City Research Online

City, University of London Institutional Repository

Citation: Kharlamov, E., Brandt, S., Jimenez-Ruiz, E. ORCID: 0000-0002-9083-4599, Kotidis, Y., Lamparter, S., Mailis, T., Neuenstadt, C., Oezcep, O., Pinkel, C., Svingos, C., Zheleznyakov, D., Horrocks, I., Ioannidis, Y. and Moeller, R. (2016). Ontology-Based Integration of Streaming and Static Relational Data with Optique. In: SIGMOD '16 Proceedings of the 2016 International Conference on Management of Data. (pp. 2109-2112). New York: ACM. ISBN 978-1-4503-3531-7

This is the accepted version of the paper.

This version of the publication may differ from the final published version.

Permanent repository link: http://openaccess.city.ac.uk/id/eprint/22947/

Link to published version: http://dx.doi.org/10.1145/2882903.2899385

Copyright and reuse: City Research Online aims to make research outputs of City, University of London available to a wider audience. Copyright and Moral Rights remain with the author(s) and/or copyright holders. URLs from City Research Online may be freely distributed and linked to.

City Research Online: http://openaccess.city.ac.uk/ publications@city.ac.uk

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

E. Kharlamov[1] S. Brandt[2] E. Jimenez-Ruiz[1] Y. Kotidis[6] S. Lamparter[2] T. Mailis[3] C. Neuenstadt[4]
Ö. Özçep[4] C. Pinkel[5] C. Svingos[3] D. Zheleznyakov[1] I. Horrocks[1] Y. Ioannidis[3] R. Möller[4]
[1] Uni. of Oxford [2] Siemens CT [3] Uni. of Athens [4] Uni. of Lübeck [5] fluid Operations [6] AUEB

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 fleet of up to hundreds of queries over such data. Although many of these queries retrieve data of the same kind like temperature measurements, they are different since they access structurally different data sources. We have investigated how Semantic Technologies can make such complex diagnostics simpler by providing an abstraction semantic layer that integrates heterogeneous data. We developed the system OPTIQUE to put our ideas in practice. In a nutshell, OPTIQUE allows to express complex diagnostic tasks with just a few high-level semantic queries. Then, the system can automatically enrich these queries, translate them into a fleet with a large number of low-level data queries, and finally optimise and efficiently execute the fleet in a heavily distributed environment. We will demo the benefits of OPTIQUE on a real world scenario of Siemens Energy. For this purpose we prepared anonymised streaming and static data relevant to 950 Siemens power generating turbines with more than $100,000$ sensors and deployed OPTIQUE on multiple distributed environments with up to 128 nodes. By registering and monitoring continuous semantic high-level queries that combine streaming and static data the demo attendees will be able to see how OPTIQUE makes diagnostics of turbines easy. They will also see how OPTIQUE can handle more than a thousand concurrent complex diagnostic tasks that integrate heterogeneous data in real-time with a 10 TB/day throughput. Finally, they will see that creating a semantic layer, such as the one over the Siemens demo data, can be done in realistic time with the help of our bootstrapping interactive system.

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 Energy* runs service centres dedicated to diagnostics of thousands power generating appliances across the globe. A typical task for such centres is to detect in real-time a failure of appliances 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 gas and steam turbines, generators, and compressors installed in power plants, and of static data that includes structure of equipment, history of its exploitation and repairs, and even weather conditions. These data is scattered across multiple and heterogeneous data streams with 30 GB/day throughput and static DBs with hundreds TBs of data.

Even for a single diagnostic task that a turbine may fail, Siemens engineers have to analyse streams with temperature measurements from up to $2,000$ censors installed in different parts of the turbine, analyse historical data of turbine's temperature, compute temperature patterns, compare them to patterns in other turbines, compare weather conditions, etc. This requires to pose a fleet with hundreds of queries, 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 computed answers is expensive—it takes up to 80% of overal diagnostic time [10].

Ontology-Based Integration Approach. To tackle this issue in Siemens Energy we propose a data integration approach that is 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 to create 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 [5]. 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 relationship between classes, composed classes, class hierarchies, etc. The Siemens Energy ontology that we developed [10] contains hundreds of terms and axioms that encode 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 Energy ontology is mapped to all specific data and procedures that return temperatures of 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 hidden 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 an ontological query in *OBSSDI* has three stages: *(i)* in *enrichment* stage the ontological query is automatically reformulated with the help of axioms in another ontological query in order to access as much of relevant data as possible, *(ii)* in *unfolding* stage the en-

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

riched ontological query is automatically translated with the help of mappings in possibly many queries over the data, *(iii)* in *execution* stage the unfolded data 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 show only *what* this data is about. This allows to formulate the Siemens Energy diagnostic task above using only one ontological query instead of a fleet of hundreds data queries that today Siemens IT specialists have to write. Observe that these fleet of queries does not disappear: the enrichment and unfolding stages of the evaluation by an *OBSSDI* system will turn the high-level ontological query into the fleet of low-level data queries automatically. Another important benefit of *OBSSDI* is *modularity* and *compositionality* of its assets: every mapping relates only one ontological term to the data, thus, the semantics of the ontology is modularised for each separate term which allows to construct its assets independently from each other and on demand; then, the same ontological terms can be used in different queries, thus, by defining mappings for only a few ontological terms one will be able compose many queries using these mapped terms.

OBSSDI extends existing semantic data integration solutions that either assume that data is in (static) relational DBs, e.g [3, 4], or streaming, e.g., [2, 6] but not of both kinds. *OBSSDI* also extends existing solutions for unified processing of streaming and static semantic data e.g. [13], 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 with 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 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 semantics of ontological queries,

[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 speedup 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 devoted query language for hybrid semantic queries has required properties. Addressing C3 is vital to ensue 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 devoted optimisation techniques performs poorly [3], since the queries that are automatically computed after enrichment and unfolding can be very inefficient, e.g., they contain many redundant joins and unions.

Our Contributions. Besides proposing *OBSSDI* we addressed the challenges C1-C3 and implemented our solutions in the OPTIQUE system. For C2, we introduced STARQL [12] query language that allows to pose semantic queries over both streaming and static data. STARQL queries are expressed over OWL 2 QL ontologies and *OBSSDI* mappings that relate each ontological term to a set of queries over the underlying data in the global-as-view fashion [5]. STARQL queries admit polinomial-time enrichment and can unfoldable into $\text{SQL}^{(+)}$ queries, i.e. SQL queries enhanced with the essential operators for stream handling. For C3, we introduced EXASTREAM [11, 14], 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. Finally, for C1, we developed BOOTOX [9], a system for bootstrapping, i.e., extracting, ontologies and mappings from static and streaming relational schema and data that proved its efficiency in creating *OBSSDI* assets. See Section 2 for more details on OPTIQUE solutions for C1-C3 challenges.

Demo Overview. During the demonstration the attendees will be able to see how OPTIQUE makes diagnostics for Siemens easy: they will set and monitor continuous diagnostic tasks as STARQL queries, see how EXASTREAM can handle more than a thousand complex diagnostic tasks, and 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. For IT specialists OPTIQUE offers support for the whole lifecycle of ontologies and mappings: semi-automatic bootstrapping form 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 allows to extract W3C standardised OWL 2 ontologies and R2RML mappings from relational streaming and static data. Consider for example 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 (existentially quantified) 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. For example, 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 [8] for keyword-based query answering over DBs. Moreover, BOOTOX also allows to incorporate third party OWL 2 ontologies in an existing OPTIQUE's deployment using ontology alignment techniques.

The ontological terms bootstrapped with BOOTOX are then used

```
CREATE STREAM S_out AS
CONSTRUCT    GRAPH NOW { ?c2 rdf:type :MonInc }
FROM         STREAM S_Msmt [NOW-"PT10S"^^xsd:duration, NOW]->"PT1S"^^xsd:duration,
             STATIC DATA <http://www.optique-project.eu/siemens/ABoxstatic>,
             ONTOLOGY <http://www.optique-project.eu/siemens/TBox>
USING        PULSE WITH START = "00:10:00CET", FREQUENCY = "1S"
WHERE        {?c1 a sie:Assembly. ?c2 a sie:Sensor. ?c1 sie:inAssembly ?c2.}
SEQUENCE BY  StdSeq AS seq
HAVING       MONOTONIC.HAVING(?c2,sie:hasValue)

CREATE AGGREGATE MONOTONIC:HAVING ($var,$attr) AS
HAVING EXISTS ?k IN SEQ: GRAPH ?k { $var sie:showsFailure } AND
        FORALL ?i < ?j  IN seq, ?x, ?y:
        IF ( ?i,  ?j < ?k  AND GRAPH ?i {$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

to formulate STARQL ontological queries and the bootstrapped mappings – to translate these queries into data queries. We shall now discuss STARQL queries and their translation.

Diagnostic Queries. In order to express diagnostic tasks we developed a query language STARQL [12] that allows 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, thus allowing to employ the result of one query as input when constructing 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 space limit we cannot present STARQL in details. Instead, we will illustrate its main features on the following example diagnostic task: *Detect a real-time failure of the turbine caused by the a temperature increase within 10 seconds.* This task can be expressed using STARQL over the Siemens ontology [10] as in Figure 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 seconds) and with slide (here 1 second). The `PULSE` declaration specifies the output frequency. In the `WHERE` clause bindings for sensors (attached to some turbine's assembly) 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(?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 [12]: 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 OWL 2 QL ontology language and the queries are essentially conjunctive with value comparison and aggregate functions. 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 devoted STARQL2SQL$^{(+)}$ translator that unfolds STARQL queries to SQL$^{(+)}$ queries, i.e. SQL queries enhanced with the essential operators for stream handling.

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 on the cloud [11, 14] that has been 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 conforms to the CQL semantics [1]. In contrast to other *DSMS*, the user does not need to consider low-level details of the execution of a query. 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 been proved mostly useful in the Siemens use case since it allows 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 pre-compiled. *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 such as the *Locality-Sensitive Hashing* technique [7] for computing the correlation between values of multiple streams.

Whenever SQL abstractions are not sufficient (or efficient) for complex stream processing scenarios, we use standard SQL to com-

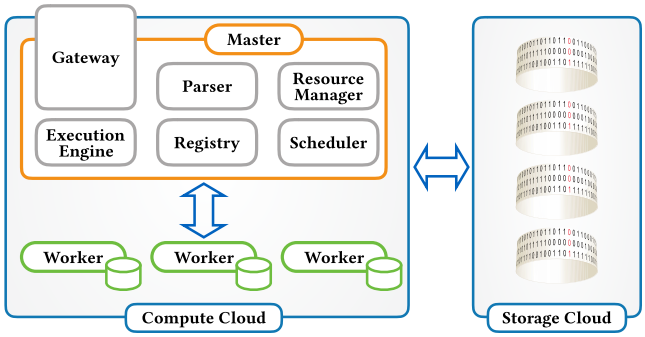

Figure 2: Distributed Stream Engine Architecture

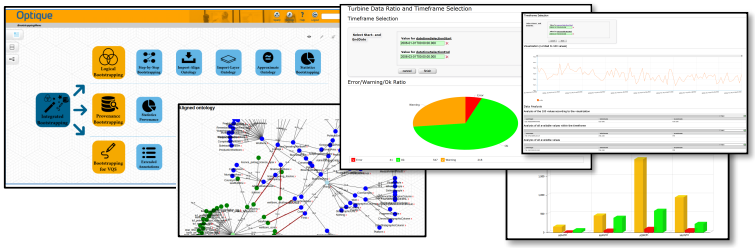

Figure 3: OPTIQUE screenshots

bine 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 that belong to 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.

These *UDFs* are transparent to OPTIQUE's users and are intended for performing the STARQL2SQL$^{(+)}$ translation.

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, in order to expedite their processing during a complex operation (as in a join).

3. DEMONSTRATION SCENARIOS

The benefits of OPTIQUE will be demonstrated on the real world scenario from Siemens Energy. In particular, we will show that:

- *formulating diagnostic tasks with* OPTIQUE *is practical*: Siemens diagnostic queries in OPTIQUE are concise and conceptually easy while fleets of Siemens data queries are and large and hard to comprehend,
- *running diagnostic tasks in* OPTIQUE *is practical*: OPTIQUE allows to process in real time up to $1,024$ complex Siemens diagnostic tasks with the throughput of up to $10,000,000$ tuples/sec by executing the tasks in parallel on a highly distrbute environment with up to 128 nodes,
- *creating* OPTIQUE *ontologies and mappings is practical*: OPTIQUE allows to create ontologies and mappings necessary for system deployment over Siemens streaming and static data in a reasonable time.

For the demonstration purpose we selected 20 diagnostic tasks typical for Siemens Energy service centres and expressed these tasks in STARQL. An example diagnostic task is to calculate the Pearson correlation coefficient between turbine stream data. Then, we prepared a demo data set that contains streaming and static data produced by 950 gas and steam turbines during 2002–2011 years. 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 devoted 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 Siemens Energy service engineers. 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 attendeed 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 progresses.

[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 devoted editors. Then, the attendees will query their deployment with diagnostic tasks from from the Siemens catalog or their own STARQL queries.

In Figure 3 we presented some OPTIQUE screenshots about the deployment module BOOTOX and monitoring dashboards.

4. REFERENCES

[1] A. Arasu, S. Babu, and J. Widom. The CQL continuous query language: semantic foundations and query execution. In: *VLDBJ* 15.2 (2006).

[2] J. Calbimonte, Ó. Corcho, and A. J. G. Gray. Enabling Ontology-Based Access to Streaming Data Sources. In: *ISWC*. 2010.

[3] D. Calvanese et al. Ontop: Answering SPARQL Queries over Relational Databases. In: *Sem. Web. Journal* (2015).

[4] C. Civili et al. MASTRO STUDIO: Managing Ontology-Based Data Access applications. In: *PVLDB* 6.12 (2013).

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

[6] L. Fischer, T. Scharrenbach, and A. Bernstein. Scalable Linked Data Stream Processing via Network-Aware Workload Scheduling. In: *SSWKBS@ISWC*. 2013.

[7] N. Giatrakos et al. In-network approximate computation of outliers with quality guarantees. In: *Information Systems* 38.8 (2013).

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

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

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

[11] H. Kllapi et al. Elastic Processing of Analytical Query Workloads on IaaS Clouds. In: *arXiv preprint arXiv:1501.01070* (2015).

[12] Ö. Özçep, R. Möller, and C. Neuenstadt. A Stream-Temporal Query Language for Ontology Based Data Access. In: *KI*. Vol. 8736. 2014.

[13] D. L. Phuoc et al. A Native and Adaptive Approach for Unified Processing of Linked Streams and Linked Data. In: *ISWC*. 2011.

[14] M. M. Tsangaris et al. Dataflow Processing and Optimization on Grid and Cloud Infrastructures. In: *IEEE Data Eng. Bull.* 32.1 (2009).