Ontology-Based Integration of Streaming and Static Relational Data with Optique

Evgeny Kharlamov[1] Sebastian Brandt[2] Ernesto Jimenez-Ruiz[1] Yannis Kotidis[3]
Steffen Lamparter[2] Theofilos Mailis[4] Christian Neuenstadt[5] Özguer Özçep[5] Christoph Pinkel[6]
Christoforos Svingos[4] Dmitriy Zheleznyakov[1] Ian Horrocks[1] Yannis Ioannidis[4] Ralf Möller[5]

[1] University of Oxford [2] Siemens CT [3] Athens University of Economics and Business
[4] University of Athens [5] University of Lübeck [6] fluid Operations AG

ABSTRACT

Real-time processing of data coming from multiple heterogeneous data streams and static databases is a typical task in many industrial scenarios such as diagnostics of large machines. A complex diagnostic task may require a collection of up to hundreds of queries over such data. Although many of these queries retrieve data of the same kind, such as temperature measurements, they access structurally different data sources. In this work we show how Semantic Technologies implemented in our system OPTIQUE can simplify such complex diagnostics by providing an abstraction layer—ontology—that integrates heterogeneous data. In a nutshell, OPTIQUE allows complex diagnostic tasks to be expressed with just a few high-level semantic queries. The system can then automatically enrich these queries, translate them into a collection with a large number of low-level data queries, and finally optimise and efficiently execute the collection in a heavily distributed environment. We will demo the benefits of OPTIQUE on a real world scenario from Siemens.

1. INTRODUCTION

Motivation. Real-time processing of streaming and static data is a typical task in many industrial scenarios such as diagnostics of large machines. This task is challenging since it often requires integration of data from multiple sources. For example *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 includes the structure of relevant equipment, history of its exploitation and repairs, and even weather conditions. 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 [10].

Ontology-Based Integration Approach. In order to streamline the diagnostic process at Siemens, we propose a data integration approach based on Semantic Technologies. In this paper we will refer to our approach as *Ontology-Based Stream-Static Data Integration* (*OBSSDI*). It 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 [8]. In *OBSSDI* the global schema 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 [10] encodes generic specifications of appliances, characteristics of sensors, materials, processes, descriptions of diagnostic tasks, etc. *OBSSDI* 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.

In *OBSSDI* the integrated data can be accessed by posing queries over the ontology, i.e., *ontological queries*. These queries are *hybrid*: they refer to both streaming and static data. Evaluation of such queries in *OBSSDI* 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 *OBSSDI* 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

SIGMOD'16, June 26-July 01, 2016, San Francisco, CA, USA

DOI: http://dx.doi.org/10.1145/2882903.2899385

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 *OBSSDI* system will automatically compute it from the the high-level ontological query. Another important benefit of *OBSSDI* 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.

OBSSDI extends existing semantic data integration solutions that either assume that data is in (static) relational DBs, e.g [3, 6], or streaming, e.g., [5, 17] but not of both kinds. *OBSSDI* also extends existing solutions for unified processing of streaming and static semantic data e.g. [7], since they assume that data is natively in the WC3 standardised RDF semantic data format while we assume the data to be relational and mapped to the semantic format.

Research Challenges. The benefits of *OBSSDI* come at a price. The main practical challenges for *OBSSDI* 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; and

[C3] 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.

Construction of ontologies and mappings in *OBSSDI* 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 *OBSSDI* systems. Addressing C2 is crucial since, to the best of our knowledge, no dedicated query language for hybrid semantic queries has the required properties. Addressing C3 is vital to ensure that *OBSSDI* queries are executable in reasonable time. Note that C3 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 [6].

Our Contributions. Besides proposing *OBSSDI*, we addressed the challenges C1-C3 and implemented our solutions in the OPTIQUE [11] system that has been successfully applied in several industrial contexts [12, 10, 2]. In order to address C1, we developed BOOTOX [9, 4], a system for "bootstrapping" *OBSSDI* assets by extracting, ontologies and mappings from static and streaming relational schema and data. In order to address C2, we introduced STARQL [19], a query language that allows for semantic queries over both streaming and static data. STARQL queries are expressed over OWL 2 QL ontologies, which are mapped to the underlying data via global-as-view mappings [8]. STARQL queries admit polynomial-time enrichment and can be unfolded into SQL$^{(+)}$ queries, i.e. SQL queries enhanced with operators for stream handling. Finally, in order to address C3, we introduced EXASTREAM [15, 18], a highly optimised engine capable of handling complex hybrid queries in real time. EXASTREAM 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. EXASTREAM incorporates several query optimisations such as adaptive main-memory indexing of stream measurements and native User Defined Functions that permit a user to express complex operators in a concise way. See Section 2 for more details on OPTIQUE solutions for C1-C3 challenges.

Demo Overview. Attendees will see how OPTIQUE simplifies diagnostics for Siemens: they will see how to set and monitor continuous diagnostic tasks as STARQL queries, how EXASTREAM can handle more than a thousand complex diagnostic tasks, and how to deploy OPTIQUE over Siemens data using BOOTOX. See Section 3 for more details on demo scenarios.

2. OPTIQUE SYSTEM

OPTIQUE is an integrated system that consist of multiple components to support *OBSSDI* end-to-end [11, 13]. 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 query enrichment, unfolding, and execution backends that allow to execute up to thousands complex ontological queries in highly distributed environments. In this section we give some details of three OPTIQUE components that address the C1-C3 challenges above.

Deployment Support. Our 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: $\mathit{Turbine}(f(\vec{x})) \leftarrow \exists \vec{y}\, \texttt{SQL}(\vec{x}, \vec{y})$, that can be seen as a view definition, where $\texttt{SQL}(\vec{x}, \vec{y})$ 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 $\exists \vec{y}\, \texttt{SQL}(\vec{x}, \vec{y})$ 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 [16] 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.

```
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
```

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

Diagnostic Queries. In order to express diagnostic tasks we developed a query language STARQL [19] that allows us to perform complex semantic queries blending streaming with static data.

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.

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 [10] as in Fig. 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 [19]: 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 queries can be unfolded into relational stream queries. We developed a dedicated STARQL2SQL$^{(+)}$ translator that unfolds STARQL queries to SQL$^{(+)}$ queries, i.e. SQL queries enhanced with the essential operators for stream handling.

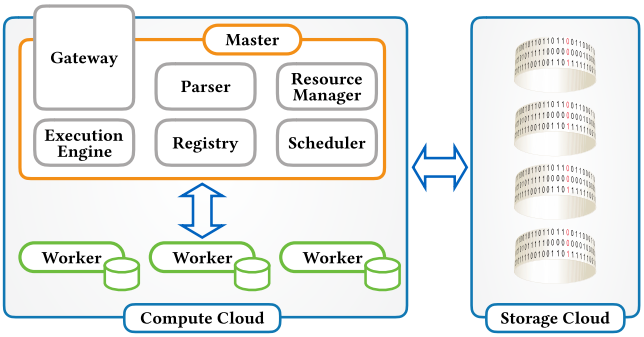

Figure 2: Distributed Stream Engine Architecture

Streaming and Static Relational Data Processing. Relational queries produced by the STARQL2SQL$^{(+)}$ translation, are handled by ExaStream, Optique's high-throughput distributed *Data Stream Management System* (*DSMS*). The ExaStream *DSMS* is embedded in Exareme, a system for elastic large-scale dataflow processing in the cloud [18, 15] that is publicly available as an open source project under the MIT License. In the following, we present some key aspects of ExaStream.

ExaStream is built as a streaming extension of the SQLite *DBMS*, taking advantage of existing Database Management technologies and optimisations. It provides a declarative language, namely SQL$^{(+)}$, for querying data streams and relations that conform to the CQL semantics [1]. In contrast to other *DSMS*, 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$^{(+)}$ 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 data-sources. 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.

ExaStream supports *parallelism* by distributing processing across different nodes in a distributed environment. Its architecture is shown in Figure 2. Queries are registered through the Asynchronous Gateway Server. Each registered query passes through the Exareme parser and then is fed to the Scheduler module. The Scheduler places stream and relational operators on worker nodes based on the node's load. These operators are executed by a Stream Engine instance running on each node.

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 JIT 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 optimizations at runtime that are not possible when the query is precompiled. *UDFs* allow us 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 such as the *Locality-Sensitive Hashing* technique [14] for computing the correlation between values of multiple streams.

Figure 3: OPTIQUE screenshots

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*:

- *timeSlidingWindow* groups tuples from the same time window and associates them with a unique window id,
- *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$^{(+)}$ translation, while they remain hidden from OPTIQUE's users.

In order to enable efficient processing of data streams of very high velocity we have implemented a number of optimisations in the stream processing engine. An optimisation that will be presented in the demo is *adaptive indexing*. With this technique EXASTREAM collects statistics during query execution and, adaptively, decides to build main-memory indexes on batches of cached stream tuples. These statistics are then used to expedite query processing during a complex operation (as in a join).

3. DEMONSTRATION SCENARIOS

We will demo the benefits of OPTIQUE on a real-world scenario from Siemens. In Figure 3 we presented some OPTIQUE screenshots about the deployment module and monitoring dashboards.

For the demonstration purpose we selected 20 diagnostic tasks typical for Siemens service centres and expressed these tasks in STARQL. An example diagnostic task is to calculate the Pearson correlation coefficient between turbine data streams. Then, we prepared a demo data set of streaming and static data from 950 turbines in the time from 2002 to 2011. This data is anonymised in a way that preserves the patterns needed for demo diagnostic tasks. During the demo we will 'play' the streaming data and thus emulate real time streams. Then, we distributed the demo-data in several installations with different number of nodes (VMs) ranging from 1 to 128, where each node has 2 processors and 4GB of main memory. To demonstrate diagnostics results we prepared a dedicated monitoring dashboard for each diagnostic task in the catalog. Dashboards show diagnostics results in real time, as well as statistics on streaming answers, relevant turbines, and other information that is typically required by the service engineers at Siemens. Finally, we deployed OPTIQUE over the Siemens data by bootstrapping ontologies and mappings and then manually post-processing and extending them so that they reach the required quality and contain necessary terms and mappings to cover 20 Siemens diagnostic tasks.

During the demo OPTIQUE will be available in three scenarios:

[S1] *Diagnostics with our deployment*: The attendees will be able to query our preconfigured Siemens deployment using diagnostic tasks from from the Siemens catalog and using their own STARQL queries, i.e., they will be able to create diagnostic tasks as parametrised continuous queries and register concrete instances of these tasks over specific data streams.

[S2] *Performance showcase of our deployment*: the attendees will be able to run various tests over our deployment using one of 128 preconfigured Siemens distributed environments and one of 10 test sets of queries. While running the tests they will monitor the throughput and progress of parallel query execution processes.

[S3] *Diagnostics with user's deployment*: the attendees will be able to deploy OPTIQUE over the Siemens data by bootstrapping ontologies and mappings, saving them, and observing and possibly improving them in dedicated editors. Then, they will query their deployed instance with diagnostic tasks from the Siemens catalog or their own STARQL queries.

Acknowledgements. This research has been partially supported by the EU project Optique (FP7-IP-318338), the Royal Society, the EPSRC grants Score!, DBonto, and MaSI^3.

4. REFERENCES

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

[2] A. Soylu et al. Ontology-based visual query formulation: An industry experience. In *ISVC*, 2015.

[3] C. Civili et al. MASTRO STUDIO: managing ontology based data access applications. *PVLDB*, 6(12), 2013.

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

[5] J. Calbimonte. Enabling ontology-based access to streaming data sources. In *ISWC*, 2010.

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

[7] D. L. Phuoc et al. A native approach for unified processing of linked streams and linked data. In *ISWC*, 2011.

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

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

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

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

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

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

[14] G. Nikos et al. In-network approximate computation of outliers with quality guarantees. *Inf. Sys.*, 38(8), 2013.

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

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

[17] L. Fischer et al. Scalable linked data stream processing via network-aware workload scheduling. In *SSWKBS*, 2013.

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

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