A Semantic Approach to Polystores

E. Kharlamov*, T. Mailis§, K. Bereta§, D. Bilidas§, S. Brandt†, E. Jimenez-Ruiz*, S. Lamparter†,
C. Neuenstadt¶, Ö. Özçep¶, A. Soylu‖, C. Svingos§, G. Xiao**, D. Zheleznyakov*,
D. Calvanese**, I. Horrocks*, M. Giese††, Y. Ioannidis§, Y. Kotidis‡, R. Möller¶, A. Waaler††

*University of Oxford †Siemens CT ‡Athens University of Economics and Business §University of Athens
¶University of Lübeck ‖NTNU **Free University of Bozen-Bolzano ††University of Oslo

***Abstract*—In the database community Polystores is an emerging and promising approach for data federation that aims at designing a unified querying layer over multiple data models. In the Semantic Web community a similar in spirit approach of Ontology-Based Data Access (OBDA) has been recently proposed, attracted a lot of attention, and proved its success in several industrial scenarios. In this paper we discuss a semantic approach to building polystores using the OBDA paradigm. We also present our system Optique that is utilized in an industrial application of performing turbine diagnostics in Siemens.**

I. Introduction

It is common that large companies nowadays possess many types of databases, data, and storage models. Developing applications that work across these different modalities is often limited by the incompatibility of systems or the difficulty of creating new connectors and translators between them [16], [14], [12]. For instance, performing diagnostics of turbines requires Siemens engineers to query and analyze sensor streaming data, static data about equipment's structure, history of its exploitation and repairs, and even weather conditions. Analyzing such complex and heterogeneous data requires a new generation of federated databases that support seamless access to the different database management systems or storage engines used in the back-ends. A new breed of systems that provide this functionality have been recently introduced in the database community under the name of *polystores* in order to distinguish them from traditional federated databases that largely supported access to multiple engines using the same data model [16].

In the Semantic Web community an approach to data federation similar in spirit to polystores, called *Ontology-Based Data Access* has been recently proposed, attracted a lot of attention, and proved its success in several industrial scenarios [14], [12], [8]. The key concept of Ontology Based Data Access (OBDA) is to use an ontology, i.e. a formal conceptualisation of the application domain, to mediate access to relational and non-relational data-sources. The proposed methodology *(i)* offers a semantic view on the application data domain, *(ii)* while allowing to infer implicit information, via a reasoning procedure. The traditional OBDA approach assumes that users formulate their information needs as queries using terms defined in the ontology, and these are then translated into some database query languages and executed over the data automatically, without an IT expert's intervention. To this end a set of *mappings* is maintained that describes the relationship between the ontological vocabulary and the schema of the data. In the course of the Optique project we had to extend the traditional OBDA definition in order to tackle our versatile use-cases:

Use Case

Our approach is motivated by industrial scenarios that demand for real-time processing of streaming and static data. *Siemens* runs service centres dedicated to diagnostics of thousands of power-generation appliances across the globe. One typical task for such centres is to detect in real-time potential faults caused by, e.g., an abnormal temperature and pressure increase. Such tasks require simultaneous processing of sequences of digitally encoded coherent signals produced and transmitted from thousands of power generating turbines, generators, and compressors installed in power plants, and of static data that include the structure of relevant equipment, history of its exploitation and repairs, and even weather conditions. As illustrated in Figure 1, these data are scattered across a large number of heterogeneous data streams in addition to static DBs with hundreds of TBs of data.

Even for a single diagnostic task, such as checking if a given turbine might develop a fault, Siemens engineers have to analyse streams with temperature and other measurements from up to $2,000$ sensors installed in different parts of the turbine, analyse historical temperature data, compute temperature patterns, compare them to patterns in other turbines, compare weather conditions, etc. This requires to pose a collection of hundreds of queries, the majority of which are semantically the same (they ask about temperature), but syntactically different (they are over different schemata). Formulating and executing so many queries, and then assembling the computed answers, takes up to 80% of the overall diagnostic time [12].

A different type of processing, relates to creating a virtual ontology on top of large relational databases that contain geometries and get frequently updated. Such an extension has been motivated by the Statoil use case [14] in the context of the project Optique. The Statoil use case demands for spatial predicates that are used by geologists in order to combine information from wellbores, seismic investigations, and general geological knowledge to assess, for example, what types of rock are in each wellbore.

Several Species of OBDA

In order to manage such diagnostic processes, we had to extend existing OBDA solutions in order to handle streaming and geospatial information. The OPTIQUE platform provides for three different user-facing abstractions consisting of a *data model*, a *query language*, and *mappings* to translate each data model to the local dialect supported by each local end-point. The three different user-facing abstractions rely on different semantics in order capture *(i)* standard ontology relations as well as *(ii)* their streaming and *(iii)* geospatial extensions.

- In each approach the data model is an *ontology*: a formal conceptualisation of the domain of interest that consists of a *vocabulary*, i.e., names of classes, attributes and binary relations, and *axioms* over the terms from the vocabulary that, e.g., assign attributes of classes, define relationships between classes, compose classes, class hierarchies, etc. The Siemens ontology that we developed [12] encodes generic specifications of appliances, characteristics of sensors, materials, processes, descriptions of diagnostic tasks, etc.
- In order to model our data to the local dialect we use mappings from the underlying language to ontology assertions. These mappings relate each ontological term to a set of queries over the underlying data. For example, the generic attribute *temperature-of-sensor* from the Siemens ontology is mapped to all specific data and procedures that return temperature readings from sensors in dozens of different turbines and DBs storing historical data, thus, all particularities and varieties of how the temperature of a sensor can be measured, represented, and stored are captured in these mappings.
- The integrated data can be accessed by posing queries over the ontology, i.e., *ontological queries*. These queries, depending on the user-facing abstraction are either standard ontological queries, *hybrid* queries that refer to both streaming and static data, and geospatial queries. Evaluation of such queries has three stages: *(i)* in the *enrichment* stage ontology axioms are used to expand the ontological query in order to access as much of relevant data as possible; *(ii)* in the *unfolding* stage the mappings are used to translate the enriched ontological query into (possibly many) queries over the data; and *(iii)* in the *execution* stage the unfolded queries are executed over the data.

The main benefit of our approach is that the combination of ontologies and mappings provide for location transparency by allowing 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 the Siemens diagnostic task above using 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 enrichment and unfolding stages of the evaluation by an OBDA system will automatically compute it from the high-level ontological query. Another important benefit of the approach

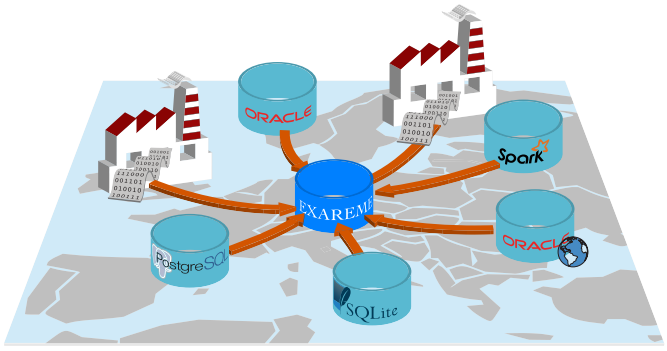

Fig. 1. Streaming and static processing in the Siemens use case.

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.

It should be noted, that the OPTIQUE platform, additionally to OBDA, provides a relational user-facing abstraction for static and stream processing via the system's federation hubs. The two hubs, EXAREME and EXASTREAM, process information based on SQL language extensions with user defined functions. The latter allow to express complex dataflows, such as data mining and data analytics tasks, that cannot be captured in standard OBDA because of the strict semantics of ontologies.

Research Challenges

The benefits of OBDA come at a price. The main practical challenges that are not addressed by existing Semantic Technologies include:

[C1] development of tools for semi-automatic support to construct high quality ontologies and mappings over relational and streaming data;

[C2] development of a query language over ontologies that combines streaming and static data, and allows for efficient enrichment and unfolding that preserves the semantics of ontological queries;

[C3] development of a geospatial extension of the OBDA approach, that leverages the technologies of geospatial databases into ontological query processing.

[C4] development of a backend that can optimise large numbers of queries automatically generated via enrichment and unfolding, and efficiently execute them over distributed streaming and static data; and

[C5] development of the corresponding visualisations tools that will aid non-experts in formulating complex queries and accessing their corresponding results.

Construction of ontologies and mappings in OBDA is done independently and prior to query formulation and processing. Nevertheless, addressing C1 is practically important since such tools can dramatically speed up deployment and maintenance

(e.g., adjustment to new query requirements) of OBDA systems. Addressing C2 is crucial since, to the best of our knowledge, no dedicated query language for streaming-static semantic queries has the required properties. Addressing C3 is of primary importance in order to ensure that geospatial-semantic queries can be adopted to access information residing in geospatial databases. Addressing C4 is vital to ensure that OBDA queries are executable in reasonable time. Note that C4 is not trivial since even in the context where the data is only static and not distributed, query execution without dedicated optimisation techniques performs poorly since the queries that are automatically computed after enrichment and unfolding can be very inefficient, e.g., they may contain many redundant joins and unions [5]. Challenging C5 is of primary importance in a working environment, that non-experts need to access complicated data-sources without any knowledge of the underlying schema.

Besides proposing OBDA, we addressed the challenges C1-C5 and implemented our solutions in the OPTIQUE [13] system that has been successfully applied in several industrial contexts [14], [12], [31], [**?**]. In Section II we provide an overview of the OPTIQUE platform and illustrate the processes that take place during a query formulation/execution/analysis cycle. Section III presents BOOTOX [11], [4], a system for "bootstrapping" OBDA assets by extracting, ontologies and mappings from static and streaming relational schema and data. Section IV presents the visual components of the OPTIQUE platform. OPTIQUEVQS is a visual query formulation system that allows domain experts to formulate and pose queries. The answers to these query are returned by a flexible wiki-based Diagnostic Dashboard that can be easily customised by end users themselves. Section V introduces the distinct OBDA systems that are responsible for mapping semantic queries over OWL 2 QL ontologies (and their streaming and geospatial extensions), to the underlying data via global-as-view mappings [10]. It also presents STARQL [25], a query language that allows for semantic queries over both streaming and static data. Section VI, presents EXAREME [18], [23], a highly optimised database engine, and EXASTREAM, its streaming counterpart capable of handling complex streaming-static queries in real time. EXAREME supports parallel query execution and its Infrastructure as a Service architecture enables us to elastically scale the system to support user-demand in complex diagnostic scenarios.

II. SYSTEM OVERVIEW & RELATION TO POLYSTORES

OPTIQUE is an integrated system that consists of multiple components to support OBDA end-to-end [13], [15], [**?**]. For IT specialists OPTIQUE offers support for the whole lifecycle of ontologies and mappings: semi-automatic bootstrapping from relational data sources, importing of existing ontologies, semi-automatic quality verification and optimisation, cataloging, manual definition and editing of mappings. For end-users OPTIQUE offers tools for query formulation support, query cataloging, answer monitoring, as well as integration with GIS systems. Query evaluation is done via OPTIQUE's

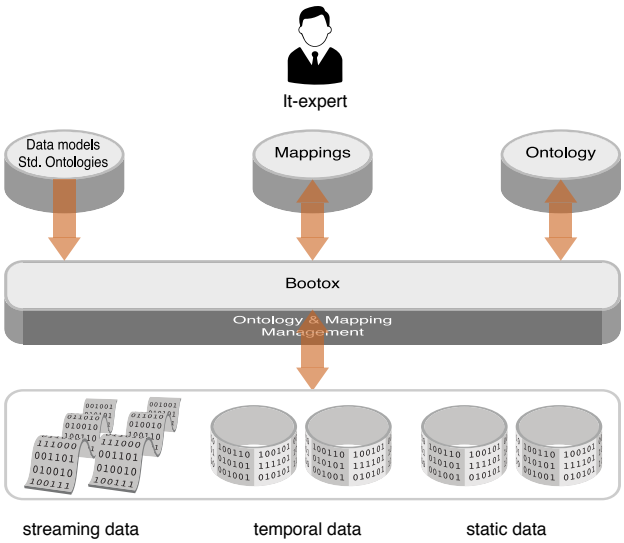

Fig. 2. Bootstrapping Ontologies

query enrichment, unfolding, and execution backends that allow to execute complex ontological queries in highly distributed environments.

In this section we give some details of the OPTIQUE components that address the C1-C5 challenges above.

For the system deployment, one can bootstrap ontologies and mappings from the underlying relational data sources, incorporate external ontologies into the system, and edit ontologies and mappings [19]. This is performed by the BOOTOX component as illustrated in Figure 2.

The query formulation, transformation, execution, and answer visualisation procedures are performed in a sequence of stages presented in Figure 3:

- 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, depending on their type, are internally translated by OPTIQUEVQS to: *(i)* SPARQL expressions, i.e. queries designed to retrieve information from standard ontologies; *(ii)* STARQL expressions, i.e. queries designed to retrieve information from streaming ontologies; *(iii)* GeoSPARQL expressions, i.e. queries designed to retrieve information from geospatial ontologies.
- 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 [7]. *(i)* SPARQL expressions are rewritten and unfolded to SQL queries by the ONTOP query transformation engine [27]; *(ii)* STARQL expressions are rewritten and unfolded to SQL$^{\oplus}$ queries by the STARQL2SQL$^{\oplus}$ query

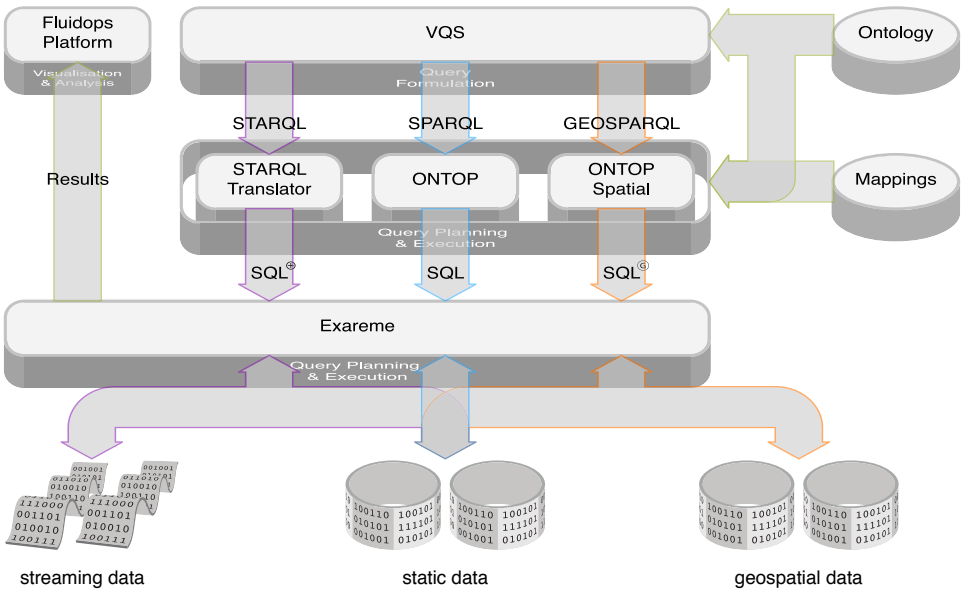

Fig. 3. Querying heterogeneous Data Sources

transformation engine[25]. SQL$^{\oplus}$ extends standard SQL with operators for stream handling; *(iii)* GeoSPARQL expressions are rewritten and unfolded to SQL queries by the ONTOP-SPATIAL query transformation engine[25]. The corresponding SQL queries also contain geospatial operators.

- Query Execution: SQL queries (possibly containing geospatial operators) are executed by EXAREME, a system for large scale elastic data processing on the cloud. EXAREME federates information from the underlying endpoints and examines the possibility to push specific query fragments whenever it is beneficial for the overall query execution time. SQL$^{\oplus}$ streaming queries are executed by EXASTREAM, i.e. EXAREME's streaming counterpart that uses parallelism to cope with the huge data sets provided by Siemens.
- Visualisation & Analysis: Resulting query answers are visualised using templates and widgets such as tables, timelines, maps, charts, etc., depending on the data modalities. The Optique platform implementation is based on the Information Workbench (IWB) [20], a generic and extensible platform for semantic data management which provides a rich infrastructure for platform.

The OPTIQUE platform shares many common characteristics with Polystores. The different OBDA variants that allow to federate information from relational data-sources via a query language, an ontology, and mappings can be matched to *islands of information* in Polystore terminology consisting of a query language, a data model, and shims to translate island utterances into the local dialect supported by each storage system. OPTIQUE also preserves the location transparency prerequisite, ensuring that the same answer is produced to any given query, even though the data may reside in perhaps multiple federation end-points. OPTIQUE federates relevant information from underlying data sources without altering the initial information and without suppressing local database functionality, while pushing specific query fragments for execution to relational endpoints, whenever it is beneficial.

We now consider the different aspects of the OPTIQUE platform:

III. DEPLOYMENT SUPPORT

The BOOTOX component extracts W3C standardised OWL 2 ontologies and R2RML mappings from relational streaming and static data. Consider, e.g., a class *Turbine*; a mapping for it is an expression: $\mathit{Turbine}(f(\vec{x})) \leftarrow \exists \vec{y}\ \mathtt{SQL}(\vec{x}, \vec{y})$, that can be seen as a view definition, where $\mathtt{SQL}(\vec{x}, \vec{y})$ is an SQL query, $\vec{x}$ are its output variables, $\vec{y}$ are its variables that are projected out and f is a function that converts tuples returned by `SQL` into identifiers of objects populating the class *Turbine*. Intuitively, mapping bootstrapping of BOOTOX boils down to discovery of 'meaningful' queries $\exists \vec{y}\ \mathtt{SQL}(\vec{x}, \vec{y})$ over the input data sources that would correspond to either a given element of the ontological vocabulary, e.g., the class *Turbine* or attribute *temperature-of-sensor*, or to a new ontological term. BOOTOX employs several novel schema- and data-driven query discovery techniques. E.g., BOOTOX can map two tables like `Turbine` and `Country` into classes by projecting them on primary keys, and the attribute `locatedIn` of `Turbine` into an object property between these two classes if there is either an explicit or implicit foreign key between `Turbine` and `Country`. For more complex mappings, BOOTOX requires users to provide a set of examples of entities from the class, e.g., *Turbine*, where each example is a set of keywords, e.g., $\{\mathit{albatros}, \mathit{gas}, \mathit{2008}\}$. Then the system turns these keywords into SQL queries by exploiting graph-based techniques similar to [21] for keyword-based query answering over DBs. Moreover, BOOTOX also allows us

to incorporate third party OWL 2 ontologies in an existing OPTIQUE's deployment using ontology alignment techniques.

The ontological terms bootstrapped by means of BOOTOX provide the vocabulary for the formulation of STARQL ontological queries and the bootstrapped mappings. In the following we will discuss STARQL queries and how we process them.

IV. VISUAL QUERY SYSTEM & DIAGNOSTICS DASHBOARD

A. Visual Query System

Most diagnostic engineers cannot be expected to learn formal query languages like SPARQL, STARQL, and GEOSPARQL. The Optique platform therefore contains a visual query system [**?**], called OptiqueVQS [31], [29], [30], that makes it easy for users without IT background to formulate the most commonly needed queries.

OptiqueVQS is widget-based and supports tree-shaped conjunctive queries [29]. In Figure 4, an example query is shown as a tree, representing typed variables as nodes and object properties as arcs. Typed variables can be added to the query by using the concept-object property list. If a query node is selected, a faceted widget displays controls for refining the corresponding typed variable, e.g. constraining a data property. Once a restriction is set on a data property or a data property is selected for output, it is reflected in the label of the corresponding node in the query graph. The user can always jump to a specific part of the query by clicking on the corresponding variable-node in the query diagram.

Dynamic properties (i.e., whose extensions are time dependent) are colored in blue and as soon as one is selected OptiqueVQS switches to STARQL mode. A stream button appears on top of the query diagram and lets the user configure parameters such as slide and window width interval. If the user clicks on the "Result Overview" button, a template selection widget appears for selecting a template for each stream attribute, which is by default "echo". The user can register the query in by clicking on the "Register query" button.

The STARQL language is very expressive, but as Listing 1 demonstrates, a simple idea like a value increasing throughout a time interval may require a query with several quantifiers – a far too demanding formalisation task for the average user. Thus OptiqueVQS splits the formulation of streaming queries into two steps where the user has to specify *(i)* which data streams are of interest, this corresponds to the static part in the WHERE clause, and *(ii)* 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.

For handling GEOSPARQL queries, OPTIQUEVQS does not consider any extended features. This is because geospatial relations are expressed with the use of ordinary object and datatype properties.

Fig. 4. Visual Query System

Fig. 5. Components of Diagnostic Dashboard

B. Diagnostics Dashboard

In order to address diverse needs of end users in query formulation and answer visualisation we developed a flexible wiki-based *Diagnostic Dashboard* that can be easily customised by end users themselves. Result visualisation widgets allow to visualise query answers, inspect query results, do incremental query refinement, and export of relevant result fragments to external diagnostic tools. Moreover, the widgets allow to perform monitoring of incoming data streams and query answers for continuos queries over these streams. In Figure 5 we present three examples of our visualisation widgets. 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.

V. QUERY TRANSFORMATION

This Section is dedicated to the enrichment and unfolding stages that occur during a query execution cycle. As illustrated in Section II, depending on the type of the semantic query, a different component is responsible for making the translation to the appropriate formalism. ONTOP is dedicated to transforming SPARQL queries to SQL queries, the

STARQL2SQL$^{\oplus}$ component transforms STARQL streaming queries to SQL$^{\oplus}$ queries, while ONTOP-SPATIAL transforms GeoSPARQL queries to SQL queries containing geospatial operators.

A. ONTOP

ONTOP is an open-source OBDA system released under the Apache license, developed at the Free University of Bozen-Bolzano [5], [22]. The ONTOP system exposes relational databases as virtual RDF graphs by linking the terms (classes and properties) in the ontology to the data sources through mappings. This virtual RDF graph can then be queried using SPARQL, by translating the SPARQL queries into SQL queries over the relational databases. This translation process is transparent to the user.

ONTOP allows for RDFS [3] and OWL 2 QL [24] as ontology languages. OWL 2 QL is based on the DL-Lite family of lightweight description logics [6], which guarantees that queries over the ontology can be rewritten into equivalent queries over the databases. ONTOP supports the W3C RDB2RDF Mapping Language (R2RML), which is a widely used standard. Intuitively, a mapping assertion consists of a source (an SQL query retrieving values from the database) and a target (defining RDF triples with values from the source). For querying ontologies, ONTOP supports essentially all features of SPARQL 1.0 and the OWL 2 QL entailment regime of SPARQL 1.1. The core of ONTOP is the SPARQL engine Quest, which is in charge of rewriting SPARQL queries over the virtual RDF graph into SQL queries over the relational database. It does so by first *enriching* the SPARQL query using the axioms in the ontology, and then *unfolding* the resulting query by means of the mapping.

Example 1: In the corresponding example, we provide an overview of the enrichment and unfolding stages. The following ontology captures a fragment of the domain knowledge of our running example. It describes the concepts of a turbine with the following OWL axioms:

```
:GasTurbine rdfs:subClassOf :Turbine.
:DieselTurbine rdfs:subClassOf :Turbine.
```

These axioms simply say that if an individual belongs to the class `GasTurbine` or the class `DieselTurbine`, then it belongs to the class `:Turbine`. I.e. each diesel or gas turbine is also a turbine. The ontology instance assertions in this example can be populated from a database by means of the following mappings:

```
:{gtid} rdf:type :GasTurbine.
  ⟵ SELECT gtid FROM tbl_Gas_Turbines
:{dtid} rdf:type :DieselTurbine.
  ⟵ SELECT dtid FROM tbl_Diesel_Turbines.
```

The simple SPARQL query asking for turbines:

```
SELECT ?s WHERE {?s rdf:type :Turbine.}
```

first will be enriched using the ontology axioms so as to ask for gas and for diesel turbines, and then will be unfolded into the following SQL query, which performs the appropriate union between the relevant tables in the relational database:

```
SELECT concat(":", T1.gtid) AS s
FROM tbl_Gas_Turbines T1
  UNION
SELECT concat(":", T2.dtid) AS s
FROM tbl_Diesel_Turbines T2
```

B. STARQL2SQL$^{\oplus}$ *Translator*

In order to express semantic queries blending streaming with static data we developed the STARQL query language [25].

The syntax of STARQL extends so-called *basic graph patterns* of W3C standardised SPARQL query language for RDF databases. STARQL queries can express basic graph patterns, and typical mathematical, statistical, and event pattern features needed in real-time diagnostic scenarios. Moreover, STARQL queries can be nested, in the sense that the result of one query may be used in the construction of another query. STARQL has a formal semantics that combines open and closed-world reasoning and extends snapshot semantics for window operators [1] with sequencing semantics that can handle integrity constraints such as functionality assertions.

Example 2: Due to the space limitation we cannot present STARQL in details. Instead, we will illustrate its main features on the following example diagnostic task: *Detect a real-time fault in a turbine caused by a temperature increase within 10 seconds.* This task can be expressed in STARQL over the Siemens ontology [12] as in Listing 1 and it requires to combine streaming and static data. An output stream `S_out` is defined by the following language constructs: The `CONSTRUCT` specifies the format of the output stream, here instantiated by RDF triples asserting that there was a monotonic increase. The `FROM` clause specifies the resources on which the query is evaluated: the `ONTOLOGY`, `STATIC DATA`, and input `STREAM`(s), for which a window operator is specified with window range (here 10 sec) and with slide (here 1 sec). The `PULSE` declaration specifies the output frequency. In the `WHERE` clause, bindings for sensors (attached to the assembly structure of the turbine) are chosen. For every binding, the relevant condition of the diagnostic task is tested on the window contents. Here this condition is abbreviated by `MONOTONIC.HAVING(seq, ?c, sie:hasValue)` using a macro that is defined at the bottom of Fig. 1 in an `AGGREGATE` declaration. In words, the conditions asks whether there is some state `?k` in the window s.t. the sensor shows a failure message at `?k` and s.t. for all states before `?k` the attribute value `?attr` (in the example instantiated by `sie:hasValue`) is monotonically increasing.

STARQL has favourable computational properties [25]: despite its expressivity, answering STARQL queries is efficient since they can be efficiently enriched and then unfolded into efficient relational stream queries. STARQL query enrichment is polynomial-time in the size of the input ontology if the ontology is expressed in the OWL 2 QL ontology language and the queries are essentially conjunctive with value comparison and aggregates. STARQL unfolding is linear-time in the size of both mappings and query and enriched STARQL

Listing 1. An example diagnostic task in STARQL, where the prefix *sie* stands for the URI of the Siemens ontology

```
CREATE STREAM  Str_out AS
CONSTRUCT      GRAPH NOW { ?c2 rdf:type :MonInc }
FROM           STREAM Str_Msmt [NOW-"PT10S"^^xsd:duration, NOW]->
                              "PT1S"^^xsd:duration,
               STATIC DATA sie:Static,
               ONTOLOGY sie:Ontology
USING          PULSE WITH START = "00:10:00CET", FREQUENCY = "1S"
WHERE          {?c1 a sie:Assembly. ?c2 a sie:Sensor.
               ?c1 sie:inAssembly ?c2.}
SEQUENCE BY    StandardSequencing AS seq
HAVING         MONOTONIC.HAVING(seq, ?c2, sie:hasValue)

CREATE AGGREGATE MONOTONIC.HAVING ($seq, $var, $attr) AS
       HAVING EXISTS ?k IN $seq: GRAPH ?k {$var sie:showsFault "true"}
       AND FORALL ?i, ?j  IN $seq:
           IF ( ?i < ?j < ?k  AND GRAPH ?i {$var $attr ?x}
           AND GRAPH ?j {$var $attr ?y}) THEN ?x < ?y
```

queries can be unfolded into relational stream queries. We developed a dedicated STARQL2SQL$^{\oplus}$ translator that unfolds STARQL queries to SQL$^{\oplus}$ queries, i.e. SQL queries enhanced with the essential operators for stream handling.

C. ONTOP-SPATIAL

ONTOP-SPATIAL[1] extends Ontop to enable the on-the-fly GeoSPARQL-to-SQL translation on top of geospatial databases and thus becomes the first OBDA system with geospatial support. It is able to connect to a geospatial database (currently PostGIS, Spatialite or Oracle-spatial) and create virtual geospatial RDF graphs on top of it, using ontologies and mappings. It supports the following components of GeoSPARQL [26]: *Core*, *Topology Vocabulary*, *Geometry topology extension*, *RDFS entailment* and a subset of *Geometry Extension*. To the best of our knowledge, it is also the first GeoSPARQL implementation that supports the *query rewrite extension* of GeoSPARQL. In [2] we explain how GeoSPARQL queries are processed by ONTOP-SPATIAL and are transformed into the respective spatial SQL queries that are evaluated by geospatial databases.

For example, the GeoSPARQL query:

```
SELECT DISTINCT ?name ?build ?type
WHERE {?s1 f:type ?type .
 ?s1 geo:asWKT ?g1 .
 ?s2 geo:asWKT ?g2 .
 ?s2 rdf:type osm:Building .
 ?s2 osm:hasName ?name .
 ?s2 osm:buildingCategory ?build .
FILTER(geof:sfIntersects(?g1, ?g2)) }
```

retrieves buildings that are affected by floods (i.e., they have intersecting geometries with the flood geometries).

VI. QUERY PLANNING & EXECUTION

A. EXAREME

Relational queries produced by ONTOP, are handled by EXAREME, OPTIQUE's distributed *Database Management System* (DBMS) that also operates as the platforms federation hub. EXAREME is a system for elastic large-scale dataflow processing in the cloud [23], [18] that is publicly available as an open source project under the MIT License. It is responsible for federating information from external datasources and executing intensive queries. In the following, we present some of its key aspects:

EXAREME, that supports *parallelism* by allocating processing across different workers in a distributed environment, is separated into the following components (Figure 6): *(i)* the *Master* is the main entry point to the system, through the gateway, and is responsible for the coordination of the rest of the components; *(ii)* the *Query Optimizer* determines the most efficient distributed way to execute a given query; *(iii)* the *Scheduler* translates the optimal query plan into the distributed machine code of the system and creates the final execution plan by assigning tasks to workers; *(iv)* the *Execution Engine* schedules the operators of the query, respecting their dependencies in the dataflow graph and the available resources; it also monitors the dataflow execution and handles failures; *(v)* finally workers execute tasks and transfer intermediate results by fetching the partitions needed for the execution, each task is executed by an SQLite[2] database engine instance running on each worker.

Query evaluation over federated data sources takes into consideration processing capabilities of endpoints and examines the possibility to push specific query fragments for execution there. Exareme uses a Volcano-style optimizer [17], capable of identifying common subexpressions coming from different parts of complex queries and deciding on the re-usage of these subexpressions [28]. Possibilities to push processing are considered as a post-optimization step for each run of the search algorithm. Specifically, each node in the resulted query plan is marked as candidate for execution in an endpoint, if all data below him are coming from a single endpoint and no descendant node in the plan is re-used from a different part of the query. The final decision for the candidates depends on the number of descendant tables of each candidate. If this number in one, then the fragment is sent for execution in

[1]https://github.com/ConstantB/ontop-spatial

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

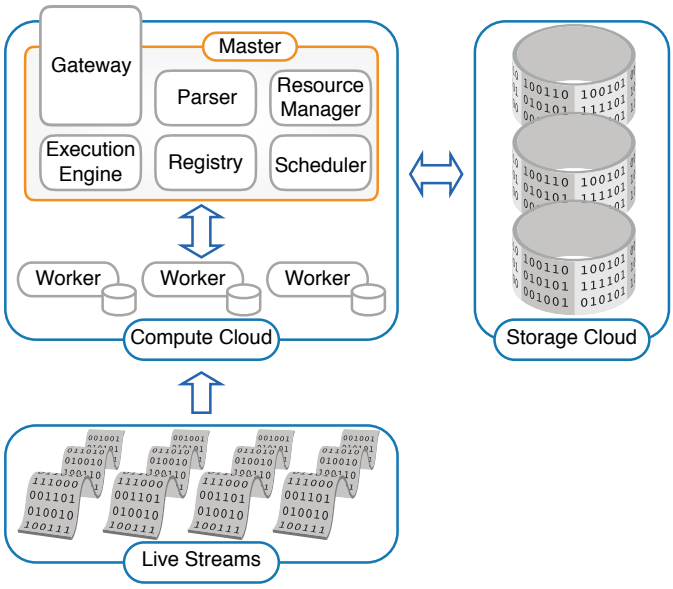

Fig. 6. Distributed Stream Engine Architecture

the endpoint, otherwise a cost-based comparison of the two alternatives is conducted. Exareme uses a federated analyser that gathers statistics about data from the external sources. This is an offline process, that only needs to be done for data mentioned in the OBDA mappings. After some sub-plan has been chosen for external evaluation, a specific virtual table operator responsible for the communication with the specific endpoint is called in order to send the fragment for execution and import the result into the system.

The EXAREME system natively supports *User Defined Functions* (UDFs) with arbitrary user code. The engine blends the execution of UDFs together with relational operators using JIT tracing compilation techniques that greatly speeds up the execution process. UDFs allow to express very complex dataflows using simple primitives. For OPTIQUE we used UDFs to implement communication with external sources, window partitioning on data streams, and data mining algorithms.

B. EXASTREAM

Relational queries produced by the STARQL2SQL$^{\oplus}$ translation, are handled by EXASTREAM [**?**], OPTIQUE's high-throughput distributed *Data Stream Management System* (DSMS).

EXASTREAM is built as a streaming extension of EXAREME, taking advantage of existing Database Management technologies and optimisations. It provides a declarative language, namely SQL$^{\oplus}$, for querying data streams and relations. In contrast to other DSMSs, the user does not need to consider low-level details of each 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 optimizer makes it possible to process SQL$^{\oplus}$ queries that blend streaming with static data. This has proven particularly useful in the Siemens use case since it allows us to combine streaming attributes (such as temperature measurements of a turbine) with metadata that remain invariant in time (such as the model or structure of a turbine) as well as archived stream data (such as past sensor readings, temperature measurements, etc.). Static relational tables may be stored in our system, or, they may be *federated* from external datasources. Moreover, EXASTREAM allows defining database schemata on top of streaming and static data. This gives a wide range of opportunities for applying Semantic Web technologies and optimisations, e.g., bootstrapping techniques, that rely on these features.

Whenever SQL abstractions are not sufficient (or efficient) for complex stream processing scenarios, we use standard SQL to combine data and process them with UDFs. Two main operators, implemented as UDFs, that incorporate the algorithmic logic for transforming SQLite into a DSMS are *timeSlidingWindow* and *wCache*: *(i) timeSlidingWindow* groups tuples from the same time window and associates them with a unique window id, *(ii) wCache* acts as an index for answering efficiently equality constraints on the time column when processing infinite streams. The time column may be the *window identifier* produced by the *timeSlidingWindow* operator. *WCache* will then produce results to multiple queries accessing different streams. The purpose of these UDFs is to perform the STARQL2SQL$^{\oplus}$ translation, while they remain hidden from OPTIQUE's users.

VII. CONCLUSIONS & FUTURE WORK

In this paper we presented OBDA, and explained the mechanisms that allow to provide direct end-user access to static, streaming, and geospatial information. We discussed the main requirements that an OBDA solution should fulfil in order to process disparate streaming and static sources and based on our findings we developed the OPTIQUE platform. The OPTIQUE platform can be seen as a federation hub where an ontology provides a 'global' schema that consolidates local schemata of the integrated data sources.

The OPTIQUE platform is currently being tested within the Siemens IT environment. The overall goal is to integrate the OPTIQUE platform into the Siemens system for stream monitoring and data analytics. Future work involves extending the OPTIQUE platform and the OBDA paradigm in order to *(i)* access NoSQL datasources such as SPARK and MongoDB endpoints [**?**], [**?**]; *(ii)* examine temporal extensions of the underlying formalisms [**?**]; *(iii)* and incorporate uncertainty and imprecision via fuzzy and probabilistic extensions of the traditional OBDA paradigm [**?**], [**?**], [**?**].

REFERENCES

[1] A. Arasu et al. The CQL continuous query language: sem- antic foundations and query execution. *VLDBJ*, 15(2), 2006.

[2] K. Bereta and M. Koubarakis. Ontop of geospatial databases. In *International Semantic Web Conference*, pages 37–52. Springer, 2016.

[3] D. Brickley, R. V. Guha, and B. McBride. Rdf vocabulary description language 1.0: Rdf schema. w3c recommendation (2004). *URL http://www. w3. org/tr/2004/rec-rdf-schema-20040210*, 2004.

[4] C. Pinkel et al. RODI: A benchmark for automatic mapping generation in relational-to-ontology data integration. In *ESWC*, 2015.

[5] D. Calvanese, B. Cogrel, S. Komla-Ebri, R. Kontchakov, D. Lanti, M. Rezk, M. Rodriguez-Muro, and G. Xiao. Ontop: Answering sparql queries over relational databases. *Semantic Web*, (Preprint):1–17, 2016.

[6] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The dl-lite family. *Journal of Automated reasoning*, 39(3):385–429, 2007.

[7] D. Calvanese, I. Horrocks, E. Jiménez-Ruiz, E. Kharlamov, M. Meier, M. Rodriguez-Muro, and D. Zheleznyakov. On rewriting, answering queries in obda systems for big data. In *OWLED*, 2013.

[8] B. Charron, Y. Hirate, D. Purcell, and M. Rezk. Extracting semantic information for e-commerce. In *The Semantic Web - ISWC 2016 - 15th International Semantic Web Conference, Kobe, Japan, October 17-21, 2016, Proceedings, Part II*, pages 273–290, 2016.

[9] D. Calvanese et al. Ontop: Answering SPARQL Queries over Relational Databases. *Sem. Web. Journal*, 2015.

[10] A. Doan, A. Y. Halevy, and Z. G. Ives. *Principles of Data Integration*. Morgan Kaufmann, 2012.

[11] E. Jiménez-Ruiz et al. BootOX: Practical Mapping of RDBs to OWL 2. In *ISWC*, 2015.

[12] E. Kharlamov at al. How Semantic Technologies Can Enhance Data Access at Siemens Energy. In *ISWC*, 2014.

[13] E. Kharlamov et al. Optique: Towards OBDA Systems for Industry. In *ESWC (Selected Papers)*, 2013.

[14] E. Kharlamov et al. Enabling Ontology Based Access at an Oil and Gas Company Statoil. In *ISWC*, 2015.

[15] E. Kharlamov et al. Optique: Ontology-based data access platform. In *ISWC (Posters & Demos)*, 2015.

[16] V. Gadepally, P. Chen, J. Duggan, A. J. Elmore, B. Haynes, J. Kepner, S. Madden, T. Mattson, and M. Stonebraker. The BigDAWG polystore system and architecture. *CoRR*, abs/1609.07548, 2016.

[17] G. Graefe and W. J. McKenna. The volcano optimizer generator: Extensibility and efficient search. In *Data Engineering, 1993. Proceedings. Ninth International Conference on*, pages 209–218. IEEE, 1993.

[18] H. Kllapi et al. Elastic Processing of Analytical Query Workloads on IaaS Clouds. *arXiv:1501.01070*, 2015.

[19] P. Haase, I. Horrocks, D. Hovland, T. Hubauer, E. Jiménez-Ruiz, E. Kharlamov, J. W. Klüwer, C. Pinkel, R. Rosati, V. Santarelli, A. Soylu, and D. Zheleznyakov. Optique system: towards ontology and mapping management in obda solutions. In *WoDOOM*, pages 21–32, 2013.

[20] P. Haase, C. Hütter, M. Schmidt, and A. Schwarte. The Information Workbench as a Self-Service Platform for Linked Data Applications. In *WWW*, 2012.

[21] V. Hristidis and Y. Papakonstantinou. Discover: Keyword Search in Relational Databases. In *VLDB*, 2002.

[22] R. Kontchakov, M. Rezk, M. Rodríguez-Muro, G. Xiao, and M. Zakharyaschev. Answering sparql queries over databases under owl 2 ql entailment regime. In *International Semantic Web Conference*, pages 552–567. Springer, 2014.

[23] M. Tsangaris et al. Dataflow Processing and Optimization on Grid and Cloud Infrastructures. *IEEE D. Eng. Bull.*, 32, '09.

[24] B. Motik, B. C. Grau, I. Horrocks, Z. Wu, A. Fokoue, C. Lutz, et al. Owl 2 web ontology language: Profiles. *W3C recommendation*, 27:61, 2009.

[25] Ö. Özçep et al. A Stream-Temporal Query Language for Ontology Based Data Access. In *KI*, 2014.

[26] M. Perry and J. Herring. Ogc geosparql-a geographic query language for rdf data. *OGC Implementation Standard. Sept*, 2012.

[27] M. Rodriguez-Muro, R. Kontchakov, and M. Zakharyaschev. Obda with ontop. In *ORE*, pages 101–106, 2013.

[28] P. Roy, S. Seshadri, S. Sudarshan, and S. Bhobe. Efficient and extensible algorithms for multi query optimization. In *ACM SIGMOD Record*, volume 29, pages 249–260. ACM, 2000.

[29] A. Soylu, M. Giese, E. Jimenez-Ruiz, G. Vega-Gorgojo, and I. Horrocks. Experiencing OptiqueVQS – a multi-paradigm and ontology-based visual query system for end-users. *Universal Access in the Information Society*, 15(1):129–152, 2016.

[30] A. Soylu, M. Giese, R. Schlatte, E. Jimenez-Ruiz, O. Ozcep, and 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)*, volume 1588 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2016.

[31] A. Soylu, E. Kharlamov, D. Zheleznyakov, E. Jimenez-Ruiz, M. Giese, and I. Horrocks. Ontology-based Visual Query Formulation: An Industry Experience. In *Proceedings of the 11th International Symposium on Visual Computing (ISVC 2015)*, volume 9474 of *LNCS*, pages 842–854. Springer, 2015.