Towards Expressive Stream Reasoning

Heiner Stuckenschmidt[1], Stefano Ceri[2],
Emanuele Della Valle[2], Frank van Harmelen[3]

[1] University of Mannheim
[2] Politechnico di Milano
[3] Vrije Universiteit Amsterdam

February 26, 2010

Abstract

Stream Data processing has become a popular topic in database research addressing the challenge of efficiently answering queries over continuous data streams. Meanwhile data streams have become more and more important as a basis for higher level decision processes that require complex reasoning over data streams and rich background knowledge. In previous work the foundation for complex reasoning over streams and background knowledge was laid by introducing technologies for wrapping and querying streams in the RDF data format and by supporting simple forms of reasoning in terms of incremental view maintenance. In this paper, we discuss how this existing technologies should be extended toward richer forms of reasoning using Sensor Networks as a motivating example.

1 Motivation

The use of the Internet as a major source of information has created new challenges for computer science and has let to significant innovation in areas such as databases, information retrieval and semantic technologies. Currently, we are facing another major change in the way information in provided. Traditionally information used to be mostly static with changes being the exception rather than the rule. Nowadays, more and more dynamic information, which used to be hidden inside dedicated systems, is getting available to decision makers.

For instance, oil operation engineers base their decision processes on real time data acquired from sensors on oil rigs, on the sea surface and on the seabed. A typical oil production platform is equipped with about 400.000 sensors for measuring environmental and technical parameters. Some of the questions they faces are:

- Given an alarm on a well in progress to drown, how long time do I have given the historical behavior of that well?

Dagstuhl Seminar Proceedings 10042
Semantic Challenges in Sensor Networks
http://drops.dagstuhl.de/opus/volltexte/2010/2555

- Given this brand of turbine, what is the expected time to failure when the barring starts to vibrate as now detected?
- How do I detect weather events from observation data?
- Which sensors have observed a Blizzard within a 100 mile radius of a given location.

Answering these questions requires to process an (almost) "continuous" flow of information – with the recent information being more relevant as it describes the current state of a dynamic system – against a rich background knowledge.

Continuous processing of these flows of information (namely data streams) has been largely investigated in the database community [3]. Specialized Data Stream Management Systems (DSMS) are available on the market and features of DSMS are appearing also in major database products, such as Oracle and DB2.

On the contrary, continuous processing of data streams *together with rich background knowledge* requires specialized reasoners, but work on semantic technologies is still focusing on rather static data. In existing work on logical reasoning, the knowledge base is always assumed to be static (or slowly evolving). There is work on changing beliefs on the basis of new observations [19], but the solutions proposed in this area are far too complex to be applicable to gigantic data streams of the kind illustrated in the oil production example above.

As argued in [16], we strongly believe that there is a need to close this gap between existing solutions for belief update and the actual needs of supporting decision process based on data streams and rich background knowledge. We named this little explored, yet high-impact research area Stream Reasoning.

Definition 1 ***Stream Reasoning****: logical reasoning in real time on gigantic and inevitably noisy data streams in order to support the decision process of extremely large numbers of concurrent users.*

We have been working on stream reasoning since 2008. In previous work, we laid the foundation for complex reasoning over streams and background knowledge by introducing technologies for wrapping and querying streams in the RDF data format [15, 9, 6] and by supporting simple forms of reasoning in terms of incremental maintenance of materializations of ontological entailments [8]. We wrap up those works in Section 2. In Section 3, we discuss how these existing technologies should be extended toward richer forms of reasoning. In particular, we dedicate Section 4 to describe an architecture that supports these richer forms of stream reasoning. We further discuss the relation between our ideas on stream reasoning and the semantic sensor network initiative[1] in Section 5. Finally in Section 6, we close with a summary of open problems and future works.

[1] http://www.w3.org/2005/Incubator/ssn/

2 Previous Work

In order to bridge the identified gap, we have been working since 2008 on Continuous-SPARQL (or shortly C-SPARQL) [4, 9], an extension to SPARQL[24] for continuous querying over streams of RDF and background knowledge represented as RDF graphs.

Listing 1 shows an example of C-SPARQL query that detects a blizzard: a severe storm condition lasting for 3 hours or more characterized by low temperatures, strong winds, and heavy snow.

```
1  REGISTER STREAM BlizzardDetection COMPUTE EVERY 10m AS
2  PREFIX grs: <http://www.georss.org/georss/>
3  PREFIX so: <http://knoesis.wright.edu/ssw/ont/sensor-observation.owl#>
4  PREFIX w: <http://knoesis.wright.edu/ssw/ont/weather.owl#>
5  CONSTRUCT {?station so:generatedObservation [ rdf:type w:blizard ] }
6  FROM <http://streamreasoning.org/weatherStations.rdf>
7  FROM STREAM <http://streamreasoning.org/weatherObservation.rdf>
8    [RANGE 3h STEP 10m]
9  WHERE {
10   ?station grs:point "66.348085,10.180662";
11            so:generatedObservation ?o1, ?o2, ?o3 .
12         ?o1 rdf:type w:TemperatureObservation ;
13             so:observedProperty w:\_AirTemperature ;
14             so:result [ so:value ?temperature  ] .
15         ?o2 rdf:type w:WindObservation ;
16             so:observedProperty w:\_WindSpeed ;
17             so:result [ so:value ?speed  ] .
18   ?o3 rdf:type w:SnowfallObservation .
19 }
20 AGGREGATE { (?averageTemperature, AVG(?temperature), ?station) }
21 FILTER (?averageTemperature < "0.0"^^xsd:float)
22 AGGREGATE { (?minSpeedWind, MIN(?speed), ?station) }
23 FILTER (?minSpeedWind > "40.0"^^xsd:float)
```

Listing 1: Example of C-SPARQL which detects a blizzard

At line 1, the REGISTER clause is use to tell the C-SPARQL engine that it should register a **continuous query**, i.e. a query that will continuously compute answers to the query. In particular, we are registering a query that generates as output an RDF stream (i.e., we use REGISTER STREAM). The COMPUTE EVERY clause states the frequency of every new computation, in the example every 10 minutes. At line 6, the standard SPARQL clause FROM is used to load in the default graph the location of all weather stations. At line 7, the clause FROM STREAM defines the RDF stream of weather observations. We supposed that the observations are encoded in RDF using the Semantic Sensor Web ontology[2]. Next, line 8 defines the **window** of observation over the RDF stream of weather observations. Streams, for their very nature, are volatile and for this reason should be consumed on the fly; thus, they are observed through a window, including the last elements of the stream, which changes over time. In the example, *the window comprises weather observations produced in the last 3 hour, and the window slides every 10 minutes*. The WHERE clause is standard; it includes a set of matching patterns and FILTER clauses as in standard SPARQL. At line 20 and 22, the AGGREGATE function is used to compute two aggregates: the average temperature and the minimum wind speed. At line 21 and 23, the FILTER clause is used to select only those stations where

[2] http://knoesis.wright.edu/research/semsci/application_domain/sem_sensor/ont/sensor-observation.owl

the average temperature has been below 0 and the minimum wind speed has been above 40 km/h. Finally, selected stations are used to construct the elements of the RDF stream specified in the CONSTRUCT clause at line 5.

As Listing 1 illustrates, C-SPARQL enables the encoding of the typical questions an oil operation engineer has to answer. This is possible, because C-SPARQL extends SPARQL with the notions of *window* and of *continuous processing*.

Two approaches, alternative C-SPARQL exists: Streaming SPARQL [12] and Time-Annotated SPARQL (or simply TA-SPARQL) [25]. Both languages introduce the concept of window over stream, but only C-SPARQL brings the notion of continuous processing, typical of stream processing, into the language; all the other proposal still rely on permanent storing the stream before processing it using one-shot queries. Moreover, only C-SPARQL proposes an extension to SPARQL to support aggregate functions[3]. Such a support for aggregates is designed specifically to exploit optimization techniques [9] that push, whenever possible, aggregates computation as close as possible to the raw data streams.

3 Richer Form Of Stream Reasoning

We believe that C-SPARQL is an important piece of the Stream Reasoning puzzle, but supporting basic RDF entailment is only a first step toward stream reasoning.

There are many ideas of how to support incremental reasoning on the different levels of complexity. In particular, there are approaches for the incremental maintenance of materialized views in logic [31], object-oriented [22] and graph databases [32], for extensions of the well known RETE algorithm for incremental rule-based reasoning [17, 10] and even some initial ideas of how to support incremental reasoning in description logics [14, 23].

All of these methods operate incrementally, thus they are appropriate to treat information that changes, but none of them is explicitly dedicated to process an (almost) continuous flow of information with the recent information being more relevant. In [8], we presented a first attempt in this direction. We describe a novel approach for incremental maintenance of materialization of ontological entailments that makes it possible to process C-SPARQL queries on entailment regimes more expressive than basic RDF.

The novelty of such approach is in extending incremental reasoning techniques with the notion of *window* and *continuous processing*.

Window Traditional reasoning problems are based on the idea that all the information available should be taken in to account when solving the problem. In stream reasoning, we eliminate this principle and restrict reasoning to a certain window of concern which consists of a subset of statement recently observed on the stream while previous information is ignored. This is necessary for different reasons. First of all, ignoring older statements allows us to saves computing resources in terms of memory and processing time to react to important events in real time. Further, in many real-time applications there is a silent assumption that older information becomes irrelevant at some point.

[3] For a comparison between C-SPARQL and SPARQL 1.1 support for aggregates see [7]

Continuous Processing Traditional reasoning approaches are based on the idea that the reasoning process has a well defined beginning (when a request is posed to the reasoner) and end (when the result is delivered by the system). In stream reasoning, we move from this traditional model to a continuous processing model, where requests in terms of reasoning goals are registered at the reasoner and are continuously evaluated against a knowledge base that is constantly changing.

We believe that the same notions can be applied to rule-based and description logic reasoning; the results being richer and scalable forms of stream reasoning.

Moreover, a lot of progress has been made recently in terms of distributed and parallel processing as a means for scaling up logical reasoning about Semantic Web data. The focus of most of the approaches is on distributing RDF schema reasoning as the possibility of implementing it in terms of a number of simple deduction rules eases distribution. Existing approaches include the use of Peer-to-peer architectures as a basis for distributed RDF Schema reasoning [1, 21] and more recently the use of the Map-Reduce paradigm [30]. The only attempt so far to support distributed reasoning on expressive background knowledge is [26].

The next steps are to put these different ideas together and engineer an experimental stream reasoning system to explore the possibilities and problems of the approach. Investigating architectural approaches to support this richer forms of stream reasoning is a crucial step.

4 Stream Reasoning Architecture

In this section, we look at possible approaches for supporting stream reasoning from an architectural point of view. We start with illustrating our original [15] conceptual view on a stream reasoner, we then discuss the problem of dealing with the complexity of reasoning in the face of rapidly changing data and propose the notion of *cascading reasoners* as a way to cope with this complexity. We then describe how the idea of a cascade of reasoners can be implemented by connecting different stream reasoners to form a network of reasoners also providing the basis for scaling up stream reasoning by *distribution and parallelization*.

4.1 Conceptual view on a stream reasoner

Figures 1 and 2 show a conceptual view on a stream reasoning system as envisioned in our first work [15]: the former describes input and output, whereas the latter presents the information processing steps in a stream reasoner.

4.1.1 Input and Output

The system takes possibly multiple data streams as well as possibly multiple static - or better infrequently changing - knowledge models as input.

Similar to a Stream Data Management System, users can register reasoning tasks at a stream reasoner. Such a task will normally consist of a goal predicate that is to

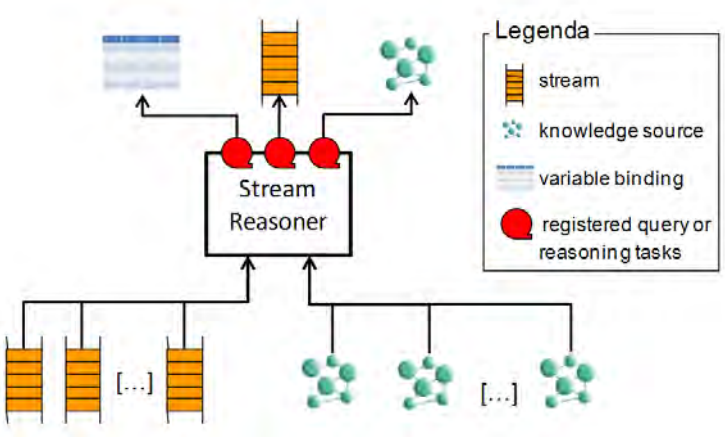

Figure 1: Conceptual view on a stream reasoner

be instantiated. The definition of this target predicate as well as additional information necessary to perform derivations can be considered as static knowledge models that are input to the system.

The stream reasoner now tries to find instantiations of the different goal predicates by applying a certain reasoning method it implements. There are different ways instantiated goal predicates are returned by the reasoner. We envision the scenario where a static answer is returned. Such an answer is either a variable bindings easy processable by any program or a knowledge model that can be further processed by any reasoner. In both these cases, such a answer has to be considered as an instantaneous results that will soon expire. For this reason, the stream reasoning task has also to specify the time range in which such a answer is valid. As an alternative, the stream reasoner could also produce a stream as a result itself. In this case, instantiations of the goal predicates are added to the output stream as they are derived. Such a streaming answer is appropriate for further processing by other stream reasoners.

4.1.2 Information Processing Steps

In Figure 2, we present the information processing steps necessary to process data streams together with a rich background knowledge.

The top part of the figure represents the problem space, whereas the the bottom part of the figure represents the four information processing steps that we consider for Stream Reasoning[4] Streaming information flows flow from left to right. Background knowledge enters the system at the reasoning step.

Data arrives to the Stream Reasoner as streamed input. A first step **selects** the relevant data in the input stream by exploiting the *window-processing* [2] and *load-*

[4]These steps are inspired by the pluggable algorithmic framework of the LarKC platform for Web scale reasoning [18].

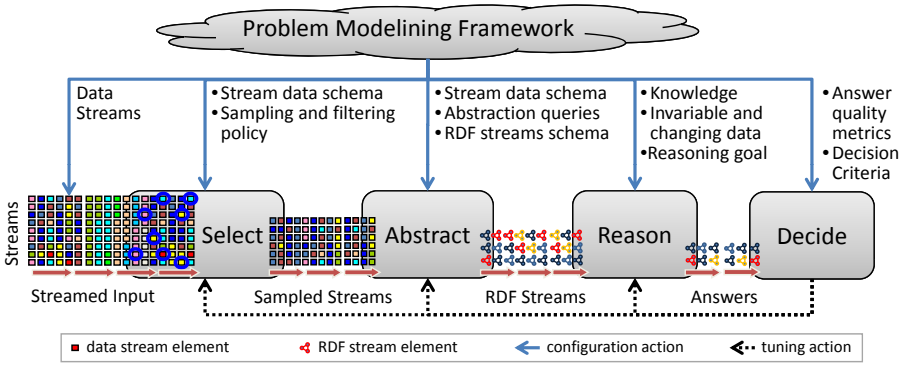

Figure 2: Information processing steps in a stream reasoner [15]

shedding techniques [28]. Such techniques were developed to deal with bursty streams that may have unpredictable peaks during which the load may exceed available system resources. A *window* extracts from the stream the *last* data stream elements, which are considered by the query. Such extraction can be *physical* (a given number of triples) or *logical* (all the triples which occur during a given time interval, the number of which is variable over time). Windows are explicitly specified in registering a reasoning task to the Stream Reasoner. *Load-shedding* techniques probabilistically drop stream data elements as they are input to the selection step based on a set of sampling policies. Sampling policies can be either specified explicitly by the system administrator at stream-registration time, or inferred by gathering statistics over time.

The sampled streams resulting from the selection step are fed into a second step that **abstracts** from fine grain data streams into aggregated events. Such step processes high volume frequently changing streaming information with high performing techniques that inject more abstract information, which changes at a lower frequency, in the following processing steps.

Aggregation queries and data compression techniques are exploited in the abstraction step. By **abstraction query** we mean, a continuous query that, given a large set of (possibly) unrelated low-level data in the input streams, produces an aggregated event in the output stream. *Data compression* techniques includes the usage of histograms [29] or wave-lets [13], when the abstraction is meant to be an aggregation (e.g., counting the number of cars running through traffic detectors), and using Bloom filters [11] for duplicate elimination, set difference, or set intersection. Either in the abstraction step or immediately after, data streams are consolidated as **RDF streams** [15, 6, 9]. RDF streams are new data formats set at the confluence of relational data streams and of conventional atoms usually injected into reasoners (i.e., RDF triples).

In the **reasoning** step, RDF streams generated by the previous steps are injected in the background knowledge in order to performe the registered reasoning tasks. We envision multiple reasoning techniques to be cascade within this step (see Section 4.2 for further discussion).

As last step, before producing the answer to the registered reasoning tasks, the answering process reaches the **decision** step. In such step quality metrics and decision criteria, defined by the application developer, are used to check if the quality of the answer is good enough and to adapt the behavior of each step (e.g., changing the sampling frequency).

4.2 Cascading Reasoners

A fundamental problem of the concept of a stream reasoner is the fact that many relevant reasoning methods, e.g. for description logics are not able to deal with high frequency data streams. While they try to derive instantiations of the goal predicate, newly incoming data will pile up eventually crashing the system. To avoid this problem we have to be aware that there is a trade-off between the complexity of the reasoning method and the frequency of the data stream the reasoner is able to handle without running into problems.

A possible way of solving this problem is to avoid feeding high frequency data directly into a complex reasoning engine. Instead, a stream reasoning system should consist of a hierarchy of processing steps of increasing complexity. This idea is an analogy to the well known memory hierarchy in computer systems where the conflict between memory size and access time is approached using the same principle. Figure 3 illustrates this idea of 'cascading reasoners' for processing streaming data.

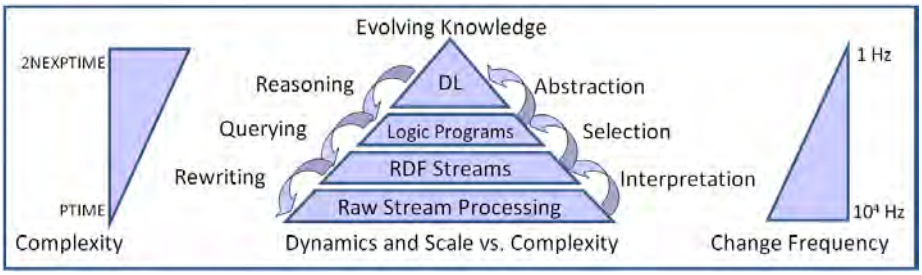

Figure 3: Cascading processors

On the lower levels raw data streams will not be fed directly into a reasoner, but into a stream data management system designed to cope with large amounts of streaming data. This stream data management system plays two roles:it wraps the data stream into a virtual RDF data model and it provides the possibility to query this virtual RDF stream using a dedicated query language such as C-SPARQL [5]. Only the results of queries against this system which will a data stream consisting of query results in RDF that arrive in a much lower frequency as only certain parts of the raw stream match the registered queries will be passed on to the higher levels. On the next higher levels, relatively simple but efficient reasoning methods including RDF schema and simple rule based reasoning will be used to further process the result stream. At this level, database techniques such as methods for maintaining materialized views can be employed to guarantee efficiency. As mentioned above, these reasoners will compute

instantiations of certain goal predicates. Only the results of these instantiations - which again can be assumed to be computed in a lower frequency than query answers - will be passed on a a data stream for the next higher levels. Only at the top of the hierarchy where the frequency of change has been reduced significantly, we can expect to be able to use expressive reasoners, e.g. for description logics or spatio-temporal reasoning. Another advantage of this layered approach is the possibility to push down processing steps in the hierarchy to speed up reasoning. More specifically, it has been shown that description logic reasoning to some extend can be reduced to rule-based reasoning [20] and that rule-based reasoning, in turn, can be reduced to query processing under certain conditions [8]. Using these techniques, we can make sure that only inferences that cannot be carried out on the lower layers of the hierarchy are actually carried out using more expressive reasoning methods.

4.3 Networks of Reasoners

The cascade of reasoners discussed in the previous section is a rather conceptual view on a stream reasoning system. On the system level, this idea can be implemented by linking individual stream reasoning systems as discussed above into a network of reasoners implementing different kinds of methods. Figure 4 shows a possible combination of different reasoners into a network.

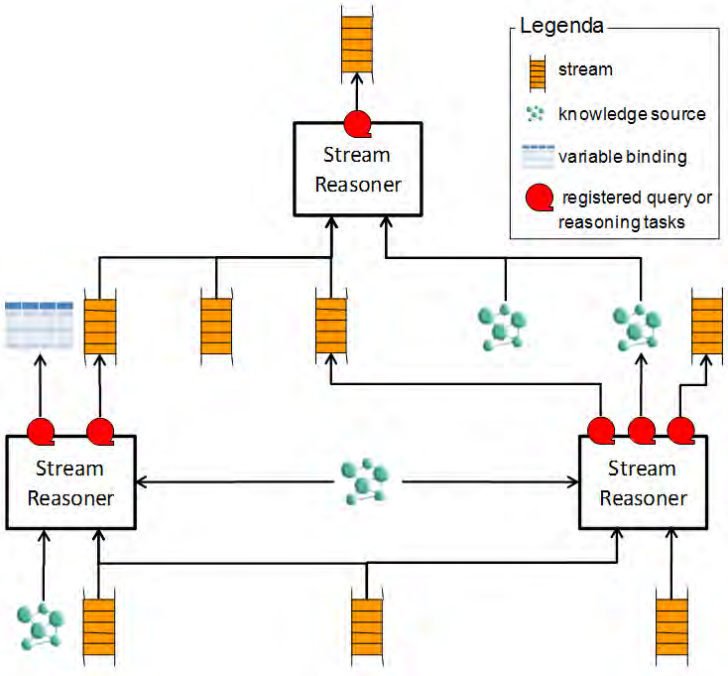

Figure 4: A network of stream reasoners

The figure illustrates different characteristic features of the concept of efficient stream reasoning. Let us assume that the streams at the bottom of the figure are high frequency data streams - in this case the stream reasoners in the lower half of the figure will be C-SPARQL engines that directly operate on these data streams and evaluate registered queries on the streams at the same time linking them to an RDF schema represented by the knowledge model shown in the center of the lower part of the figure. As one result each of the engines produces an RDF stream that serves as input for the stream reasoning engine in the upper part of the figure. This engine could implement more complex reasoning methods on top of the result streams. Further, it possible to include low frequency data streams directly at this level.

In this setting, the idea of pushing down processing in the hierarchy can be implemented by allowing not only users to register queries at reasoners, but also allow reasoners higher up in the hierarchy to automatically register queries at reasoners further down in the hierarchy.

The figure shows another advantage of implementing cascading reasoners in terms of a network: it enables distributed and parallel processing at all levels. Stream reasoning engines can be located close to the origin of the stream and different data streams can be handled by different C-SPARQL engines reducing scalability problems.

5 Links to the semantic sensor network activity

We consider stream reasoning to be a complementary but currently missing aspect of the idea of the semantic sensor web [27]. The idea of the semantic sensor web is to use semantic web techniques for providing processable metadata about sensors acquiring real time information about the physical world to support the analysis of acquired data. Currently a W3C incubator group on this topic is developing supporting standards, in particular a general sensor ontology that can be extended to cover different domains and linked to existing standards as well as a semantic sensor markup language for annotating sensors using terms from the ontology. The current view on semantic sensor networks is a rather static one that focusses on metadata assigned to sensors and reasoning about this static data to support use cases such as the discovery of relevant sensors and data. We see two obvious relations of the concept of stream reasoning to this initiative that are described in the following.

5.1 Online Reasoning on Streams and Metadata

Being able to reason on data streams provides new possibilities concerning the idea of semantic sensor nets. In particular, it can help to enable online processing of semantic annotations. At the moment, the idea of sensor markup languages is that reasoning takes place offline in order to identify sensors or data that is then used. Stream reasoning on the other hand allows us to perform online reasoning on data streams and sensor metadata. In this case the semantic description of the sensor - e.g. the measured phenomenon - can be seen as a static knowledge base that acts as input to the stream reasoner which automatically links the streaming information it processes to this metadata - e.g. by adding a 'type' statement to every measurement arriving that links the

raw data to the measured phenomenon. This dynamic way of linking data with metadata is currently not possible and has to be done a posteriori leading to an unnecessary delay in the processing model.

5.2 Reasoners as Sensors

As mentioned above, we assume that stream reasoners are also able to produce data streams as output. This output data stream can be fed to other stream reasoners or published as data feed. In both cases, the stream reasoner itself can be seen as an advanced sensor able to produce high level sensor data. Similar to conventional sensors, such a reasoner could benefit from the existence of processable metadata describing the sensor type (in this case some kind of reasoner), the measured phenomenon (the goal predicates) etc. As for conventional sensors, such metadata will support the discovery of interesting stream reasoners and help to interpret the output stream.

6 Conclusions

We believe that providing the ability to reason about streaming data to cope with the increasing amount of dynamic data on the web is the next big step in semantic technologies. In this paper, we have presented some initial ideas concerning the concept of stream reasoning. It is clear that these ideas that are mostly at the conceptual level have to be backed up by a formal theory of reasoning about streaming information. The development of such a theory is a major challenge, but work in this direction can be based on a number of theoretical results on individual parts of the whole picture.

Languages like C-SPARQL and their corresponding infrastructures provide an excellent starting point for further investigating expressive reasoning over data streams. By providing a uniform RDF-based representation of heterogeneous streams, C-SPARQL partially solves the challenge of providing efficient access protocols for heterogeneous streams. Further, the extraction of patterns from data streams is subject of ongoing research in machine learning. For instance, results from statistical relational learning are able to derive classification rules from example data in a very effective way. There are two lines of research that follow immediately. The first is to link relational learning methods with infrastructures like C-SPARQL to facilitate pattern extraction on top of streaming RDF data, the second is the extension of existing reasoning mechanisms to support continuous processing. In this direction first steps have been made in connection with incremental maintenance of RDF views. These approaches need to be generalized to more expressive languages. Once the problem of continuous reasoning with richer models has been understood sufficiently, the remaining challenges in terms of scaling up to very high data volumes can be investigated and approaches like distributed and parallel processing as well as the compilation of continuous queries into weaker formalism can be approached.

References

[1] Philippe Adjiman, Francois Goasdou, and Marie-Christine Rousset. Somerdfs in the semantic web. *Journal of Data Semantics (JoDS)*, 8:158–181, 2007.

[2] Arvind Arasu, Shivnath Babu, and Jennifer Widom. The CQL Continuous Query Language: Semantic Foundations and Query Execution. *The VLDB Journal*, 15(2):121–142, 2006.

[3] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *The 21 st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems(PODS 2002)*, pages 1–16, 2002.

[4] Davide Francesco Barbieri, Daniele Braga, Stefano Ceri, Emanuele Della Valle, and Michael Grossniklaus. C-SPARQL: SPARQL for Continuous Querying. In *Proc. Intl. Conf. on World Wide Web (WWW)*, pages 1061–1062, 2009.

[5] Davide Francesco Barbieri, Daniele Braga, Stefano Ceri, Emanuele Della Valle, and Michael Grossniklaus. C-sparql: Sparql for continuous querying. In *Proceedings of the 18th International Conference on World Wide Web, WWW 2009*, pages 1061–1062, 2009.

[6] Davide Francesco Barbieri, Daniele Braga, Stefano Ceri, Emanuele Della Valle, and Michael Grossniklaus. Continuous queries and real-time analysis of social semantic data with c-sparql. In *Proceedings of Social Data on the Web Workshop at the 8th International Semantic Web Conference*, 10 2009.

[7] Davide Francesco Barbieri, Daniele Braga, Stefano Ceri, Emanuele Della Valle, and Michael Grossniklaus. Feedbacks on sparql 1.1 support for aggregates. Technical report, LarKC Project., February 2010. Available on line at http://wiki.larkc.eu/c-sparql/sparql11-feedback.

[8] Davide Francesco Barbieri, Daniele Braga, Stefano Ceri, Emanuele Della Valle, and Michael Grossniklaus. Incremental reasoning on streams and rich background knowledge. In Lora Aroyo, Grigoris Antoniou, and Eero Hyvonen, editors, *ESWC*, Lecture Notes in Computer Science. Springer, 2010.

[9] Davide Francesco Barbieri, Daniele Braga, Stefano Ceri, and Michael Grossniklaus. An Execution Environment for C-SPARQL Queries. In *Proc. Intl. Conf. on Extending Database Technology (EDBT)*, 2010.

[10] B. Berster. Extending the rete algorithm for event management. In *Proceedings of the Ninth International Symposium on Temporal Representation and Reasoning (TIME'02)*, 2002.

[11] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

[12] Andre Bolles, Marco Grawunder, and Jonas Jacobi. Streaming SPARQL – Extending SPARQL to Process Data Streams. In *Proc. Europ. Semantic Web Conf. (ESWC)*, pages 448–462, 2008.

[13] Kaushik Chakrabarti, Minos Garofalakis, Rajeev Rastogi, and Kyuseok Shim. Approximate query processing using wavelets. *The VLDB Journal*, 10(2-3):199–223, 2001.

[14] B. Cuenca-Grau, C. Halaschek-Wiener, and Y. Kazakov. History matters: Incremental ontology reasoning using modules. In *Proceedings of the 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007*, volume 4825 of *Lecture Notes in Computer Science*, pages 183–196, 2007.

[15] Emanuele Della Valle, Stefano Ceri, Davide Francesco Barbieri, Daniele Braga, and Alessandro Campi. A First Step Towards Stream Reasoning. In *Proc. Future Internet Symposium (FIS)*, pages 72–81, 2008.

[16] Emanuele Della Valle, Stefano Ceri, Frank van Harmelen, and Dieter Fensel. It's a Streaming World! Reasoning upon Rapidly Changing Information. *IEEE Intelligent Systems*, 24(6):83–89, 2009.

[17] F. Fabret, M. Regnier, and E. Simon. An adaptive algorithm for incremental evaluation of production rules in databases. In *Proceedings of the 19th International Conference on Very Large Data Bases*, pages 455 – 466, 1993.

[18] Dieter Fensel, Frank van Harmelen, Bo Andersson, Paul Brennan, Hamish Cunningham, Emanuele Della Valle, Florian Fischer, Zhisheng Huang, Atanas Kiryakov, Tony Kyung il Lee, Lael Schooler, Volker Tresp, Stefan Wesner, Michael Witbrock, and Ning Zhong. Towards larkc: A platform for web-scale reasoning. In *ICSC*, pages 524–529. IEEE Computer Society, 2008.

[19] Peter Gaerdenfors, editor. *Belief Revision*. Cambridge University Press, 2003.

[20] Benjamin Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. Description logic programs: combining logic programs with description logic. In *Proceedings of the 12th international conference on World Wide Web*, pages 48 – 57, 2003.

[21] Zoi Kaoudi, Iris Miliaraki, and Manolis Koubarakis. Rdfs reasoning and query answering on top of dhts. In *7th International Semantic Web Conference (ISWC 2008)*, 2008.

[22] H.A. Kuno and E.A. Rundensteiner. Incremental maintenance of materialized object-oriented views in multiview: Strategies and performance evaluation. *IEEE Transactions on Data and Knowledge Engineering*, 10(5):768–792, september 1998.

[23] Bijan Parsia, Christian Halaschek-Wiener, and Evren Sirin. Towards incremental reasoning through updates in owl-dl. In *Proceedings of the Reasoning on the Web Workshop at WWW2006, Edinburgh, UK*. 2006.

[24] Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF. `http://www.w3.org/TR/rdf-sparql-query/`.

[25] Alejandro Rodriguez, Robert McGrath, Yong Liu, and James Myers. Semantic Management of Streaming Data. In *Proc. Intl. Workshop on Semantic Sensor Networks (SSN)*, 2009.

[26] Anne Schlicht and Heiner Stuckenschmidt. Distributed resolution for expressive ontology networks. In *Web Reasoning and Rule Systems*, 2009.

[27] Amit Sheth, Cory Henson, and Satya S. Sahoo. Semantic sensor web. *IEEE Internet Computing*, 12(4):78–83, 2008.

[28] Nesime Tatbul, Uğur Çetintemel, Stan Zdonik, Mitch Cherniack, and Michael Stonebraker. Load shedding in a data stream manager. In *vldb'2003: Proceedings of the 29th international conference on Very large data bases*, pages 309–320. VLDB Endowment, 2003.

[29] Nitin Thaper, Sudipto Guha, Piotr Indyk, and Nick Koudas. Dynamic multidimensional histograms. In *SIGMOD '02: Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 428–439, New York, NY, USA, 2002. ACM.

[30] Jacopo Urbani, Spyros Kotoulas, Eyal Oren, and Frank van Harmelen. Scalable distributed reasoning using mapreduce. In *Proceedings of the ISWC '09*, volume 5823 of *LNCS*. Springer, 2009.

[31] Raphael Volz, Steffen Staab, and Boris Motik. Incrementally maintaining materializations of ontologies stored in logic databases. *J. Data Semantics*, 2:1–34, 2005.

[32] Y. Zhuge and H. Garcia-Molina. Graph structured views and their incremental maintenance. In *Proceedings of the Fourteenth International Conference on Data Engineering*, pages 116 – 125, 1998.