Preliminary version of a paper that appeared at Semper 16

Domain Experts Surfing on Stream Sensor Data over Ontologies

Ahmet Soylu[1], Martin Giese[2], Rudolf Schlatte[2], Ernesto Jimenez-Ruiz[3], Özgür Özçep[4], and Sebastian Brandt[5]

[1] Norwegian University of Science and Technology, Norway
ahmet.soylu@ntnu.no
[2] University of Oslo, Norway
{martingi, rudi}@ifi.uio.no
[3] University of Oxford, UK
ernesto.jimenez-ruiz@cs.ox.ac.uk
[4] University of Lübeck, Germany
oezcep@ifis.uni-luebeck.de
[5] Siemens AG, Germany
sebastian-philipp.brandt@siemens.com

Abstract. An increasing number of sensors are being deployed in business critical environments, systems, and equipments; and stream vast amount of data. The operational efficiency and effectiveness of business processes relies on domain experts' agility in interpreting data into actionable business information. Yet domain experts rarely have technical skills and knowledge on formal data retrieval tools, such as textual query languages, to specify and extract data of interest. In this paper, we report an ontology-based visual query system, namely OptiqueVQS, how it is extended for a stream query language called STARQL, and its first encounter with domain experts at Siemens AG. OptiqueVQS is valuable also conceptually as an instance of the end-user programming paradigm in pervasive environments aiming to empower end users to orchestrate data and devices distributed across the physical environment.

Keywords: Visual query formulation, Ontology, Streams, Sensors, OBDA

1 Introduction

The advances in pervasive computing and the emergence of low cost wireless and non-intrusive sensors open up new possibilities for industries such as oil and gas, power, and mining. For example, operators can recognize hazardous conditions by observing stream data coming from plant equipments such as pumps, motors, and turbines. However, the operational efficiency and effectiveness of business processes relies on domain experts' agility in interpreting data into actionable business information, so as to give reactive and proactive responses with respect to important data patterns appearing in data streams (cf. [17]). Yet domain experts rarely have technical skills and knowledge on formal textual query languages

for streams, such as CQL [1], C-SPARQL [2] and STARQL [11], to specify and extract data of interest.

Turnaround time between an important event and reaction could be reduced drastically, if domain experts could directly specify and isolate important data fragments rather than having IT experts in the middle. A simple example could be shutting down an overheated turbine; however, an event could also be of a more complex nature involving more than one sensory source and static data. Visual query formulation (cf. [5]) is a viable approach as it aims to lower the knowledge and skill barriers to a minimum. Ontology-based visual query formulation is gaining attention as ontologies come with certain benefits compared to visual query formulation over database schemas (cf. [14]). First of all ontologies provide higher level abstractions closer to end users' understanding, and from a technical point of view federation and reasoning are among the most promising (cf. [6]). Secondly, ontology-based data access (OBDA) technologies extend the reach of ontology-based querying from triple stores to relational databases (cf. [16]).

Much work is done on ontology-based visual query formulation for SPARQL and non-stream querying (cf. [14]). Therefore, we extended the functionality of our ontology-based visual query system, OptiqueVQS, for stream querying in the context of use cases provided by Siemens AG[1]. OptiqueVQS is valuable also conceptually as an example of end-user programming [10] in pervasive environments, and the concept presented in this paper is indeed valid for the end-user environment as well like, for instance, in terms of activity recognition [8]. The abundance of data and internet-connected objects render it difficult for IT experts to consider all possible eventualities and necessitate tools for empowering end users to orchestrate data and devices distributed across the physical environment on their own.

In what follows, Section 2 and Section 3 introduces the Siemens case and STARQL respectively. Section 4 presents OptiqueVQS with stream querying. Section 5 presents its first encounter with domain experts at Siemens AG. Finally Section 7 presents the related work and Section 7 concludes the paper.

2 The Siemens Use Case

Siemens runs several service centers for power plants, each responsible for remote monitoring and diagnostics of many thousands of gas/steam turbines and associated components such as generators and compressors. Diagnosis engineers working at the service centers are informed about any potential problem detected on site. They access a variety of raw and processed data with pre-defined queries in order to isolate the problem and to plan appropriate maintenance activities. For diagnosis situations not initially anticipated, new queries are required, and an IT expert familiar with both the power plant system and the data sources in question (e.g., up to 2.000 sensors in a part of appliance and static data sources) has to be involved to formulate these queries. Thus, unforeseen situations may lead to significant delays of up to several hours or even days.

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

With few built-in features for manipulating time intervals, traditional data base systems often offer insufficient support for querying time series data, and it is highly non-trivial to combine querying techniques with the statistics-based methods for trend analysis that are typically in use in such cases. By enabling diagnosis engineers to formulate complex queries on their own with respect to an expressive domain vocabulary, IT experts will not be required anymore for adding new queries, and manual preprocessing steps can be avoided.

3 STARQL

STARQL [11] provides an expressive declarative interface to both historical and streaming data. In STARQL, querying historical and streaming data proceeds in an analogous way and in both cases the query may refer to static data. The answers coming from the static sub-query are used for the stream processing in the remainder of the query. This separation between the static and dynamic aspects provides a useful abstraction which eases the query building process.

```
PREFIX ns1 : <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ns2 : <http://www.siemens.com/ontology/gasturbine/>
CREATE PULSE WITH FREQUENCY = "PT1s"^^xsd:duration

CREATE STREAM S_out AS
SELECT { ?_val0 ?Train_c1 ?Turbine_c2 ?Generator_c3 ?BearingHouse3_c4
  ?JournalBearing_c5 ?TemperatureSensor_c6 }
FROM STREAM measurement
  [NOW - "PT10s"^^xsd:duration, NOW]->"PT1s"^^xsd:duration
WHERE {
  ?Train_c1 ns1:type ns2:Train.
  ?Turbine_c2 ns1:type ns2:Turbine.
  ?Generator_c3 ns1:type ns2:Generator.
  ?BearingHouse3_c4 ns1:type ns2:BearingHouse3.
  ?JournalBearing_c5 ns1:type ns2:JournalBearing.
  ?TemperatureSensor_c6 ns1:type ns2:TemperatureSensor.
  ?Train_c1 ns2:hasTurbine ?Turbine_c2.
  ?Train_c1 ns2:hasGenerator ?Generator_c3.
  ?Generator_c3 ns2:hasBearingHouse3 ?BearingHouse3_c4.
  ?BearingHouse3_c4 ns2:hasJournalBearing ?JournalBearing_c5.
  ?JournalBearing_c5 ns2:isMonitoredBy ?TemperatureSensor_c6.
  ?Turbine_c2 ns2:hasName "Bearing Assembly"^^xsd:string.
}
SEQUENCE BY StdSeq AS seq
HAVING  EXISTS i IN seq
  ( GRAPH i { ?TemperatureSensor_c6 ns2:hasValue ?_val0 } )
```

Fig. 1. An example diagnostic task in STARQL.

The relevant slices of the temporal data are specified with a window. In the case of historical data, this is a window with fixed endpoints. In the case of streaming data, it is a moving window and it contains a reference to the developing time NOW and a sliding parameter that determines the rate at which snapshots of the data are taken. The contents of the temporal data are grouped according to a sequencing strategy into a sequence of small graphs that represent different states. On top of the sequence, relevant patterns and aggregations are formulated in the HAVING-clause, using a highly expressive template language. In Figure 1 an example STARQL query is given, which asks for a train with turbine named "Bearing Assembly", and queries for the journal bearing temperature reading in the generator. It uses a simple echo to display the results. For more information on STARQL, we refer to Ozcep et al. [11].

4 OptiqueVQS

OptiqueVQS[2] [13] is meant for end users who lack technical skills and knowledge. The interface of OptiqueVQS is designed as a widget-based user-interface mashup (UI mashup).

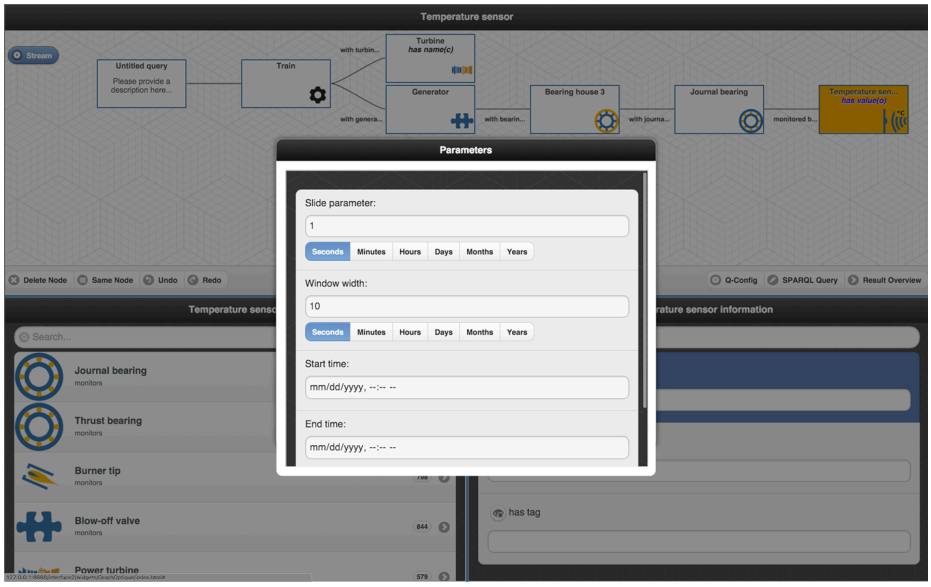

Fig. 2. OptiqueVQS with stream querying – parameter selection.

In Figure 2, a query is shown as a tree in the upper widget (W1), representing typed variables as nodes and object properties as arcs. New typed variables

[2] Demo video: https://youtu.be/TZTxujz5hCc

can be added to the query by using the list in the bottom-left widget (W2). If a query node is selected, the faceted widget (W3) at the bottom-right shows controls for refining the corresponding typed variable, e.g. setting a value for a data property or switching to a more specific concept. Once a restriction is set on a data property or a data property is selected for output (i.e., using the eye icon), it is reflected in the label of the corresponding node in the query graph. The user has to follow the same steps to involve new concepts in the query and can always jump to a specific part of the query by clicking on the corresponding variable-node in W1.

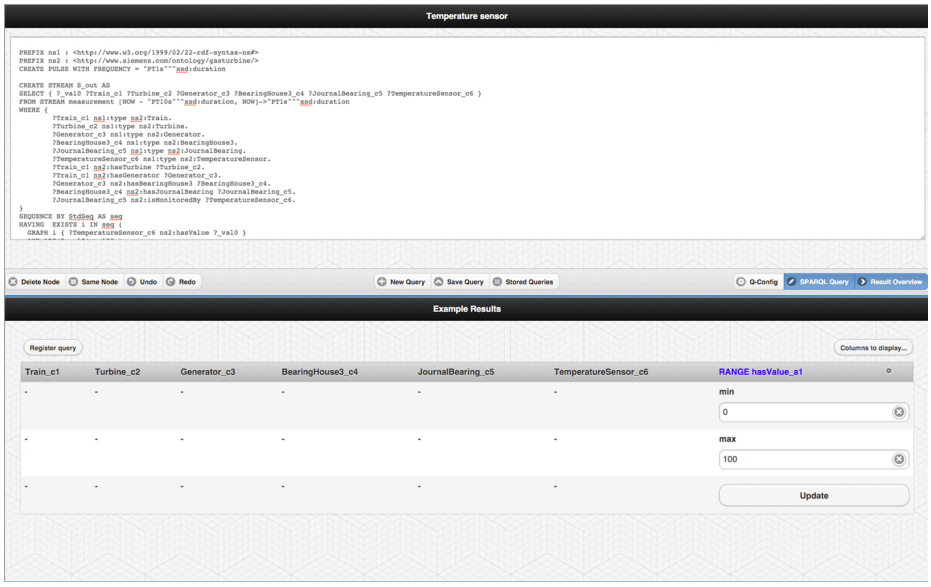

Fig. 3. OptiqueVQS with stream querying – template selection.

In W3, 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 W1 and lets the user configure parameters such as slide (i.e., frequency at which the window content is updated/moves forward) and window width interval. If the user clicks on the "Result Overview" button, a template selection widget (W4) appears for selecting a template for each stream attribute, which is by default "echo" (see Figure 3). W4 is normally used for displaying example results in SPARQL mode. The example query depicted in Figure 2 and Figure 3 represents the query example given in Section 3 with the exception that a "range" template is selected. The user can register the query in W4 by clicking on the "Register query" button.

OptiqueVQS currently supports three-shaped conjunctive queries and a restricted fragment of STARQL, for example, templates that correlate different

stream properties are not supported. We also refer interested readers to Soylu et al. [15,13] for the backend of OptiqueVQS (i.e., extraction of concepts and relationships).

5 User Study

The experiment was designed as a think-aloud study, since the goal of the experiment was not purely summative, but to a large extent formative. The experiment is built on a "turbine ontology" with 40 concepts and 65 properties.

Table 1. Profile information of the participants.

#	**Age**	**Occupation**	**Education**	**Technical skills**	**Similar tools**
P1	37	R&D engineer	PhD	4	1
P2	54	Diagnostics Engineer	Bachelor	5	3
P3	39	Engineer	PhD	5	2

A total of three participants, who cover the relevant occupation profiles, took part in the experiment; the profiles of participants are summarized in Table 1. A brief introduction on the topic and tool was delivered to the participants along with an example. Then they were asked to fill in a profile survey. The survey asks users about their age, occupation and level of education, and asks them to rate their technical skills, such as on programming and query languages, and their familiarity with similar tools on a Likert scale (i.e., 1 for "not familiar at all," 5 for "very familiar"). Participants were then asked to formulate a series of information needs as queries with OptiqueVQS, given at most three attempts for each query. Each participant performed the experiment in a dedicated session, while being observed by a surveyor. Participants were instructed to think aloud, including any difficulties they encountered (e.g., frustration and confusion), while performing the given tasks. Table 2 lists the tasks (3-5 are stream queries) representing the information needs used in the experiment.

Once users were done with the tasks, they were asked to fill in an exit survey asking about their experiences with the tool. The survey asks users to rate whether the questions were easy to do with the tool (*S1*), the tool was easy to learn (*S2*), was easy to use (*S3*), gave a good feeling of control and awareness (*S4*), was aesthetically pleasing (*S5*), was overall satisfactory (*S6*), and was enjoyable to use (*S7*) on a Likert scale (again, 1 for "strongly disagree" and 5 for "strongly agree"). Users were also asked to comment on what they did like and dislike about the tool and to provide any feedback which they deem important.

The results of the experiment are presented in Table 3. In total, 15 tasks were completed by the participants with 100 percent correct completion rate and 66 percent first-attempt correct completion rate. One should be aware that query formulation is an iterative process and query reformulation is a natural step.

Table 2. Information needs used in the experiment.

#	**Information need**
T1	Display all trains that have a turbine and a generator.
T2	Display all turbines together with the temperature sensors in their burner tips. Be sure to include the turbine name and the burner tags.
T3	For the turbine named "Bearing Assembly", query for temperature readings of the journal bearing in the compressor. Display the reading as a simple echo.
T4	For a train with turbine named "Bearing Assembly", query for the journal bearing temperature reading in the generator. Display readings as a simple echo.
T5	For the turbine named "Burner Assembly", query for all burner tip temperatures. Display the readings if they increase monotonically.

Table 3. The results of the experiment (c for complete, t for time in seconds, and a for attempt count).

	T1			**T2**			**T3**			**T4**			**T5**			**Av.**		
#	c	t	a	c	t	a	c	t	a	c	t	a	c	t	a	c	t	a
P1	1	120	1	1	150	1	1	130	1	1	70	1	1	60	1	1	106	1.0
P2	1	120	1	1	180	2	1	240	2	1	60	1	1	180	1	1	156	1.4
P3	1	45	1	1	40	2	1	40	2	1	60	2	1	60	1	1	168	1.6
Av.	1	95	1	1	123	1.6	1	136	1.6	1	63	1.3	1	100	1	1	143	1.3

Table 4. The results of the exit survey.

Question	**P1**	**P2**	**P3**	**Avg.**
"I think that I would like to use this system frequently."	5	4	4	4.3
"I found the system unnecessarily complex."	1	3	2	2.0
"I thought the system was easy to use."	5	4	5	4.6
"I think that I would need the support of a technical person to be able to use this system."	1	1	1	1.0
"I found the various functions in this system were well integrated."	4	4	4	4.0
"I thought there was too much inconsistency in this system."	2	2	2	2.0
"I would imagine that most people would learn to use this system very quickly."	5	4	4	4.3
"I found the system very cumbersome to use."	1	2	2	1.6
"I felt very confident using the system."	4	4	3	3.6
"I needed to learn a lot of things before I could get going with this system."	2	1	1	1.3

The feedback provided by the participants through the exit survey is presented in Table 4 and Table 5. The usability scores given by participants are quite high. Users' comments suggest that they did like the design of interface, while they had minor issues with the fact that users need click on the "Run Query" button in order to select a template from the tabular view. A straight forward solution for stream based queries would be to change the name of button to "Select a Template" to prevent confusion, as the "Run Query" button is originally meant for non-stream query tasks. Users generally praised the capabilities and the design of OptiqueVQS. Compared to diagnosis engineers, R&D engineers often need to formulate more complex queries, which include advanced operators such as "OR" and negation. Users also would like to be able to combine multiple queries and be able to connect concepts which are not directly linked.

Table 5. The feedback given by the participants.

"What did you like about the tool?"	**Person**
"Easy to learn"	P1
"Nice user interface"	P1
"Possibility to see the original query"	P1
"Very comfortable to use"	P2
"UI looks really nice"	P3
"The floating tree shows exactly what kind of situation I am looking for. Gives a nice overview."	P3
"The interaction between the buttons and the tree work really well."	P3
"The turbine structure is really useful to find sensors quickly."	P3
"Very nice icons for the turbine parts."	P3
"Especially complex queries appear easy to understand."	P3
"What didn't you like about the tool?"	**Person**
"Should be possible to extend search for things not directly connected to the current concept."	P1
"When selecting a turbine name, the turbine box in the tree does not show me the turbine name but only c. I find this confusing."	P2
"Did not always know where to click for the stream part. E.g., the little circle on in the column."	P3
"It may be confusing to have to run the query before specifying it further. Could it be run automatically, e.g., after each change to the tree?"	P3
"Did not know what the start time and end time field means for a stream. Is that automatically registering / de-registering the query at a certain time point?"	P3
"I don not understand what the numbers on the buttons mean. Is that the number of instances of the item (i.e., turbine.)"	P3
"I find the order of items confusing. This is not alphabetical and also does not make sense from the structure of the turbine."	P3
"Why am I offered sensors that don't exist at certain locations? For instance, I see 'RotationSpeed' for the burners?"	P3

Overall, the high completion and satisfaction rates suggest that OptiqueVQS with streaming functionality is promising for end-user querying of stream data. Earlier, a user experiment was conducted with casual users [13] and another with domain experts at Statoil ASA [15] on non-streaming scenarios. The results confirm the value of OptiqueVQS as an end-user visual query formulation tool.

6 Related Work

Other notable examples of stream query languages in the Semantic Web are C-SPARQL [2], SPARQLstream [3], and CQELS [9]. These approaches extend SPARQL with a window operator whose content is a multi-set of variable bindings for the open variables in the query. Ozcep et al. [11] compare and discuss advantages and disadvantages of different approaches. However, in this paper we are rather interested in visual solutions sitting on top of any of these languages.

Although several visual tools exist for SPARQL (cf. [14]), the work is very limited for stream languages. An example is SPARQL/CQELS visual editor designed for Super Stream Collider framework [12]. However, the tool follows the jargon of the underlying language closely and is not appropriate for end users as it will demand considerable technical knowledge and skills. OptiqueVQS is a visual query system rather than a visual query language and it is not our concern to reflect the underlying formality (i.e., query language and ontology) per se. However, user behaviour is constrained so as to enforce the generation of valid queries. We are also not interested in providing full expressivity, as we believe simpler interfaces will suffice for the majority of end user queries.

Indeed, OptiqueVQS is a part of an OBDA platform, namely Optique [7]. Optique employs a data virtualisation approach to enable in-place querying of legacy relational data sources over ontologies. This is realised through a set of mappings, which describe the relationships between the terms in the ontology and their representations in the data sources and through query rewriting mechanisms for SPARQL to SQL and STARQL to CQL [4,11].

7 Conclusion

OptiqueVQS has been developed with real requirements collected from industrial partners and continuously evaluated in different contexts. In this paper, it is shown that end-user access to stream data sources with OptiqueVQS is a viable solution; and, end-user programming in pervasive environments is a reality.

Acknowledgments. This research is funded by the Seventh Framework Program (FP7) of the European Commission under Grant Agreement 318338, “Optique”.

References

1. Arasu, A., Babu, S., Widom, J.: The CQL Continuous Query Language: Semantic Foundations and Query Execution. The VLDB Journal 15(2), 121–142 (2006)

2. Barbieri, D.F., Braga, D., Ceri, S., Della Valle, E., Grossniklaus, M.: C-SPARQL: SPARQL for Continuous Querying. In: Proceedings of the 18th International Conference on World Wide Web (WWW 2009). pp. 1061–1062. ACM (2009)
3. Calbimonte, J.P., Corcho, O., Gray, A.J.G.: Enabling Ontology-based Access to Streaming Data Sources. In: Proceedings of the 9th International Semantic Web Conference (ISWC 2010). LNCS, vol. 6496, pp. 96–111. Springer (2010)
4. Calvanese, D., Cogrel, B., Komla-Ebri, S., Kontchakov, R., Lanti, D., Rezk, M., Rodriguez-Muro, M., Xiao, G.: Ontop: Answering sparql queries over relational databases. Semantic Web (in press)
5. Catarci, T., Costabile, M.F., Levialdi, S., Batini, C.: Visual query systems for databases: A survey. Journal of Visual Languages and Computing 8(2), 215–260 (1997)
6. Giese, M., Calvanese, D., Haase, P., Horrocks, I., Ioannidis, Y., Kllapi, H., Koubarakis, M., Lenzerini, M., Möller, R., Özcep, O., Rodriguez-Muro, M., Rosati, R., Schlatte, R., Schmidt, M., Soylu, A., Waaler, A.: Scalable End-user Access to Big Data. In: Akerkar, R. (ed.) Big Data Computing. CRC Press (2013)
7. Giese, M., Soylu, A., Vega-Gorgojo, G., Waaler, A., Haase, P., Jimenez-Ruiz, E., Lanti, D., Rezk, M., Xiao, G., Ozcep, O., Rosati, R.: Optique – Zooming In on Big Data Access. IEEE Computer 48(3), 60–67 (2015)
8. Krishnan, N.C., Cook, D.J.: Activity Recognition on Streaming Sensor Data. Pervasive and Mobile Computing 10, 138–154 (2014)
9. Le-Phuoc, D., Dao-Tran, M., Parreira, J.X., Hauswirth, M.: A Native and Adaptive Approach for Unified Processing of Linked Streams and Linked Data. In: Proceedings of the 10th International Conference on The Semantic Web (ISWC 2011). LNCS, vol. 7031, pp. 370–388. Springer (2011)
10. Lieberman, H., Paterno, F., Wulf, V. (eds.): End User Development, Human-Computer Interaction Series, vol. 9. Springer (2006)
11. Ozcep, O.L., Moller, R., Neuenstadt, C.: A Stream-Temporal Query Language for Ontology Based Data Access. In: The 37th Annual German Conference on Artificial Intelligence (KI 2014). LNCS, vol. 8736, pp. 183–194. Springer (2014)
12. Quoc, H.N.M., Serrano, M., Phuoc, D.L., Hauswirth, M.: Super Stream Collider: Linked Stream Mashups for Everyone. In: Proceedings of the Semantic Web Challenge at ISWC2012 (2012)
13. Soylu, A., Giese, M., Jimenez-Ruiz, E., Vega-Gorgojo, G., Horrocks, I.: 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)
14. Soylu, A., Giese, M., Kharlamov, E., Jimenez-Ruiz, E., Zheleznyakov, D., Horrocks, I.: Ontology-based End-user Visual Query Formulation: Why, what, who, how, and which? Universal Access in the Information Society (accepted)
15. Soylu, A., Kharlamov, E., Zheleznyakov, D., Jimenez-Ruiz, E., Giese, M., Horrocks, I.: Ontology-based Visual Query Formulation: An Industry Experience. In: Proceedings of the 11th International Symposium on Visual Computing (ISVC 2015). LNCS, vol. 9474, pp. 842–854. Springer, Las Vegas, Nevada, USA (2015)
16. Spanos, D.E., Stavrou, P., Mitrou, N.: Bringing relational databases into the Semantic Web: A survey. Semantic Web 3(2), 169–209 (2012)
17. Yang, Y., Wu, X., Zhu, X.: Combining Proactive and Reactive Predictions for Data Streams. In: Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining (KDD 2005). pp. 710–715. ACM (2005)