. While this perspective may be valid in the context of controlled systems, completeness of data is hardly ever the case in practical industry applications such as the ones in Siemens. Here, the fact that we do not have a measurement tuple for a certain time point does not mean there is no measurement. This could be reflected by the so-called *open-world* semantics, that allow to derive *implicit information* from the data stated explicitly, typically using some forms of background knowledge. This implicit information logically follows from what is stated explicitly, and its use can greatly increase the practical benefit of a diagnostic system.

Example 5. In our example we have that symptoms of start failures are already recorded in the DBs, while there is no explicit indication that there was a start failure. Hence, the engineer has to query not only for start failures, but also for relevant symptoms. There are several hundred symptoms of start failures. For example, symptoms of start failures such as low temperature and pressure are explicitly recorded, and they implicitly indicate that a start failure will occur within the next two minutes. ■

R4: Ontology Based Stream-Static Data Processing. Predictive analysis requires the use of both static information from the past and streaming information on the current status of appliances. Access to historical data allows to detect, for instance, seasonal patterns. Continuous monitoring of the streaming data provides prognosis for key performance indicators and countermeasures before a system shutdown occurs. Currently, service engineers do not have direct access to streaming data. However, engineers often need to access event and sensor data from several appliances, and stream

Please cite this article in press as: E. Kharlamov, et al., Semantic access to streaming and static data at Siemens, Web Semantics: Science, Services and Agents on the World Wide Web (2017), http://dx.doi.org/10.1016/j.websem.2017.02.001.

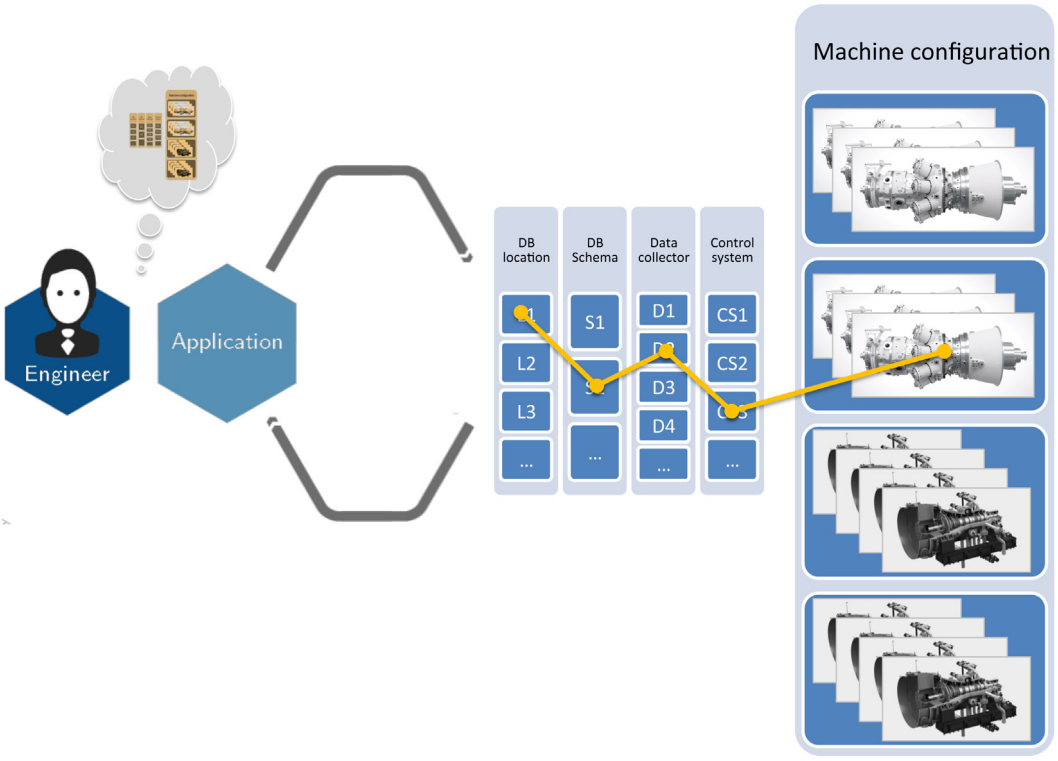

Fig. 2. Current approach to diagnostics.

processing for each related turbine. One of the requirements for the predictive analysis is the possibility to integrate sensor and event streaming data from several turbines and diagnostic centres and provide the use of continuous queries on data streams.

Example 6. In our example the data relevant for start failures are live data streams produced by sensors installed in turbines, relevant sensor data from the past, and turbine structure. ■

R5: Data Stream Management System. All the previously mentioned requirements should be accompanied by a backend system that supports low latency answering of queries on high-velocity live streams and high-volume static data sources. The aforementioned system should poses the following features: (i) *scalability:* the ability to run in a distributed environment and its capacity to easily add and remove queries without disrupting existing query execution; (ii) *declarative semantics:* the backend system should provide a declarative language, extending the SQL syntax and semantics for querying live streams and relations; (iii) *user defined functions:* the backend system should natively support *user defined functions* with arbitrary user code; (iv) *stream and static data integration:* based on its architecture and implementation, the backend system should natively support streaming and static data integration.

Summing up on the requirements above, Siemens needs a solution that: naturally integrates streaming and archived data as well as static information; allows for flexible query definition; exploits both explicit and implicit data; and allows for effective data processing.

3. Ontology based data access

Ontology Based Data Access (OBDA) is a prominent approach for end-user oriented access to databases. OBDA relies on Semantic Web technologies and it has been heavily studied by the Semantic Web community [2].

The main idea behind OBDA is to provide a user with access to the data via an *ontology* that is specific to the user's domain. The ontology can be written in some ontology language, e.g., in the Web Ontology Language OWL 2 standardised by W3C. This ontology hides from the user technical details about the database schemata while it exhibits to the user a domain specific vocabulary of classes and properties i.e., unary and binary predicates, that the user is familiar with. This vocabulary is related to the database schemata via *mappings*, which are declarative specifications, similar to view definitions in databases. There are several mapping languages available, e.g., R2RML standardised by W3C. Fig. 3 presents a general conceptual diagram illustrating OBDA: its main components and the workflow of query answering in OBDA systems.

The user formulates queries over ontologies in terms of the classes and properties. The standard query language for ontologies is SPARQL 1.1 standardised by W3C. An ontological query Q_1 is evaluated over databases in three steps. First, Q_1 is expanded with relevant information from the ontology in order to retrieve both explicit and implicit answers from the databases. This is accomplished by *query rewriting*, which takes the query Q_1 and the ontology, and produces the query Q_2. Note that Q_2 is logically equivalent to Q_1 with respect to the ontology while it "absorbs" a fragment of the ontology necessary for retrieving all answers relevant to Q_1. We refer the reader to, e.g., [14], for details on query rewriting techniques. OBDA systems typically do rewriting of so-called *conjunctive queries* with ontologies that fall in the OWL 2 QL profile of OWL 2. This profile is specifically tailored for data access and allows for efficient query processing [14]. At the second step, the query Q_2 is translated using mappings into a query Q_3 over the database schemata, e.g., into SQL when the data is relational. This step is referred to as *unfolding*. Finally, Q_3 is executed over the data by a DBMS and the answers are returned to the user.

We now illustrate OBDA on the following example which is based on the ontology and mappings that we developed for the Siemens use case. Note that, for the sake of clarity, the example is based on simplified versions of these ontology and mappings.

Please cite this article in press as: E. Kharlamov, et al., Semantic access to streaming and static data at Siemens, Web Semantics: Science, Services and Agents on the World Wide Web (2017), http://dx.doi.org/10.1016/j.websem.2017.02.001.

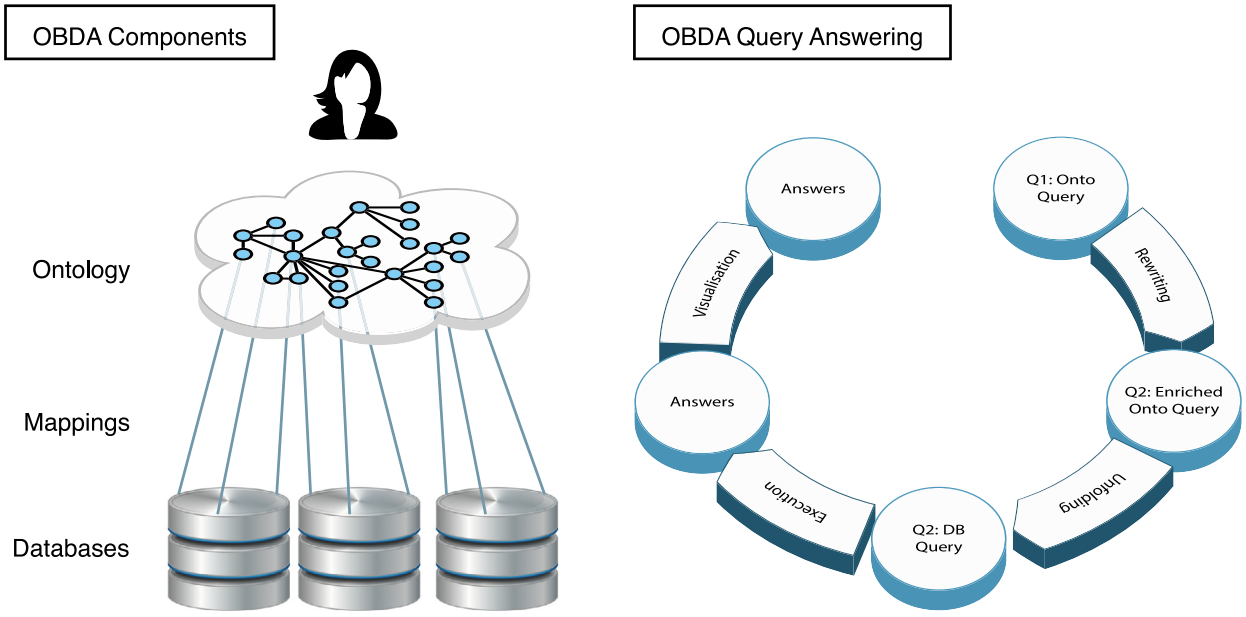

Fig. 3. OBDA: components and a general idea of query processing.

Example 7. The ontology in Fig. 4 says that turbines can be either gas or diesel. A gas turbine may have the following parts: (i) a control system that in turn has a control unit of types ART or ART2, (ii) inner turbine, (iii) lube-oil system that may have several sensors for measuring pressure, and (iv) gearbox. Moreover, a gas turbine can be located in a place such as a desert, or a frost, etc. For the sake of simplicity, we assume that diesel turbines are modelled in the same way as the gas ones.

The query in Fig. 4 asks: "*Return the pressure measured by sensors of lube oil systems in turbines*." This is an ontological query which corresponds to Q_1 in Fig. 3. This query can be written in SPARQL as follows:

```
SELECT ?Measurement
WHERE {?X rdf:type siemens:Turbine.
       ?X siemens:hasPart ?Y.
       ?Y rdf:type siemens:LubeOilSystem.
       ?Y siemens:hasSensor ?Z.
       ?Z rdf:type siemens:Sensor.
       ?Z siemens:hasPressure ?Measurement.}
```

Query rewriting techniques applied to this query and the turbine ontology produce two more queries that have the same structure, as Q_1, but the first query has Gas Turbine and the second one has Diesel Turbine in the place of Turbine. The query Q_2 is the union of Q_1 with these two queries. In terms of SPARQL, Q_2 can be obtained from Q_1 by substituting the first triple of its WHERE clause with the following expression:

```
      { ?X rdf:type siemens:Turbine } UNION
   { ?X rdf:type siemens:GasTurbine } UNION
{ ?X rdf:type siemens:DieselTurbine }
```

There are two mappings in Fig. 4. The left one says how to "populate" the property hasPressure: one has to project tuples of the table Measurement, where the value of the attribute Type is "pressure". The projection on the attribute SensorID gives the subject and on the attribute Value1 gives the object of hasPressure. The right mapping says how to "populate" the class LubeOilSystem: one has to project tuples of the table System where the Purpose is "Lubricant Delivery" on the SystemID attribute. These mappings can be used to unfold the SPARQL query Q_2 into an SQL query Q_3. We do not give Q_3 here due to space limit since this would require to introduce six more mappings. ■

3.1. How OBDA can help in improving data access in siemens

In Section 2.3, we presented five Siemens data access requirements. We will discuss now how OBDA naturally addresses all of them and thus we believe that OBDA has the potential of improving data access in Siemens.

OBDA naturally addresses Requirement R1 on integrated data access since one ontology can mediate the user and several databases with different formats via mappings. Regarding Requirement R2 on flexible definition of queries, since ontologies describe the domain of end users, formulation of queries over ontologies is conceptually much easier than over databases. Thus, by relying on intuitive query formulation tools, users can combine existing queries and write new queries without any knowledge of the schemata of multiple databases residing behind the ontology. Regarding Requirement R3 on utilising implicit information, OBDA naturally does so via logical reasoning during the query rewriting process. Regarding Requirement R4 on stream-static data processing, OBDA does not impose any restriction on the type of data to be integrated. Finally, Regarding Requirement R5 of a data stream management system, OBDA separates reasoning over queries and ontologies (that takes place before unfolding) from data processing (that takes place after the unfolding), thus, it opens doors for the development of highly optimised backends.

Thus, what we need for Siemens is an OBDA system that (i) supports distributed data processing, (ii) provides a flexible intuitive query formulation and visualisation support, (iii) relies on logical reasoning to obtain both explicit and implicit answers, (iv) accommodates static, streaming, and historic data steams, and (v) offers a highly efficient backend. As we see next, no such OBDA system exists.

3.2. Existing OBDA systems and their limitations

We now show that, despite the recent advances in OBDA systems, they are currently not mature enough to be applied off-the-shelf in Siemens and both theoretical and practical developments are required. There are several academic and industrial systems for OBDA or that are very similar to OBDA in spirit. Mastro [15], morph-RDB [16], and Ontop [17] support ontology reasoning and

Please cite this article in press as: E. Kharlamov, et al., Semantic access to streaming and static data at Siemens, Web Semantics: Science, Services and Agents on the World Wide Web (2017), http://dx.doi.org/10.1016/j.websem.2017.02.001.

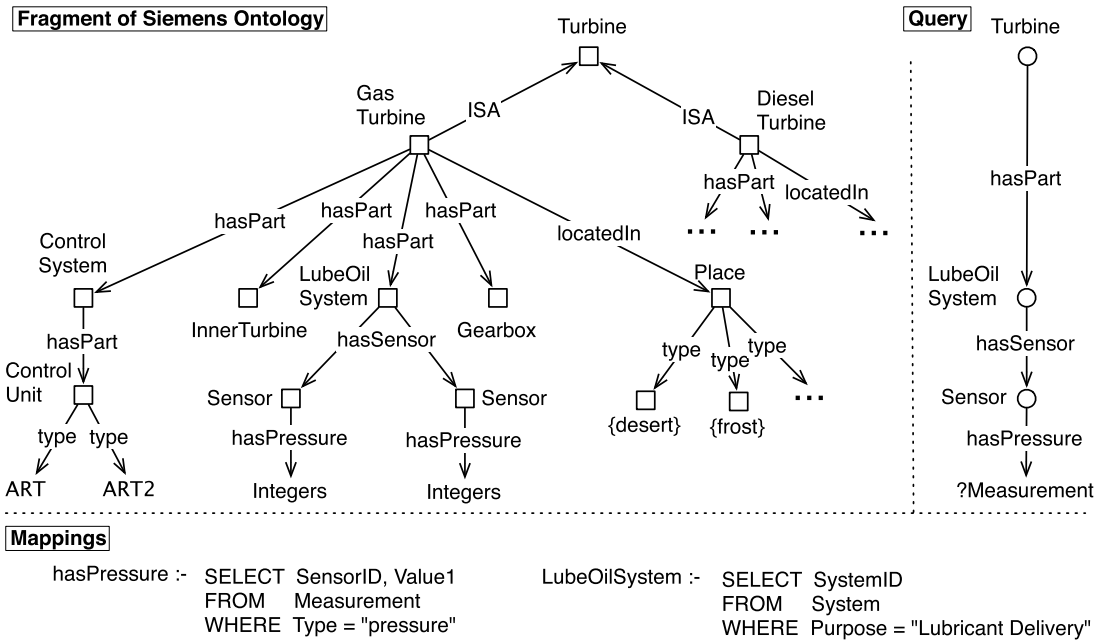

Fig. 4. Simplified Siemens ontology and mappings, example query.

thus address Requirement R3, while D2RQ [18], OntoQF [19], Virtuoso,[3] Spyder,[4] and Ultrawrap [20] do not support reasoning and thus fail Requirement R3. Moreover, all these systems fail Requirement R2: Ultrawrap, Ontop, Mastro, and morph-RDB lack user-oriented query formulation interfaces and query visualisation, since they only provide SPARQL end-points and predefined queries; while OntoQF considers ontology queries as OWL statements and has no visual query formulation support. Existing OBDA systems either assume that data is in (static) relational DBs, e.g [15,17], or streaming, e.g., [21,22] but not of both kinds. Thus, to the best of our knowledge, there is none that fulfils Requirement R4. Finally, to the best of our knowledge, no OBDA system supports backend optimisation as required by R5. For example, Ontop supports query optimisation during rewriting, but it does not compute efficient plans for unfolded queries.

We conclude that no OBDA exists that addresses Siemens requirements and can be used as it is. In order to meet Siemens requirements, we had to extend the traditional OBDA and we refer to our approach as *Ontology-Based Stream-Static Data Integration* (OBSSDI). In the following section we present two main components of our proposal: a query language and processing techniques.

4. Our OBSSDI components

In this section we present our OBSSDI Components. First, we present the language STARQL: its syntax, semantics, and how we turn STARQL ontological queries into data queries. Then, we present our backend ExaStream tailored towards answering STARQL queries.

4.1. STARQL *Query Language*

Recent efforts on temporalised [23,24] and streamified [25–32] OBDA systems provide first steps towards handling temporal and streaming data in industrial applications. However, none of these approaches satisfies the requirements of the Siemens use case: either there is no implemented and/or optimised engine or the engine is still not fully developed (see the benchmark tests in [33]). Below we give a comparative overview over recent RDF engines including our stream-temporal submodule of the Optique platform.

Streaming and Temporal ontology Access with a Reasoning-based Query Language (STARQL) [11,34–37] offers a query framework allowing to deal with streams of timestamped RDF triples on the background of mappings and an ontology. The development of STARQL was inspired by the Siemens use-case requirements. The STARQL query language framework and the prototype streaming engine enjoy the following features:

Expressivity. STARQL allows to express typical mathematical, statistical, and event pattern features needed in real-time monitoring scenarios. In spite of its expressivity, answering STARQL queries is still efficient since these can be transformed into relational stream queries.

Neat semantics. STARQL comes with formal syntax and semantics. The latter one uses certain-answer semantics [2] and on top of that, first-order logic semantics as in model checking, thereby combining open and closed-world reasoning. A snapshot semantics for window operators [38] is extended with a sequencing semantics that can handle integrity constraints such as functionality assertions.

Orthogonality. Both inputs and outputs of STARQL queries are timestamped RDF triples. Therefore, triples, coming from the result of one query, can be used as input when constructing another query.

Scope locality. While producing a STARQL query, one can select an ontology and streams over which the query will be evaluated. This feature can be important in different cases, e.g., in the case of failure testing, where one is interested in querying only the streams stemming from sensors that are (or are not) suspected to be broken.

Library functions. Often-used query patterns can be stored in a special library and re-used during query construction.

Same interface for historic and stream data. Roughly the same STARQL queries can be used to query historic data (timestamped data in a DB) or to query real-time streams.

Now we illustrate the STARQL framework by example.

[3] http://virtuoso.openlinksw.com/.

[4] http://www.revelytix.com/content/spyder.

Example 8. Consider the preventive monitoring request from Example 1. To fulfil it, the following sub-task should be performed: "*Detect a monotonic increase from the temperature sensor*". We now see how this detection can be done within the STARQL framework.

First, assume that the data stream S_Msmt is being received from the sensor; its sub-stream that contains data received during the first five seconds is as follows:

```
{{s0 :val 90}<0s>, {s0 :val 93}<1s>,
 {s0 :val 94}<2s>, {s0 :val 92}<3s>,
 {s0 :val 93}<4s>, {s0 :val 95}<5s>}.        (1)
```

This data is in the form of timestamped RDF triples. For example, the first triple {s0 :val 90}<0s> says that at the time point "0s" the sensor s0 sent the value 90.

Consider the following STARQL query fulfilling the task:

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

Intuitively, the structure of the query is as follows:

(i) The HAVING clause specifies that the sensor's value should monotonically increase.
(ii) The FROM clause tells that the query performs its check every second, considering only the data from the stream S_Msmt in the last two seconds.
(iii) The SEQUENCE BY clause groups the output triples using some standard method StdSeq.
(iv) The CREATE clause declares the query's output stream S_out_1.
(v) The CONSTRUCT clause determines the format of the timestamped RDF triples in the output stream. For instance, the output stream corresponding to the input data stream from Eq. (1) is

```
{{s0 rdf:type RMInc}<0s>,
 {s0 rdf:type RMInc}<1s>,
 {s0 rdf:type RMInc}<2s>,
 {s0 rdf:type RMInc}<5s>}                    (2)
```

where RMInc stands for Recent Monotonic Increase, so the timestamped RDF triple {s0 rdf:type RMInc}<2s> designates that the sensor s0 has been experiencing a monotonic increase for the last two seconds, from 0s to 2s. (As in SPARQL, instead of the CONSTRUCT keyword one can also use SELECT if the output is to consist only of tuples of variable bindings).

Under the OBDA approach, data are stored in relational form and not in RDF format. Hence, the STARQL engine operates on the virtual stream S_Msmt induced by mappings from some actual relational streaming source. ■

In order to assess the STARQL capabilities and functionalities, we have extended the overview tables in [39] to contain all relevant streaming query languages (that we are aware of). The results are shown in Tables 1 and 2.

Table 1 provides a comparison of streaming languages with respect to various SPARQL characteristics and features. While all languages support basic functionalities like union, join, optional and filter, some of them (including STARQL) have already incorporated SPARQL 1.1 expressiveness with IF clauses, aggregations, arithmetic expressions (not listed here) and more. Furthermore, all of them, except from EP-SPARQL (which is more based on events than time), are supporting temporal windows, though only three of them support triple windows.

The specific streaming capabilities and operators of each query language are presented in Table 2. We can identify two groups of query languages, which differ in the management of time and of temporal operators in general.

On the one hand, we have a group that allows access to timestamps by functions on each triple or object within windows. Those include SPARQLStream, C-SPARQL, and STARQL. While SPARQLStream uses reified time with additional axioms and STARQL a non-reified version with a semantics of temporal states, C-SPARQL uses an in-between approach offering temporal functions on objects for retrieving their timestamps. The latter could lead to inconsistencies, if an object occurs several times inside a window in different temporal states.

On the other hand, we find a group of languages that are developed with respect to temporal sequences and specific sequencing operators like in EP-SPARQL, TEF-SPARQL, and STARQL that are tailored for *complex event processing* (CEP). Though EP-SPARQL is different from the two other approaches as it is a more CEP based language, they all share the possibility to define temporal sequences with operators. EP-SPARQL extends SPARQL by four new binary operators: SEQ, EQUALS, OPTIONALSEQ and EQUALSOPTIONAL; while TEF-SPARQL defines temporal facts and STARQL makes use of its special HAVING clause.

Finally, STARQL offers several new operators with functionalities that have not been included in previous systems. See Table 2 for these features. All columns except for the column "W-to-S operator" are self-explaining. W-S refers to the three operators Rstream, Istream, and Dstream of [38] that describe different ways of creating and outputting a stream from window contents. Rstream outputs every triple in the window, Istream only outputs triples inserted into the window, and Dstream outputs only deleted ones. Next to cascaded streams, which can be seen as temporal sub-queries, STARQL offers the possibility of querying historically recorded data or even comparing them to a live stream (see [13,44]). Those different kinds of input streams (possibly using different kinds of window widths and slides) can additionally be synchronised in STARQL by one or more pulse functions, allowing for a regular query output for possibly asynchronous input. Moreover, due to the integration of optimised UDFs from ExaStream (such as an optimised version of the correlation function), STARQL offers the main components for an analytics aware OBDA approach as described in [43].

Besides a comparative presentation of the (language) functionalities of STARQL, we give a comparative presentation of relevant implementation features of the STARQL engine in Table 3. Most of RDF systems mentioned above rely on native implementations of query processors. CQELS for example reimplements functionalities, which do already exist in DSMS and therefore can be seen as standalone engine. EP-SPARQL is based on logical programming and backward chaining, but is also implemented from scratch. Finally, C-SPARQL relies on an internal DSMS, but has no flexibility for mappings or rewritings.

The only two systems using an OBDA approach with mappings and a flexible back end are SPARQLStream and STARQL. As they both rely on external DSMS, they also both suffer from the same disadvantages. Query rewriting and translation of results can be expensive, while the expressiveness of the underlying systems restricts the input of the RDF streaming queries. Nevertheless, OBDA approaches can rely on various backend optimisations to accelerate query processing.

Please cite this article in press as: E. Kharlamov, et al., Semantic access to streaming and static data at Siemens, Web Semantics: Science, Services and Agents on the World Wide Web (2017), http://dx.doi.org/10.1016/j.websem.2017.02.001.

Table 1
Comparison of RDF-stream query languages (Part 1).

Name	Data Model	Union, Join, Optional, Filter	IF expression	Aggregate	Property paths	Time windows	Triple windows
Streaming SPARQL [25]	RDF streams	Yes	No	No	No	Yes	Yes
C-SPARQL [40,40,41]	RDF streams	Yes	Yes	Yes	Yes	Yes	Yes
CQELS [29]	RDF streams	Yes	No	Yes	No	Yes	No
SPARQLStream [27,39,42]	(Virtual) RDF streams	Yes	Yes	Yes	Yes	Yes	No
EP-SPARQL [30]	RDF streams	Yes	No	Yes	No	No	No
TEF-SPARQL [32]	RDF streams	Yes	No	Yes	No	Yes	Yes
STARQL [35–37,43]	(virtual) RDF streams	Yes	Yes	Yes	No	Yes	No

Table 2
Comparison of RDF-stream query languages (Part 2).

Name	W-to-S operator	Cascading streams	Intra window time	Sequencing	Synchronized pulse	Historic data
Streaming SPARQL	RStream	No	No	No	No	No
C-SPARQL	RStream	No	Yes	No	No	No
CQELS	RStream	No	No	No	No	No
SPARQLStream	RStream, IStream, DStream	No	Yes	No	No	No
EP-SPARQL	RStream	No	No	Yes	No	No
TEF-SPARQL	RStream	No	No	Yes	No	No
STARQL	RStream	Yes	Yes	Yes	Yes	Yes

Table 3
Comparison of RDF stream engines.

Language	Input	Execution	Query optimization	Stored data	Reasoning
Streaming SPARQL	RDF streams	Physical stream algebra	Static plan optimization	Yes	No
C-SPARQL	RDF streams	DSMS based evaluation with triple store	Static plan optimization	Internal triple store	RDF entailment
CQELS	RDF streams	RDF stream processor	Adaptive query processing operators	Stored linked data	No
SPARQLStream	Relational streams	External query processing	Static algebra optimizations, host evaluator specific	Data source dependent	No
EP-SPARQL	RDF streams	Logic programming, backward chaining rules	No	No	RDFS, Prolog equivalent
TEF-SPARQL	RDF streams	Yes	No	Yes	Yes
STARQL	Relational streams	External query processing	Static algebra optimizations, host evaluator specific	Datasource dependent	Yes (DL-Lite$_{\mathcal{A}}$)

4.2. Streaming and static relational data processing

The queries produced by the STARQL translator are processed and answered by Optique's dedicated *Data Stream Management System* (DSMS), ExaStream. ExaStream has been designed for efficiently processing on both static and streaming information and the corresponding queries produced by the STARQL engine. It is embedded in Exareme,[5] a system for elastic large-scale dataflow processing on the cloud [45,46] that has been publicly available as an open source project under the MIT Licence. We give a short presentation on some key aspects of ExaStream, for a more detailed description, the user may refer to [12].

Data Model. An ExaStream *topology* describes the flow of streaming and static records between *computational 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.

Declarative Semantics for Computations. Computational nodes may perform several operations on data streams such as filtering, aggregation, joining, and interacting with data sources and databases, to produce the desired output. 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 most DSMSs, the user does not need to consider low-level details of query execution. Instead, 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).

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

(i) *Create Stream:* The create stream statement allows to add a new computational node to our topology that outputs a live stream.
(ii) *TimeSlidingWindow:* The specific operator, 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.
(iii) *WCache: WCache* creates the indexing structures for answering efficiently equality constraints on the `Wid` and `Time` attributes when processing infinite streams. *WCache* will then produce results to multiple queries accessing different streams.

Architecture & Implementation. ExaStream supports *parallelism* by allocating processing across different workers in a distributed environment. Its architecture is shown in Fig. 5. Queries are registered

[5] https://www.exareme.org.

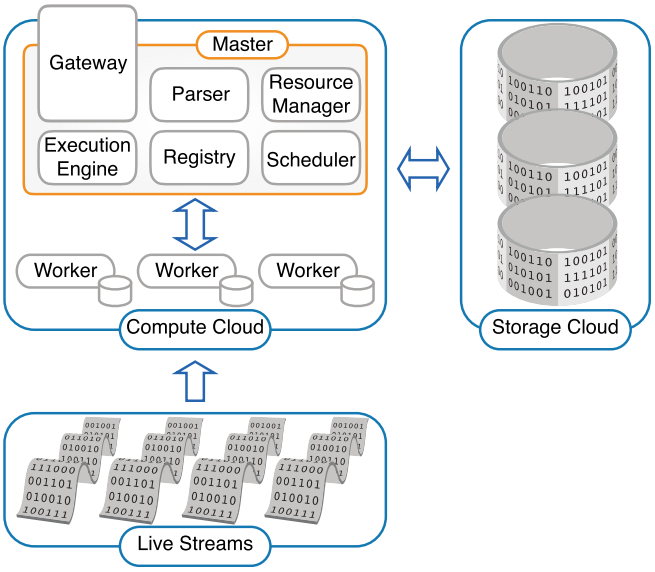

Fig. 5. Distributed Stream Engine Architecture.

through the Gateway Server. Each registered query passes through the ExaStream parser and then is fed to the Scheduler module. The Scheduler places data and compute operators (including UDFs and relational plans) on worker nodes based on each worker's load. These operators are executed by an SQLite[6] 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 computational nodes with a static input, ExaStream provides intra-query parallelism. This means that each operation of a query is distributed on multiple workers.

The ExaStream system natively supports *User Defined Functions* (UDFs) with arbitrary user code. The engine blends the execution of UDFs together with relational operators using *Just-In-Time* tracing compilation techniques. This greatly speeds-up the execution as it reduces context switches, and most importantly, only the relevant execution traces are used, allowing the engine to perform optimisations at runtime that are not possible when the query is pre-compiled. UDFs allow to express very complex dataflows using simple primitives. Communication with external sources, window partitioning on data streams, and data mining algorithms such as the *Locality-Sensitive Hashing* technique [47] for computing the correlation between values of multiple streams are implemented as UDFs.

5. The Optique platform for Siemens

The Optique platform [5] is an end-to-end OBDA solution, i.e., it supports the whole OBDA cycle from deployment to query-answering visualisation. Optique platform integrates a number of existing systems and provides several new components. It was tested with various use cases, including Norwegian Petroleum Directorate Fact Pages [48], Statoil [49] and demonstrated in [50–53].[7] We now give an overview of the platform. Details on the architecture and the individual components of the platform can be found in [8,53] and by following the references given below. Optique is a commercial platform,[8] while some of its components are available under open-source licences.

5.1. Optique *platform*

The Optique platform allows to: create and edit mappings; create, edit, and import ontologies [54–59]; integrate several relational databases and data streams; formulate and visualise one time and continuous queries; efficiently process both static and streaming information; and browse query results. Thus, the Optique platform satisfies all the Siemens system Requirements R1–R5. Note that the current version of Optique provides full support of distributed query processing on streaming and static information, in contrast to the Optique system described in [10].

The query formulation, transformation, execution, and answer visualisation procedures are performed in a sequence of stages presented in Fig. 6:

(i) *Query Formulation*: After the system is deployed, the underlying data sources can be queried via our query formulation tool OptiqueVQS. OptiqueVQS allows to compose queries by navigating over the system's ontology and constructing simple graphs corresponding to queries for standard ontologies or their streaming/geospatial extensions. Graphs are internally translated by OptiqueVQS to STARQL expressions, i.e. queries designed to retrieve information from streaming ontologies.

(ii) *Query Transformation*: The aforementioned expressions are sent to the corresponding query transformation engine for processing. The processing includes rewriting against the ontology and further unfolding into relational queries based on the corresponding mappings [60]. STARQL expressions (i) on historical data are rewritten and unfolded to pure SQL queries; (ii) on streaming data are rewritten and unfolded to SQL$^{\oplus}$ queries by the STARQL2SQL$^{\oplus}$ query transformation engine [36]. SQL$^{\oplus}$ is an extension of standard SQL with operators for stream handling.

(iii) *Query Execution*: In the query execution phase: (i) SQL historical queries are executed by the database management system that stores the historical information; (ii) SQL$^{\oplus}$ streaming queries are executed by ExaStream, a system for large scale elastic stream processing on the cloud that uses parallelism to cope with the huge data sets provided by Siemens.

(iv) *Visualisation & Analysis*: Resulting query answers are visualised using templates and widgets such as tables, timelines, maps, charts, etc., depending on the data modalities. In order to support analytical tasks required by turbine diagnostics, we developed a native integration of Optique with KNIME[9,10] data analytics system to streamline query answers into analytics, and a configurable plug-in for R analytics.[11]

The Optique platform implementation is based on the *Information Workbench* (IWB) [61], a generic and extensible platform for semantic data management which provides a rich infrastructure

6 https://www.sqlite.org.

7 Optique demo video: www.youtube.com/user/optiqueproject/playlists.

8 https://www.fluidops.com/en/company/research/optique.

9 KNIME, the Konstanz Information Miner, is an open source data analytics, reporting and integration platform. KNIME integrates various components for machine learning and data mining through its modular data pipelining concept. A graphical user interface allows assembly of nodes for data preprocessing (ETL: Extraction, Transformation, Loading), for modelling and data analysis and visualisation.

10 https://www.knime.org/.

11 https://www.r-project.org/.

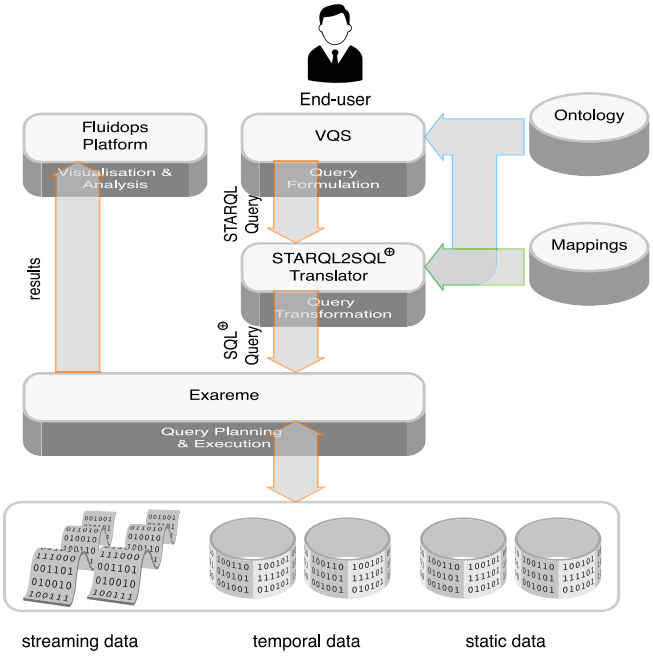

Fig. 6. System Architecture.

for our platform. We now focus on parts of the OPTIQUE platform that are tailored towards Siemens: OPTIQUEVQS and diagnostic dashboards.

5.2. *Visual query formulation with* OPTIQUEVQS

The Optique platform allows domain experts to formulate and pose queries via a visual query formulation system, called OPTIQUEVQS [62–67]. Queries, formulated via OPTIQUEVQS, are automatically translated to the underlying query language (i.e., SPARQL or STARQL) and sent to the query transformation module. In the context of the Siemens use case, OPTIQUEVQS has been extended to support STARQL to allow formulation of streaming queries [64,68,69]. OPTIQUEVQS is composed of a front-end (i.e., interface) and backend component feeding the interface with ontology fragments.

Interface. OPTIQUEVQS is a widget-based user-interface mashup (UI mashup) [62,70], that is, individual communicating widgets are the building blocks of OPTIQUEVQS. This approach offers flexibility, modularity, and adaptability and enables us to combine multiple representation paradigms, such as forms, diagrams and icons [71], and interaction paradigms, such as schema navigation, range selection, and matching [71].

In the Siemens use case, a user interacts with OPTIQUEVQS with five widgets:

(i) The first widget (W1), see the bottom left hand side of Fig. 7, is a menu-based widget and allows the user to navigate through concepts of an ontology by selecting relationships between them.
(ii) The second widget (W2), see the top side of Fig. 7, is a diagram-based widget and presents typed variables as nodes and object properties as arcs. It gives an overview of the query formulated so far.
(iii) The third widget (W3), see the bottom right hand side of Fig. 7, is a form-based widget and presents the attributes of a selected concept for selection and projection operations. Dynamic attributes (that is attributes that change over time) are coloured in blue.
(iv) The fourth widget (W4) is a form-based widget [68], see Fig. 8, and it lets the user configure parameters for temporal queries. W4 is available from a "Stream" button (see Fig. 7) as soon as the query involves dynamic attributes.
(v) The fifth widget (W5) is a tabular widget, see Fig. 9, that allows to select a template for the temporal query, and to register the user query for execution. In the case of SPARQL, this widget is used to present example results and provide functionality for sorting and aggregation operations, such as sum, max, min, and average.

A typical query formulation process begins with selecting a starting concept (called *kernel*) from W1. Each selected concept appears in W2 as a variable node and the attributes of the selected concept appear in W3. The concept chosen last automatically becomes the active node (called *pivot*) and the user can change it either by adding a new node through W1 or by clicking on an already existing node in W2. Once there is an active node in W2, W1 does not present a pure list of concepts any more, but it lists object property and range concept pairs pertaining to the active node. The user can add as many branches as he/she wishes by using W1 and W2, and add constraints and select attributes for output using the form elements presented in W3.

If a dynamic attribute is involved in the query, OPTIQUEVQS switches to STARQL mode and a button named "Stream" appears to activate the parameter configuration widget. Once the user clicks on the "Result Overview" button, the template selection widget appears. The user can also save/load and modify queries through W2. Hence, OptiqueVQS splits the formulation of streaming queries into two steps where the user has to specify which data streams are of interest, corresponding to the static part in the `WHERE` clause, and what is to be done to the specified streams. The interface for the latter allows users to pick from a list of options that include range checks, gradient checks, and spikes, and cover a large part of the query tasks needed day to day. Adding more options, or changing the queries produced for each, are simple programming tasks.

OPTIQUEVQS is free from any technical jargon, for example related to OWL and STARQL. It employs a simplified tree-shaped query representation and distributes functionality to different widgets with respect to the accord between the functionality, interaction, and representation paradigms (i.e., tabular widget for template selection, and menu-based widget for navigation). OPTIQUEVQS supports tree-shaped conjunctive queries and a fragment of STARQL. Currently, STARQL queries correlating different dynamic attributes and queries involving cycles, negation, and disjunction are not supported.

Backend. The front-end communicates with the backend via a REST API that returns a JSON object according to the performed request. It is mainly responsible for accessing and serving ontology fragments to the interface, as a user interacts with the system, and for dealing with the query log and data to improve user experience.

The main component of OPTIQUEVQS's backend is a graph projector [63,65], which feeds OPTIQUEVQS's widgets in order to enable a graph-based navigation over an ontology during query formulation. Graphs are effective mechanisms to navigate, construct, and communicate complex topological structures for end users. It is also well-known that the majority of end-user queries are conjunctive, and thus, in the semantic web setting, they could naturally be seen as graphs since we are dealing with unary and binary predicates only. However, note that OWL 2 axioms do not have a natural correspondence to a graph. Even when a set of range/domain axioms naturally suggest a graph, to the best of our knowledge there is no standard means to translate it to a graph. Therefore, we need a technique to extract a suitable graph-like structure from a set of OWL 2 axioms. For this purpose, we have

Please cite this article in press as: E. Kharlamov, et al., Semantic access to streaming and static data at Siemens, Web Semantics: Science, Services and Agents on the World Wide Web (2017), http://dx.doi.org/10.1016/j.websem.2017.02.001.

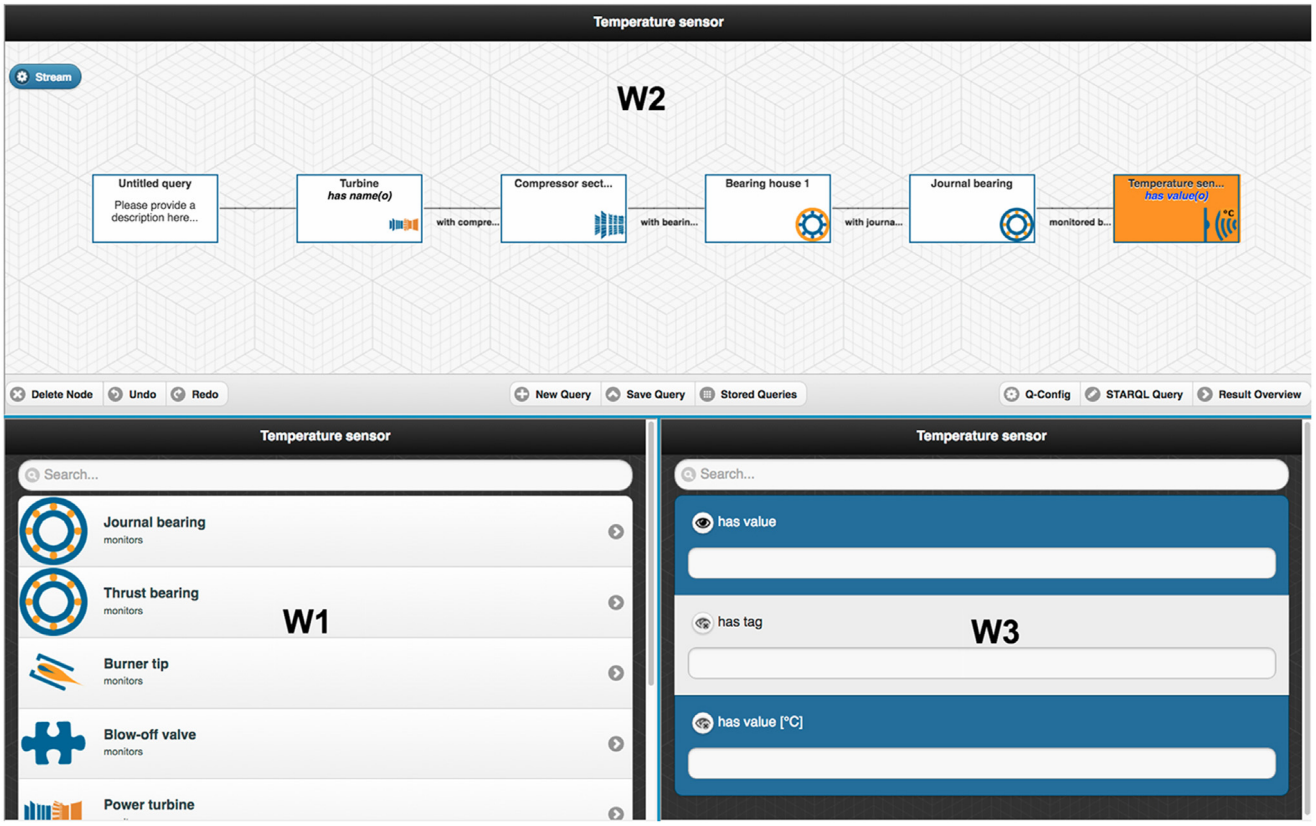

Fig. 7. OptiqueVQS interface—dynamic properties are coloured in blue. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

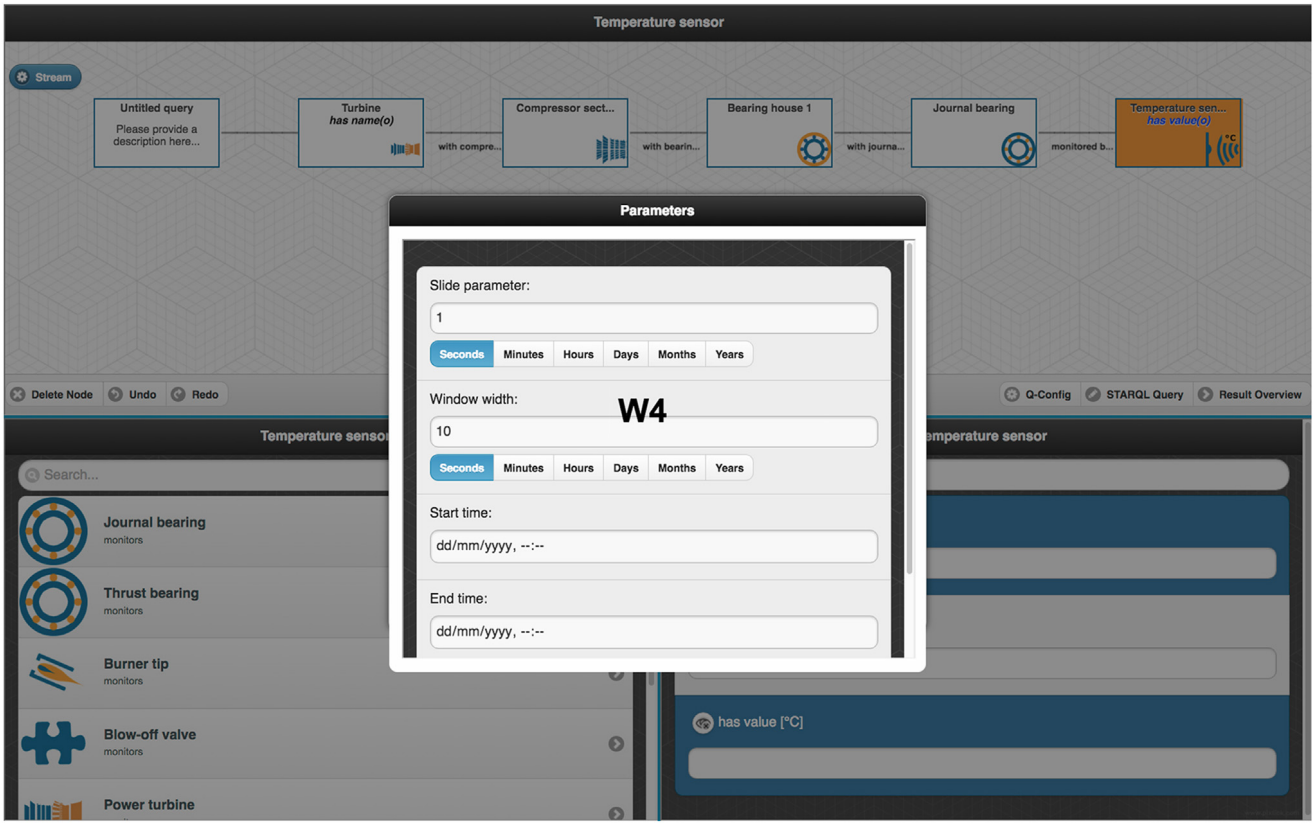

Fig. 8. OptiqueVQS interface—parameter selection.

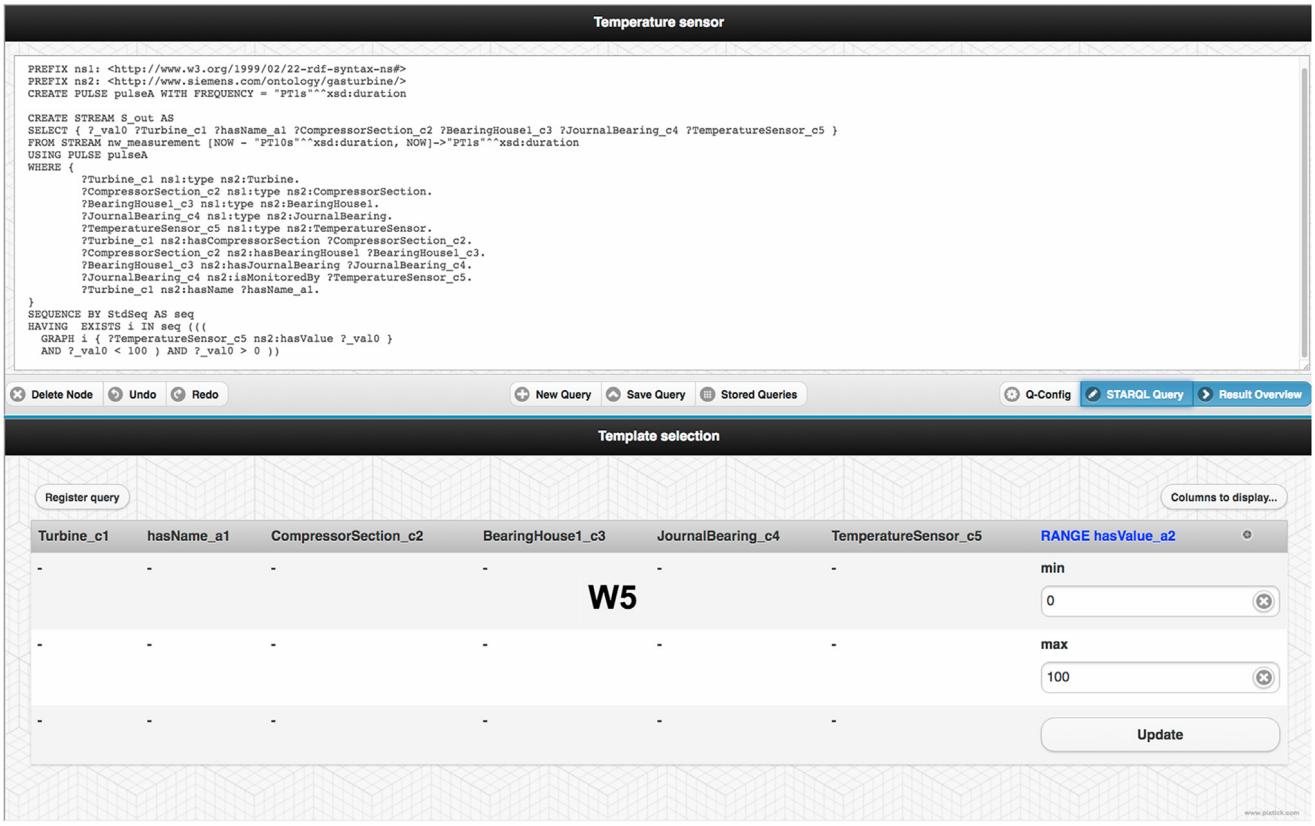

Fig. 9. OptiqueVQS interface –template selection and the generated STARQL query.

adapted a technique called navigation graph [72–79]. OptiqueVQS uses the OWL 2 reasoner HermiT [80] to build the navigation graph (e.g., extraction of classification) in order to consider both explicit and implicit knowledge defined in an ontology.

The backend also provides for a data sampler component allowing us to enrich an ontology with additional axioms to capture values that are frequently used and rarely changed. This includes the list of values and numerical ranges in an OWL data property range. Such an approach allows presenting attributes in different types, such as sliders, multi-select boxes, date pickers etc., with respect to the underlying data. Moreover, the backend maintains a query log for ranking and suggesting query extensions as a user formulates a query, that is, the W1 and W3 list concepts and properties adaptively with respect to a partial query given at any time [81].

5.3. Diagnostic dashboards and integration with Siemens analytics

In order to address diverse needs of end users we developed a flexible wiki-based *Diagnostic Dashboard* that can be easily customised by end users themselves. Result visualisation widgets allow the user to visualise query answers, inspect query results, do incremental query refinement, and export the relevant result fragments to external diagnostic tools. Moreover, the widgets support monitoring of incoming data streams and query answers for continuous queries over these streams. In Fig. 10 we present three examples of our visualisation widgets.

The D1 part of Fig. 10 shows the sensor data inspection screen for one specific turbine. Here, the user has chosen a turbine of interest and reviews data of the previous 24 h. The main chart in the centre contains thermocouple data, i.e., mostly temperature readings along the gas path of the turbine, starting from the air inlet through the burner stage to the exhaust. Without going into detail the chart suggests that the turbine of interest has been shut down and re-started numerous times in the first quarter of the day. After that it appears that the day has continued uneventful, at least in terms of the temperature profile.

If a specific sensor in the overview exhibits missing values, unusual trends, or an inadmissible number of outliers then the service engineer can inspect that sensor on a higher level of detail in the single-sensor view depicted on the top-right of Fig. 10 and annotated with D2. The dashboard shows three views: the live data stream from the sensor in question, historic data as chosen by the user, and a configurable plug-in for R analytics. The comparison between historic and live data can be used for ruling out any persistent or recurring abnormalities in the sensor readings. Conversely, all in-depth statistical analysis of sensor data can be performed and inspected in the R view. Here the underlying statistics can be configured from a library of common analytics solutions.

The lower part of Fig. 10, annotated D3, contains other turbine monitoring dashboards with various kinds of aggregated data. Note that just like the raw data views above, these aggregated views are configured by choosing an appropriate query from the query catalogue—with the query again formulated against the turbine knowledge model rather than the actual data sources. Depending on the type of data (e.g., time series data, appliance structure), a suitable visualisation paradigm has to be selected (e.g., pivot table, trend diagram, histogram). The diagnostic dashboard can also choose the representation paradigm for query answers automatically by analysing the corresponding SPARQL query.

Since a wide range of Siemens diagnostics tasks are realised using KNIME, we have also developed a KNIME extension that gives access to Optique from KNIME analytical workflows and streamline answers computed by Optique directly into KNIME. In Fig. 11 we present a screenshot of a diagnostic task implemented in KNIME and in Fig. 12 is the zoom of Fig. 11 in a fragment relevant for

Please cite this article in press as: E. Kharlamov, et al., Semantic access to streaming and static data at Siemens, Web Semantics: Science, Services and Agents on the World Wide Web (2017), http://dx.doi.org/10.1016/j.websem.2017.02.001.

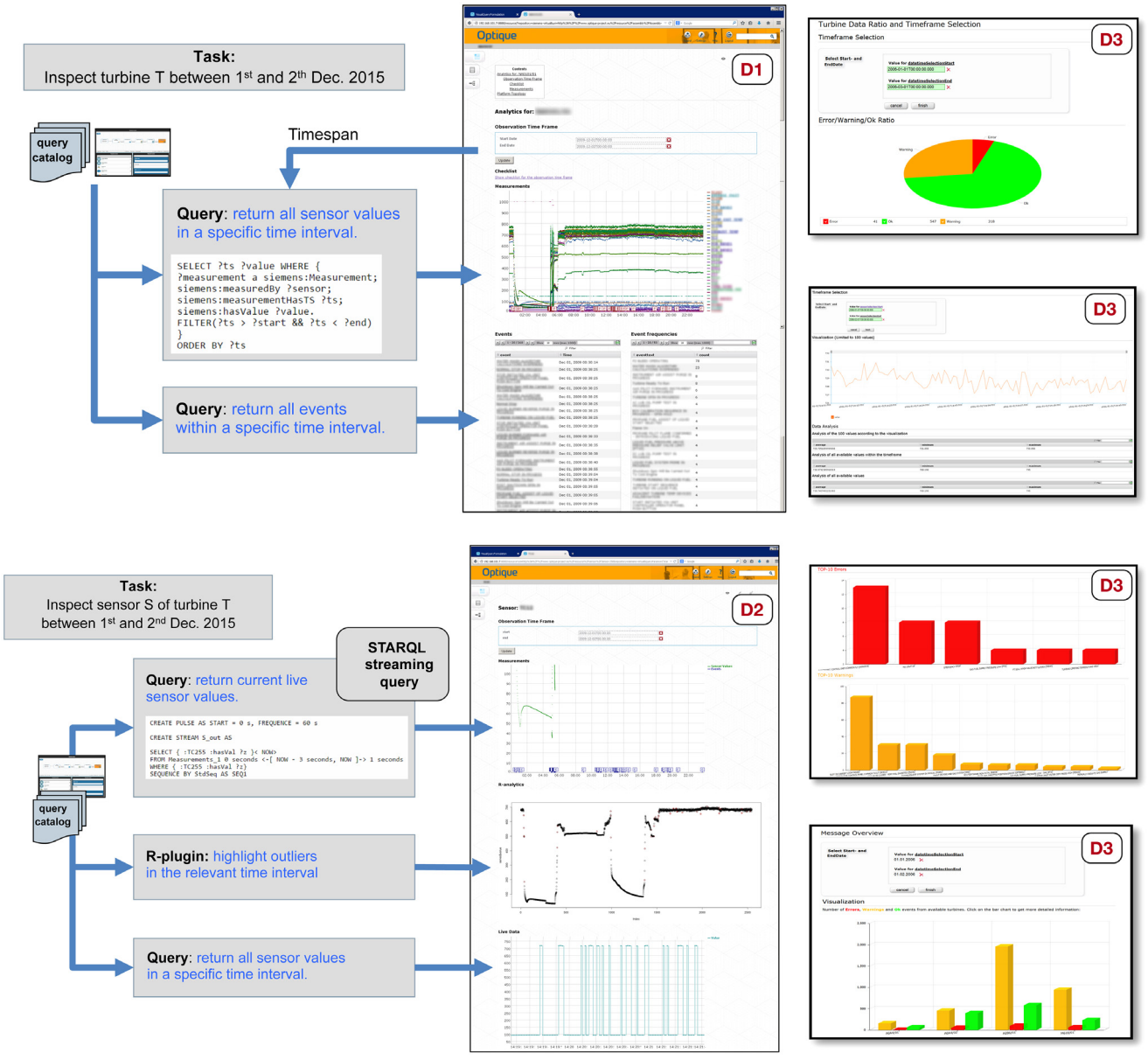

Fig. 10. Diagnostic dashboards.

analytics. In Fig. 11 one can see that the data layer is disconnected from the analytics layer by the OPTIQUE semantic layer and shown only for orientation, that is, the data from the DBs go first through the OBDA platform and only then analytics are applied on it. One can add a new semantic OBDA data source to the system using either a SPARQL or STARQL query against the OPTIQUE platform.

In Fig. 12 there is an example SPARQL query over the OBDA platform. The data produced by this query are then used to compute key *performance indicators* (KPI) and to check for outage in the turbines. Observe that the node of the work-flow diagram that corresponds to KPI computation is defined using KNIME rules that rely on the standard KNIME syntax while they are formulated against the ontological concepts. These rules say that a turbine is deemed to be in service at any time if either it is a gas turbine or a turbo compressor that satisfies extra conditions. In the former case these conditions say that the sensor signal of the rotor speed sensor should have readings above its characteristic operational speed value, the main flame sensor reading should show that the flame is on, and the turbine should generate power, i.e., Power Sensor should be above the value characteristic for that turbine while generating power. In the latter case the conditions are that the generated pressure should be at or above the nominal pressure specified for the machine.

Observe that due to ontologies each occurrence of "RangeMaxValue" in the KNIME rules is a different type of value read from the static configuration data of the machine and this can be encoded using property hierarchy. Indeed, for the latter case one can achieve it by stating that "RunningSpeedConfigValue" is a sub-property or "RangeMaxValue" and in the latter case that "MainFlameOnSignal" is a sub-property of "RangeMaxValue". Moreover, an advantage of such OBDA-backed KNIME diagrams is that one can talk about specific values in turbines and even compressors of different types at an abstract level, without giving details of such appliances. Finally, observe that KNIME has a sophisticated reporting functionality which we exploit in our system: the last node in the work-flow diagram, called Data to Report, summarises the KPI for all turbines across a specific fleet and builds a ready-to-use report for them.

Please cite this article in press as: E. Kharlamov, et al., Semantic access to streaming and static data at Siemens, Web Semantics: Science, Services and Agents on the World Wide Web (2017), http://dx.doi.org/10.1016/j.websem.2017.02.001.

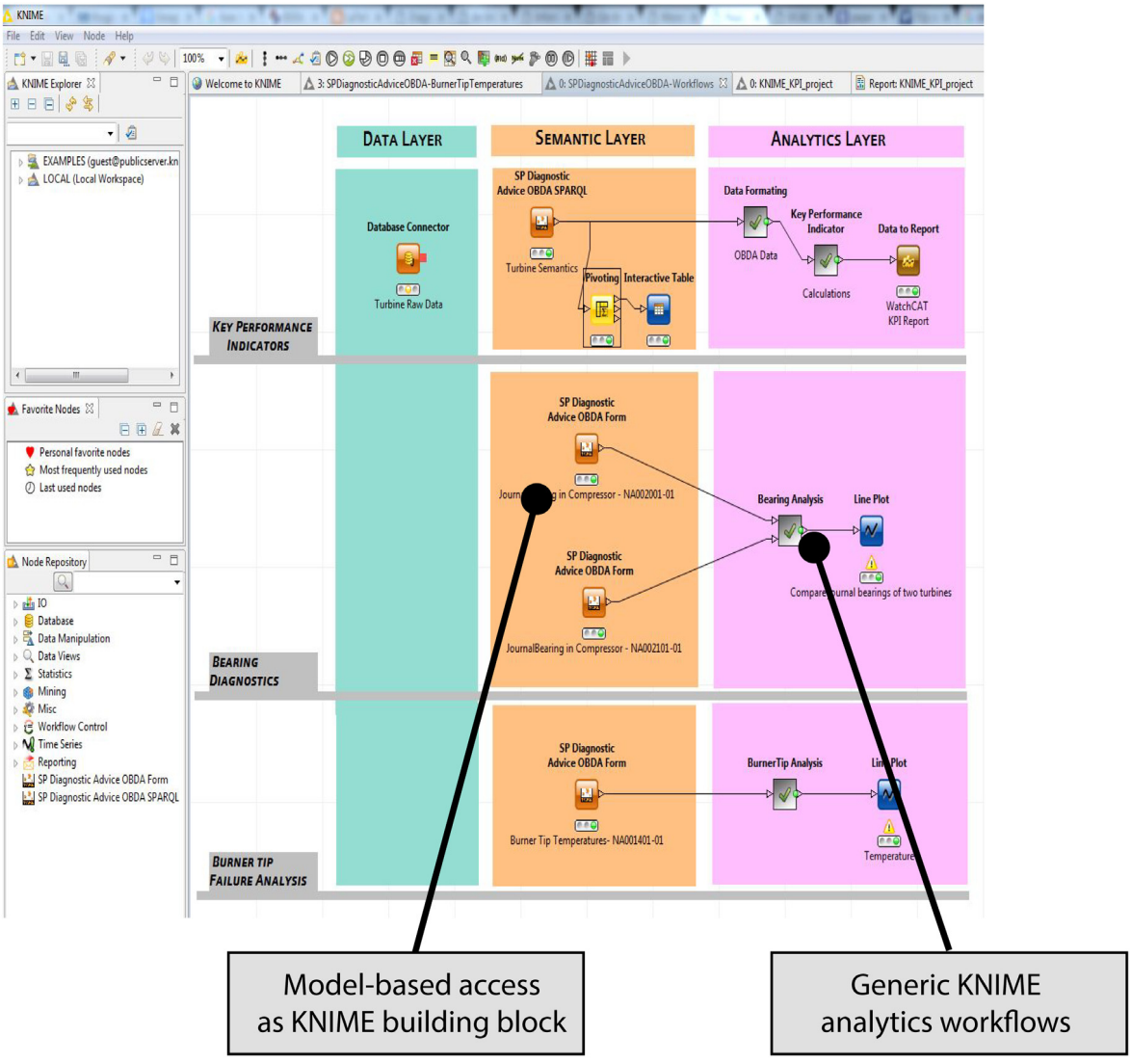

Fig. 11. Optique integration with KNIME.

6. Evaluation of OPTIQUE at Siemens

In order to demonstrate the potential of OBDA in enhancing data access in Siemens we did a preliminary deployment of the Optique OBDA platform over Siemens data and conducted a preliminary user study. We now give details of our experience.

6.1. Siemens ontology

We start with the ontology that we developed for Siemens; it is a critical component of our deployment which we used in both our evaluations. Development of an industrial ontology is a non-trivial task [82,83]. Although there are ontologies describing machinery with sensors, e.g., the *Semantic Sensor Network* (SSN) ontology [84], we could not use them: for our use case, they are too generic and overloaded with irrelevant terms, moreover, they miss required terms. Therefore, we constructed our ontologies being guided by the best practices of the SSN ontology. Our ontologies characterise Siemens database schemata of sensor and event data and abstract away from representations varying across data sources. Moreover, our ontologies are manually enriched with the domain information encoded in multiple semiformal and informal models available at Siemens. We developed three ontologies: (i) the turbine, (ii) sensor, and (iii) diagnostic ontology.

The *turbine ontology* describes the internal structure of a turbine, i.e., it lists all its parts, functional units, and their hierarchy. For example, it models that every `Turbine` must have a `ControlSystem` and a `Generator`, and that `LiquidFuelPump` is a part of a `LubOilSystem`. The ontology contains 60 classes and 15 object and datatype properties. There are three central classes in this ontology: (i) `Turbine` class for modelling product families of appliances, (ii) `Component` for modelling a hierarchy of subclasses defining the types of components that turbines are constructed of, using relations such as `hasPart` and `hasDirectPart`, (iii) `FunctionalUnit` for defining functional interrelation of components, i.e., important blocks of an appliance, such as `GasPath`, `FuelSystem` and others, as well as components belonging to these functional units.

The *sensor ontology* lists and categorises types of sensors and measuring devices mounted in a turbine as well as their deployment, measurement properties, such as accuracy and precision, and other related information. For example, it models that each sensor is mounted at some turbine's component or functional unit, or that a sensor of a specific type does only produce observations of a given type. The ontology contains 40 classes and 20 properties. The main class `Sensor` covers all types of measuring devices, e.g., `GasDetector`, `TemperatureSensor`.

Please cite this article in press as: E. Kharlamov, et al., Semantic access to streaming and static data at Siemens, Web Semantics: Science, Services and Agents on the World Wide Web (2017), http://dx.doi.org/10.1016/j.websem.2017.02.001.

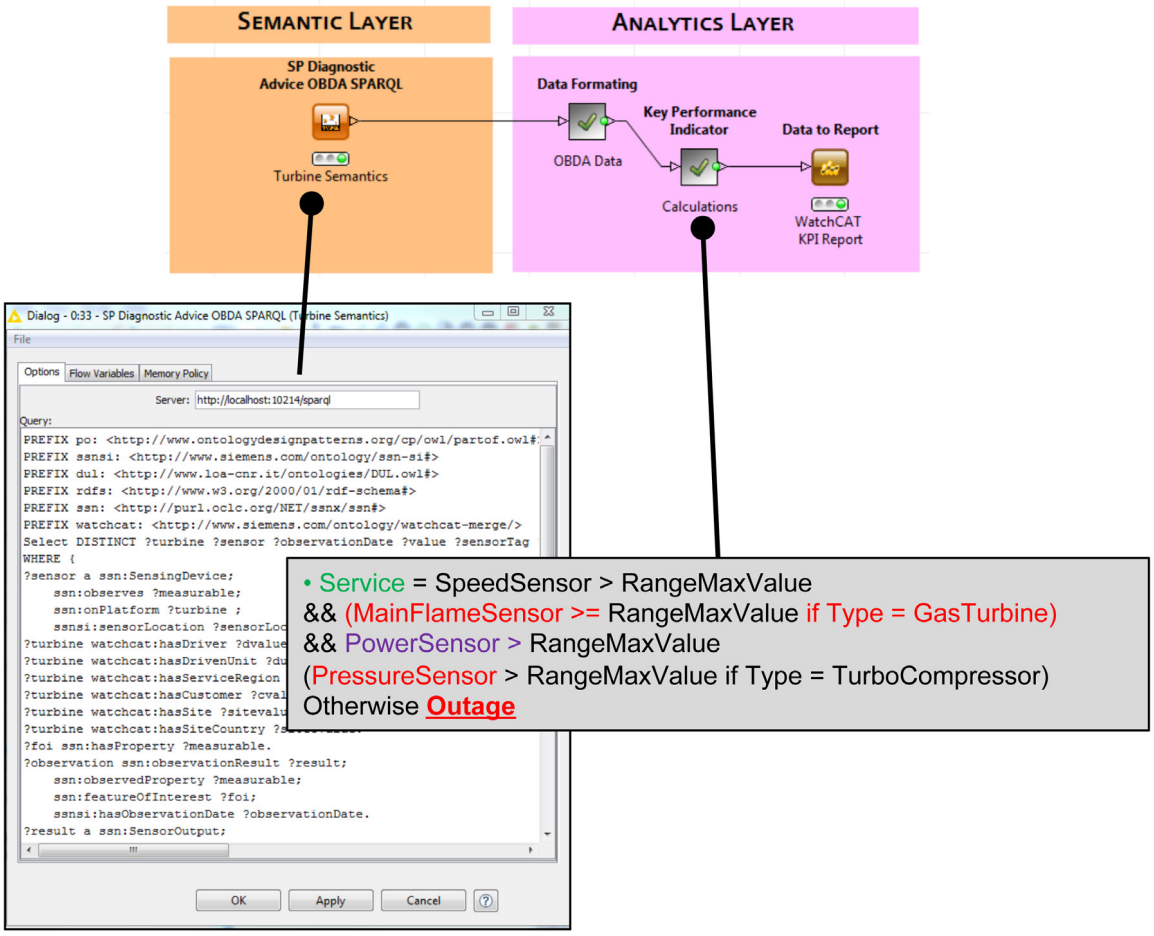

Fig. 12. Optique integration with KNIME.

Further branching on classes gives more detailed characteristics information on them, e.g., temperature sensors could measure: `BurnerTipTemperature`, `InletTemperature`, `CompressorExitTemperature`, etc.

The *diagnostics ontology* formalises relationships between measurements and events generated by turbine's sensors and control units, as well as typical symptoms of faults in turbines. For example, it models that each diagnosis has to be assigned to a turbine or its component, and must be supported by some symptoms. The ontology contains 30 classes and 10 properties. The core classes are `Observation` and `Diagnosis` connected with a relation `indicatesAt` for listing symptoms for each diagnosis.

Each of the three ontologies can be used independently and we have also developed an ontology that integrates the three. The turbine and sensor ontologies are expressed in OWL 2 QL, a tractable profile of the OWL 2 ontology language and therefore can be used straightforwardly in our OBDA setting. The diagnostics ontology must be represented in a richer ontology language: OWL 2 DL. An example of a diagnostic axiom that cannot be expressed in OWL 2 QL is: "If `Turbine` has `Failure` F1, then there must be a `Symptom` S1 in `Turbine Component` C1". To support answering diagnostic queries using OBDA, we provided an approximation of the ontology into OWL 2 QL.

6.2. Visual query formulation

Requirements. A set of site visits and user interviews have been conducted to collect design requirements. The information needs in the query catalogue were translated into actual queries, resulting in 40 queries, which were then analysed accordingly [63]. The following three main dimensions [71] have been taken into consideration for the design and implementation of OptiqueVQS:

(i) *User synopsis*: The target group at Siemens are domain experts, who have extensive domain knowledge. However a number of domain experts lack technical skills and knowledge such as on databases, programming, and query languages.
(ii) *Interaction synopsis*: The demand for information is frequent and information needs are varied. Moreover, domain experts often have unpredictable needs in terms of the data they are interested in extracting.
(iii) *Task synopsis*: Domain experts' information needs are often structurally complex, requiring to join multiple relations.

We have conducted a literature survey [85,86] to identify best practices and design patterns. The survey revealed that a multi-paradigm design combining multiple representation and interaction paradigms is key to address a broad range of user groups having frequent, unpredicted and sophisticated information needs [71,87]. According to a framework proposed by Catarci et al. [71], the following requirement has been reached:

Design: Provide a multi-paradigm user-interface having a diagram-based paradigm in the core supported by form-based and iconic representation paradigms.

Regarding the target expressiveness level, our goal is to address frequently needed query types that are comparatively less complex

Please cite this article in press as: E. Kharlamov, et al., Semantic access to streaming and static data at Siemens, Web Semantics: Science, Services and Agents on the World Wide Web (2017), http://dx.doi.org/10.1016/j.websem.2017.02.001.

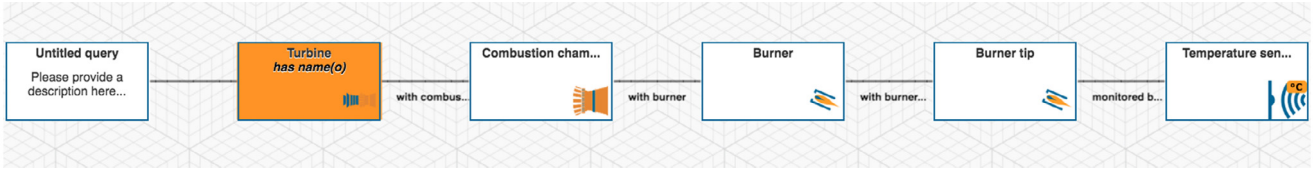

Fig. 13. A linear conjunctive query formulated by domain experts in the Siemens experiment.

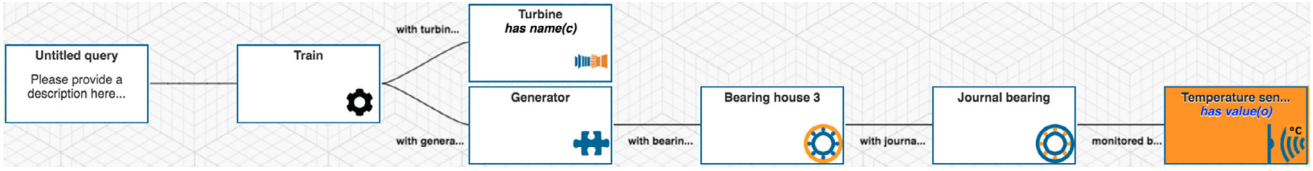

Fig. 14. A tree-shaped conjunctive query formulated by domain experts in the Siemens experiment.

for end users. According to the Siemens query catalogue, 70% of queries are tree-shaped conjunctive queries. Therefore, we are led to the following requirement:

Expressiveness: Primarily support the formulation of tree-shaped conjunctive queries.

Finally, data sources at Siemens have a temporal dimension, leading to the following requirement:

Support: Provide domain specific components to address temporal-stream data sources.

OptiqueVQS meets all these requirements as it combines multiple representation and interaction paradigms through various widgets including two widgets for stream-temporal querying. Currently, 65% of the queries in the query catalogue are supported by OptiqueVQS, that is, tree-shaped conjunctive queries (i.e., excluding queries with negation).

Evaluation. Two user experiments have been conducted with domain experts at Siemens to measure the efficiency and effectiveness of domain experts with OptiqueVQS [63–65,68]. We provide an overview of the experiments and results in the following.

The first experiment is based on non-temporal queries and relies on the diagnostic ontology, while the second experiment is based on temporal queries with the turbine ontology. Four respectively three users took part in the experiments. None of the participants had knowledge of semantic web technologies. Participants completed the following tasks during the experiment, given at most three attempts for each task, while being observed by an observer. The first half of the tasks have been used in the first experiment while the second half were used in the second experiment. The last three tasks are temporal.

(i) Find all assemblies that exist in the system.
(ii) Show all messages that turbine "NA0101/01" generated from "01.12.2009" to "02.12.2009".
(iii) Show all turbines that sent a message containing the text "Trip" between "01.12.2009" and "02.12.2009".
(iv) Show all event categories known to the system.
(v) Show all turbines that sent a message category "Shutdown" between "01.12.2009" and "02.12.2009".
(vi) Display all trains that have a turbine and a generator.
(vii) Display all turbines together with the temperature sensors in their burner tips. Be sure to include the turbine name and the burner tags.
(viii) For the turbine named "Bearing Assembly", query for temperature readings of the journal bearing in the compressor. Display the reading as a simple echo.
(ix) For a train with turbine named "Bearing Assembly", query for the journal bearing temperature reading in the generator. Display readings as a simple echo.

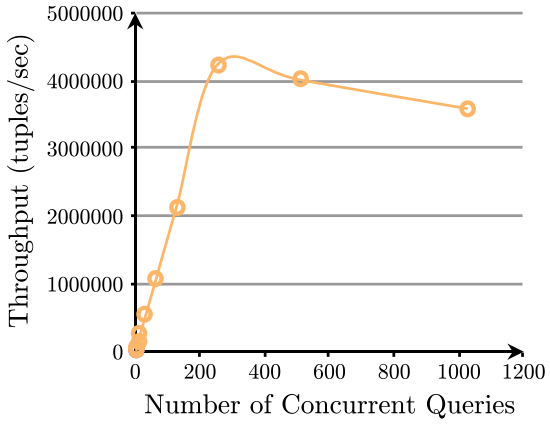

Fig. 15. Effect of parallelism on live-stream queries.

(x) For the turbine named "Burner Assembly", query for all burner tip temperatures. Display the readings if they increase monotonically.

In the first experiment, a total of 18 tasks was completed by the participants. Correct completion rate was 88% and first attempt correct completion rate was 72%. In average, a task took 1.5 attempts and 132 s to complete. In the second experiment, a total of 15 tasks were completed by the participants with 100% correct completion rate and 66% first-attempt correct completion rate. In average, a task took 1.3 attempts and 103 s to complete. Figs. 13 and 14 represent tasks (vii) and (ix) respectively. The results suggest that domain experts could translate their information needs into queries with high effectiveness and efficiency by using OptiqueVQS. One prominent wish from the domain experts is dynamic filtering of attributes in W3 with respect to active constraints. This is analogous to faceted search where facets and facet values are filtered and removed as constraints added or relaxed. Such an approach requires a mechanism for active interaction with the underlying data, which could also be precomputed offline and stored in the ontology as annotations.

6.3. Performance demonstration

The aim of the performance demo was to showcase how query distribution to multiple workers can accelerate the overall execution time of different analytic queries that involve live-stream and hybrid operations. Siemens engineers were exhibited the system's

Please cite this article in press as: E. Kharlamov, et al., Semantic access to streaming and static data at Siemens, Web Semantics: Science, Services and Agents on the World Wide Web (2017), http://dx.doi.org/10.1016/j.websem.2017.02.001.

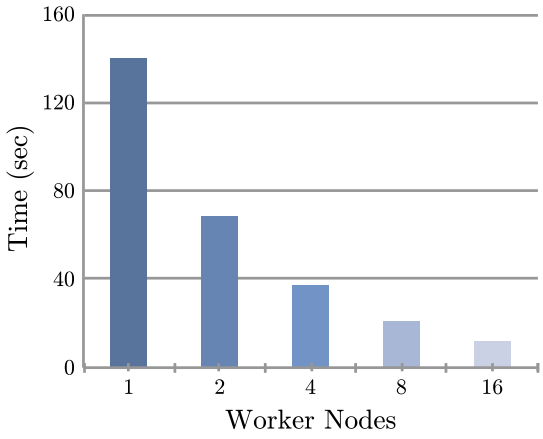

Fig. 16. Effect of parallelism on hybrid queries.

capabilities based on two predefined test queries and by being able to adjust several parameters related to the underlying queries and resources.

Demonstration setting. We deployed our system to the *MindSphere Siemens Cloud for Industry*[12] Infrastructure and used up to 128 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.

Test queries. For the performance demonstration, the following two STARQL queries were adopted: *Query I*: This query calculates the Pearson correlation between two live streams. *Query II*: This query computes the Pearson correlation of a live stream with a varying number of archived streams.

We demonstrated performance related to different types of parallelism:

Parallelism between live streams. This demo focused on the effect of accelerating live-stream operations by distributing the load to multiple worker nodes via inter-query parallelism. *Query I* could be executed (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. The window throughput was measured as the number of stream tuples that were processed per sec. Recall that each node was equipped with a two-core processor. Siemens engineers could see, as in Fig. 15, that initially the overall throughput of the system increased linearly with the number of queries. This was because ExaStream utilised the available workers and distributed the load evenly among them. When the number of queries reached the number of cores available (256) the maximum throughput of 4, 250, 226 $^{\text{tuples}}/_{\text{s}}$ could be observed. From that point onward, the additional queries injected in Exareme resulted in multiple queries sharing the same core and, as a result, the cumulative throughput decreased.

Parallelism between live & archived streams. For complex analytics such as the Pearson correlation, the ExaStream backend permits to accelerate queries by distributing the load among multiple worker nodes. In the second demo, Siemens engineers executed test *Query II*: (i) on a varying, from 1 to 16, number of available VM-worker; (ii) for a fixed live-stream velocity of 1 $^{\text{tuple}}/_{\text{min}}$; (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. As in Fig. 16, Siemens engineers could observe a significant decrease in the overall window processing time as the number of VM-workers increased. ExaStream distributed the archived relations between different worker nodes. Each node computed the Pearson coefficient between its subset of archived measurements and the live stream. As the number of archived windows was much greater than the number of available workers, intra-query parallelism resulted in significant decrease of the time required to perform the operation.

7. Lessons learned, conclusion, and future work

Lessons leaned. We organised a series of workshops with service engineers in Siemens (each was attended by up to ten people) to present our Optique OBSSDI system and to evaluate the potential of using Optique in Siemens business units. The workshops were held in two locations: at the Siemens service centre in Lincoln, UK, and Siemens AG in Munich, Germany. During the workshops we got important feedback from engineers that reinforced and guided our further development of the platform. In particular, the end users were asked to assess the system with respect to the requirements of Section 2. We now summarise the feedback.

The users gave a positive feedback on how the proposed solution addressed Requirement R1 on the integrated data access: information from different sources is integrated and can be accessed through one ontology and visualised in the diagnostics dashboard. The users provided also a very positive feedback about the query formulation components of Requirement R2 and highlighted that these facilities may greatly simplify and accelerate the process of query construction. In particular, by using the VQS interface the users were able to specify even complex queries from the query catalogue in a very intuitive way, which currently requires extensive SQL know-how as well as in-depth knowledge about the database schemata. Additionally, component suggestions on refinements and/or generalisations of the terms used in the query, additional terms and constraints, helped the users in understanding the querying capabilities of the platform and in constructing queries—which was highly valued by the users. Likewise, we received good comments on the possibility to derive implicit information using the logical reasoning—thereby satisfying Requirement R3. The hybrid stream-static data support defined in Requirement R4 which is addressed by introducing STARQL query framework was highly welcomed as an important feature for realising predictive maintenance in future. Finally, though not visible to end-users, Requirement R5 that demands for a DSMS backend providing low latency answers to queries on high-velocity live streams and high-volume static data sources is crucial for the functionality of the Optique platform.

Conclusion. In this paper we presented OBSSDI, a promising paradigm that extends classical OBDA for a direct and highly efficient end-user access to streaming, historic, and static data. We have also shown how OBSSDI could enhance such access in Siemens in the context of turbine diagnostics. We derived five requirements that an OBSSDI solution should fulfil in order to be deployed in Siemens and showed that, while the previous research and system development established the theoretical basis and demonstrated viability of OBDA, a number of limitations have to be solved before industrial deployment of such systems. Our OBSSDI solution developed within the Optique project satisfies the five Siemens requirements.

Our novel techniques behind the Optique OBSSDI system are generic and can be applied in any scenario that requires integrated and efficient semantic access to time-stamped and static data. We see this work as an important step towards ontology based data access systems of a new generation. Moreover, we believe that our Siemens experience should be valuable for both theoretical

[12] http://www.industry.siemens.com/services/global/en/portfolio/plant-data-services/cloud-for-industry/pages/default.aspx.

and practical Semantic Web researchers since it opens a number of avenues for future theoretical research and shows important practical limitations of existing systems.

Future Work. We have observed that OWL 2 QL, the profile of OWL 2 that is specifically tailored towards OBDA, is not sufficient to capture important industrial domain knowledge. For example, OWL 2 QL allows for axioms with existential quantification on the right hand side, while it is often the case that for Siemens such axioms are not natural and universal quantification is required. An important future work would be to understand how ODBA can be based on ontologies that are not OWL 2 QL and to develop practical approximation algorithms for query answering in this context. Another important observation was that diagnostic tasks required by Siemens often require computation of aggregate values, such as, averages of temperature values reported by sensors. Such computation requires to extract from databases and transfer a lot of sensor data which is then used only for aggregation, e.g., one often has to transfer thousands of temperature values just to compute one average value. We believe that it is important to enhance OBDA with a capability to push such aggregate computations down to data sources and avoid expensive data transfer. This can already be done by encoding aggregation in mappings, but we do not find it satisfactory since this makes OBDA systems query dependent. We expect a better way to enhance OBDA with aggregation capabilities.

Regarding the query language, to reach 100% coverage of the Siemens query catalogue we plan to further extend STARQL functionality in order to support a wider range of aggregate and analytic functions. These functions will be implemented as UDFs in the Exareme backend and will be incorporated to the STARQL query language semantics. Furthermore, additional sequencing strategies based on machine learning techniques are planned to be implemented. These will enable the use of STARQL for predictive diagnostics. In regard to back end optimisations, an initial line of work has been presented in the literature [12]. A central objective in cloud computing is to maintain the utilisation of the cloud high, using only the resources that are really needed for the described workload. Thus, our future work involves the adaptive adjustment of ExaStream's topology to meet the requirements of a varying workload. We further intend to extend our optimiser, so that it supports join reordering on the fly, whenever the rate of input streams changes. This involves constantly monitoring the incoming streams and making the appropriate changes on join orders and the related index structures whenever beneficial. Regarding the integration with the Siemens infrastructure, an important practical next step for us is to improve the diagnostics dashboards by allowing for automatic report generation, that incorporates marketing or business intelligence queries, e.g., "Return all gas turbine of a particular product line sold after 2006". We also plan to improve Optique's integration with KNIME by organising STARQL and SPARQL queries defining OBDA data sources in a tool-box of KNIME connectors in a spirit of a query catalogue. Regarding OptiqueVQS, our future work is provenance computation for query answers and computation of suggestions for query repairs, e.g., if a query returns an empty answer set, then the users would like to know why it is the case and how the query can be reformulated to obtain answers. We also plan to increase the expressiveness of OptiqueVQS gradually without compromising from the usability. This includes simpler forms of negation, disjunction, and cycles as well as ability to correlate multiple temporal-stream properties.

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), ED^3 (EP/N014359/1).

References

[1] A. Doan, A.Y. Halevy, Z.G. Ives, Principles of Data integration, Morgan Kaufmann, 2012.
[2] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, R. Rosati, Linking data to ontologies, J. Data Semant. 10 (2008) 133–173.
[3] I. Horrocks, What are ontologies good for?in: Evolution of Semantic Systems, Springer, 2013, pp. 175–188.
[4] B. Cuenca Grau, I. Horrocks, B. Motik, B. Parsia, P. Patel-Schneider, U. Sattler, OWL 2: The next step for OWL, J. Web Semant. 6 (4) (2008) 309–322.
[5] I. Horrocks, M. Giese, E. Kharlamov, A. Waaler, Using semantic technology to tame the data variety challenge, IEEE Internet Comput. 20 (6) (2016) 62–66.
[6] M. Giese, A. Soylu, G. Vega-Gorgojo, A. Waaler, P. Haase, E. Jimenez-Ruiz, D. Lanti, M. Rezk, G. Xiao, O. Ozcep, R. Rosati, Optique –zooming in on big data access, IEEE Computer 48 (3) (2015) 60–67.
[7] M. Giese, D. Calvanese, I. Horrocks, Y. Ioannidis, H. Klappi, M. Koubarakis, M. Lenzerini, R. Moller, O. Ozcep, M. Rodriguez Muro, R. Rosati, R. Schlatte, A. Soylu, A. Waaler, Scalable end-user access to big data, in: Big Data Computing, Chapman and Hall/CRC, 2013.
[8] 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, Satellite Events, 2013, pp. 125–140.
[9] D. Calvanese, M. Giese, P. Haase, I. Horrocks, T. Hubauer, Y.E. Ioannidis, E. Jiménez-Ruiz, E. Kharlamov, H. Kllapi, J.W. Klüwer, M. Koubarakis, S. Lamparter, R. Möller, C. Neuenstadt, T. Nordtveit, Ö.L. Özçep, M. Rodriguez-Muro, M. Roshchin, D.F. Savo, M. Schmidt, A. Soylu, A. Waaler, D. Zheleznyakov, Optique: Obda solution for big data, in: ESWC, Satellite Events, 2013.
[10] 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: International Semantic Web Conference, Springer, 2014, pp. 601–619.
[11] C. Neuenstadt, R. Möller, Özgür.L. Özçep, OBDA for temporal querying and streams with STARQL, in: D. Nicklas, Özgür.L. Özçep (Eds.), HiDeSt '15—Proceedings of the First Workshop on High-Level Declarative Stream Processing (co-located With KI 2015), in: CEUR Workshop Proceedings, vol. 1447, CEUR-WS.org, 2015, pp. 70–75.
[12] C. Svingos, T. Mailis, H. Kllapi, L. Stamatogiannakis, Y. Kotidis, Y. Ioannidis, Real time processing of streaming and static information, in: 2016 IEEE international Conference on Big Data, 2016.
[13] E. Kharlamov, S. Brandt, E. Jiménez-Ruiz, Y. Kotidis, S. Lamparter, T. Mailis, C. Neuenstadt, Ö.L. Özçep, C. Pinkel, C. Svingos, D. Zheleznyakov, I. Horrocks, Y.E. Ioannidis, R. Möller, Ontology-Based integration of streaming and static relational data with optique, in: ACM SIGMOD, 2016, pp. 2109–2112.
[14] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, R. Rosati, Tractable reasoning and efficient query answering in description Logics: The DL-Lite family, J. Autom. Reason. 39 (3) (2007) 385–429.
[15] 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.
[16] F. Priyatna, O. Corcho, J. Sequeda, Formalisation and experiences of R2RML-based SPARQL to SQL query translation using morph, in: WWW, 2014.
[17] M. Rodriguez-Muro, R. Kontchakov, M. Zakharyaschev, Obda with ontop, in: ORE, 2013, pp. 101–106.
[18] C. Bizer, A. Seaborne, D2RQ-Treating non-RDF databases as virtual RDF graphs, in: ISWC, 2004.
[19] 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.
[20] J.F. Sequeda, D.P. Miranker, Ultrawrap: SPARQL execution on relational data, J. Web Sem. 22 (0).
[21] J. Calbimonte, Enabling ontology-based access to streaming data sources, in: ISWC, 2010.
[22] L. Fischer, T. Scharrenbach, A. Bernstein, Scalable linked data stream processing via network-aware workload scheduling, in: Proceedings of the 9th International Workshop on Scalable Semantic Web Knowledge Base Systems, Sydney, Australia, October 21, 2013, 2013, pp. 81–96.
[23] A. Artale, R. Kontchakov, F. Wolter, M. Zakharyaschev, Temporal description logic for ontology-based data access, in: IJCAI, IJCAI'13, 2013, pp. 711–717.
[24] S. Borgwardt, M. Lippmann, V. Thost, Temporal query answering in the description logic dl-lite, in: FroCoS, 2013, pp. 165–180.
[25] 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.
[26] D.F. Barbieri, D. Braga, S. Ceri, E.D. Valle, M. Grossniklaus, C-sparql: a continuous query language for rdf data streams, Int. J. Semant. Comput. 4 (1) (2010) 3–25.
[27] J.-P. Calbimonte, O. Corcho, A.J.G. Gray, Enabling Ontology-Based Access to Streaming Data Sources, in: ISWC, ISWC'10, 2010, pp. 96–111.

Please cite this article in press as: E. Kharlamov, et al., Semantic access to streaming and static data at Siemens, Web Semantics: Science, Services and Agents on the World Wide Web (2017), http://dx.doi.org/10.1016/j.websem.2017.02.001.

[28] 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.
[29] 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.
[30] 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. http://dx.doi.org/10.1145/1963405.1963495.
[31] D. Anicic, S. Rudolph, P. Fodor, N. Stojanovic, Stream reasoning and complex event processing in etalis, Semantic Web 3 (4) (2012) 397–407.
[32] 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.
[33] Y. Zhang, P. Minh Duc, O. Corcho, J.P. Calbimonte, Srbench: A Streaming RDF/SPARQL Benchmark, in: Proceedings of international Semantic Web Conference 2012, 2012.
[34] Özgür L. Özçep, R. Möller, C. Neuenstadt, D. Zheleznyakov, E. Kharlamov, Deliverable D5.1 –a semantics for temporal and stream-based query answering in an OBDA context, Deliverable FP7-318338, EU, October 2013.
[35] 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, in: Lecture Notes in Computer Science, vol. 8714, 2014.
[36] Özgür.L. Özçep, R. Möller, C. Neuenstadt, A stream-temporal query language for ontology based data access, in: KI 2014, in: LNCS, vol. 8736, Springer International Publishing Switzerland, 2014, pp. 183–194.
[37] Özgür.L. Özçep, R. Möller, C. Neuenstadt, Stream-query compilation with ontologies, in: B. Pfahringer, J. Renz (Eds.), Proceedings of the 28th Australasian Joint Conference on Artificial intelligence 2015, AI 2015, in: LNAI, vol. 9457, Springer International Publishing, 2015.
[38] A. Arasu, S. Babu, J. Widom, The cql continuous query language: semantic foundations and query execution, VLDB J. 15 (2006) 121–142.
[39] J.-P. Calbimonte, Ontology-Based Access to Sensor Data Streams, (Dissertation), Universidad Politecninca de Madrid, 2013.
[40] 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.
[41] 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.
[42] 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.
[43] 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, vol. 9982, 2016, pp. 344–362.
[44] 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: ACM DEBS, 2016, pp. 350–353.
[45] 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.
[46] H. Kllapi, P. Sakkos, A. Delis, D. Gunopulos, Y. Ioannidis, Elastic processing of analytical query workloads on iaas clouds, arXiv preprint arXiv:1501.01070.
[47] N. Giatrakos, Y. Kotidis, A. Deligiannakis, V. Vassalos, Y. Theodoridis, In-network approximate computation of outliers with quality guarantees, Inf. Syst. 38 (8) (2013) 1285–1308.
[48] M.G. Skjæveland, E.H. Lian, I. Horrocks, Publishing the norwegian petroleum directorate's factpages as semantic web data, in: ISWC, 2013, pp. 162–177.
[49] 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.
[50] 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, Scalable semantic access to siemens static and streaming distributed data, in: ISWC, Posters & Demonstrations, 2016.
[51] 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, Posters & Demonstrations, 2015.
[52] 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, Posters & Demonstrations, 2015.
[53] 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's factpages, in: ISWC, Posters & Demos, 2013.
[54] P. Haase, I. Horrocks, D. Hovland, T. Hubauer, E. Jiménez-Ruiz, E. Kharlamov, J.W. Klüwer, C. Pinkel, R. Rosati, V. Santarelli, A. Soylu, D. Zheleznyakov, Optique system: towards ontology and mapping management in obda solutions, in: WoDOOM, 2013, pp. 21–32.
[55] 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.
[56] 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, 2015.
[57] 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.
[58] C. Pinkel, C. Binnig, E. Jiménez-Ruiz, E. Kharlamov, W. May, A. Nikolov, M.G. Skjæveland, A. Solimando, M. Taheriyan, C. Heupel, I. Horrocks, RODI: Benchmarking relational-to-ontology mapping generation quality, in: Semantic Web –Interoperability, Usability, Applicability, 2017.
[59] D. Zheleznyakov, E. Kharlamov, V. Klungre, M.G. Skjæveland, D. Hovland, M. Giese, I. Horrocks, A. Waaler, Keywdb: A system for keyword-driven ontology-to-rdb mapping construction, in: ISWC, Posters & Demonstrations, 2016.
[60] D. Calvanese, I. Horrocks, E. Jiménez-Ruiz, E. Kharlamov, M. Meier, M. Rodriguez-Muro, D. Zheleznyakov, On rewriting, answering queries in obda systems for big data, in: OWLED, 2013.
[61] P. Haase, C. Hütter, M. Schmidt, A. Schwarte, The Information Workbench as a Self-Service Platform for Linked Data Applications, in: WWW, 2012.
[62] A. Soylu, M. Giese, E. Jimenez-Ruiz, G. Vega-Gorgojo, I. Horrocks, Experiencing OptiqueVQS –a multi-paradigm and ontology-based visual query system for end-users, Univers. Access Inform. Soc. 15 (1) (2016) 129–152.
[63] A. Soylu, E. Kharlamov, D. Zheleznyakov, E. Jimenez 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 (2017) (in press).
[64] A. Soylu, M. Giese, R. Schlatte, E. Jimenez-Ruiz, E. Kharlamov, O. Ozcep, C. Neuenstadt, S. Brandt, Querying industrial stream-temporal data: an ontology-based visual approach, J. Ambient Intell. Smart Environ. 9 (1) (2017) 77–95.
[65] A. Soylu, E. Kharlamov, D. Zheleznyakov, E. Jimenez-Ruiz, M. Giese, I. Horrocks, Ontology-based visual query formulation: An industry experience, in: Proceedings of the 11th international Symposium on Visual Computing, ISVC 2015, in: LNCS, vol. 9474, Springer, 2015, pp. 842–854.
[66] A. Soylu, M. Giese, E. Jiménez-Ruiz, E. Kharlamov, D. Zheleznyakov, I. Horrocks, Why not simply google?in: Proceedings of the 8th Nordic Conference on Human-Computer interaction, NordiCHI 2014, ACM, 2014, pp. 1039–1042.
[67] A. Soylu, M. Giese, E. Jimenez-Ruiz, E. Kharlamov, D. Zheleznyakov, I. Horrocks, OptiqueVQS –towards an ontology-based visual query system for big data, in: MEDES, 2013.
[68] A. Soylu, M. Giese, R. Schlatte, E. Jimenez-Ruiz, O. Ozcep, S. Brandt, Domain experts surfing on stream sensor data over ontologies, in: Proceedings of the 1st international Workshop on Semantic Web Technologies for Mobile and Pervasive Environments, SEMPER 2016, in: CEUR Workshop Proceedings, vol. 1588, CEUR-WS.org, 2016.
[69] A. Soylu, M. Giese, R. Schlatte, Özgür Özçep, S. Brandt, A visual query system for stream data access over ontologies, in: Proceedings of the Satellite Events of the 13th European Conference on the Semantic Web, ESWC 2016, in: LNCS, vol. 9989, Springer, 2016.
[70] A. Soylu, F. Moedritscher, F. Wild, P. De Causmaecker, P. Desmet, Mashups by orchestration and widget-based personal environments: Key challenges, solution strategies, and an application, Program: Electron. Libr. Inform. Syst. 46 (4) (2012) 383–428.
[71] T. Catarci, M.F. Costabile, S. Levialdi, C. Batini, Visual query systems for databases: A survey, J. Vis. Lang. Comput. 8 (2) (1997) 215–260. http://dx.doi.org/10.1006/jvlc.1997.0037.
[72] M. Arenas, B. Cuenca Grau, E. Kharlamov, Š Marciuška, D. Zheleznyakov, Faceted search over RDF-based knowledge graphs, Web Semant. Sci. Serv. Agents World Wide Web 37–38 (2016) 55–74.
[73] B.C. Grau, E. Kharlamov, S. Marciuska, D. Zheleznyakov, M. Arenas, Semfacet: Faceted search over ontology enhanced knowledge graphs, in: ISWC, Posters & Demonstrations, 2016.
[74] B.C. Grau, E. Kharlamov, S. Marciuska, D. Zheleznyakov, Y. Zhou, Querying life science ontologies with semfacet, in: Proceedings of the 7th International Workshop on Semantic Web Applications and Tools for Life Sciences, Berlin, Germany, December 9–11, 2014, 2014.

Please cite this article in press as: E. Kharlamov, et al., Semantic access to streaming and static data at Siemens, Web Semantics: Science, Services and Agents on the World Wide Web (2017), http://dx.doi.org/10.1016/j.websem.2017.02.001.

[75] M. Arenas, B.C. Grau, E. Kharlamov, S. Marciuska, D. Zheleznyakov, Faceted search over ontology-enhanced RDF data, in: CIKM, 2014, pp. 939–948.
[76] B.C. Grau, E. Kharlamov, D. Zheleznyakov, M. Arenas, S. Marciuska, On faceted search over knowledge bases, in: Informal Proceedings of the 27th International Workshop on Description Logics, Vienna, Austria, July 17–20, 2014, 2014, pp. 153–156.
[77] M. Arenas, B.C. Grau, E. Kharlamov, S. Marciuska, D. Zheleznyakov, E. Jiménez-Ruiz, Semfacet: semantic faceted search over yago, in: 23rd International World Wide Web Conference, WWW '14, Seoul, Republic of Korea, April 7–11, 2014, Companion Volume, 2014, pp. 123–126.
[78] M. Arenas, B.C. Grau, E. Kharlamov, S. Marciuska, D. Zheleznyakov, Towards semantic faceted search, in: 23rd International World Wide Web Conference, WWW '14, Seoul, Republic of Korea, April 7–11, 2014, Companion Volume, 2014, pp. 219–220.
[79] M. Arenas, B.C. Grau, E. Kharlamov, S. Marciuska, D. Zheleznyakov, Enabling faceted search over OWL 2 with semfacet, in: Proceedings of the 11th International Workshop on OWL: Experiences and Directions, OWLED 2014, co-located with 13th International Semantic Web Conference on, ISWC 2014, Riva del Garda, Italy, October 17–18, 2014, 2014, pp. 121–132.
[80] B. Glimm, I. Horrocks, B. Motik, G. Stoilos, Z. Wang, Hermit: An OWL 2 reasoner, JAR 53 (3) (2014) 245–269.
[81] A. Soylu, M. Giese, E. Jiménez-Ruiz, E. Kharlamov, D. Zheleznyakov, I. Horrocks, Towards exploiting query history for adaptive ontology-based visual query formulation, in: MTSR, 2014, pp. 107–119.
[82] E. Kharlamov, B.C. Grau, E. Jiménez-Ruiz, S. Lamparter, G. Mehdi, M. Ringsquandl, Y. Nenov, S. Grimm, M. Roshchin, I. Horrocks, Capturing industrial information models with ontologies and constraints, in: ISWC, 2016, pp. 325–343.
[83] E. Kharlamov, B.C. Grau, E. Jiménez-Ruiz, S. Lamparter, G. Mehdi, M. Ringsquandl, Y. Nenov, S. Grimm, M. Roshchin, I. Horrocks, SOMM: industry oriented ontology management tool, in: ISWC, Posters & Demonstrations, 2016.
[84] M. Compton, P.M. Barnaghi, L. Bermudez, R. Garcia-Castro, Ó Corcho, S. Cox, J. Graybeal, M. Hauswirth, C.A. Henson, A. Herzog, V.A. Huang, K. Janowicz, W.D. Kelsey, D.L. Phuoc, L. Lefort, M. Leggieri, H. Neuhaus, A. Nikolov, K.R. Page, A. Passant, A.P. Sheth, K. Taylor, The SSN ontology of the W3C semantic sensor network incubator group, J. Web Sem. 17 (2012) 25–32.
[85] A. Soylu, M. Giese, Qualifying ontology-based visual query formulation, in: Proceedings of the 11th international Conference Flexible Query Answering Systems, FQAS 2015, in: Advances in Intelligent Systems and Computing, vol. 400, Springer, 2015, pp. 243–255.
[86] A. Soylu, M. Giese, E. Kharlamov, E. Jimenez-Ruiz, D. Zheleznyakov, I. Horrocks, Ontology-based end-user visual query formulation: Why, what, who, how, and which?, Universal Access in the Information Society (2016). http://dx.doi.org/10.1007/s10209-016-0465-0 (in press).
[87] A. Katifori, C. Halatsis, G. Lepouras, C. Vassilakis, E. Giannopoulou, Ontology visualization methods - a survey, ACM Comput. Surv. 39 (4) (2007) 10:1–10:43.

Please cite this article in press as: E. Kharlamov, et al., Semantic access to streaming and static data at Siemens, Web Semantics: Science, Services and Agents on the World Wide Web (2017), http://dx.doi.org/10.1016/j.websem.2017.02.001.