A Proposal for Publishing Data Streams as Linked Data

- A Position Paper -

Davide F. Barbieri
Dipartimento di Elettronica e Informazione
Politecnico di Milano
Piazza L. da Vinci 32, 20133 Milano
dbarbieri@elet.polimi.it

Emanuele Della Valle
Dipartimento di Elettronica e Informazione
Politecnico di Milano
Piazza L. da Vinci 32, 20133 Milano
dellavalle@elet.polimi.it

ABSTRACT

Streams are appearing more and more often on the Web in sites that distribute and present information in real-time streams. We anticipate a rapidly growing need of mashing up this streaming information with more static one. While best practices for linking static data on the Web were published and facilitate the mash up of static information published on the Web, streams were neglected. In this short position paper, we propose an approach to publish Data Streams as Linked Data.

Keywords

Data Streams, Linked Data, Virtual RDF, Stream Reasoning

1. INTRODUCTION

A growing number of Web sites are distributing and presenting information in real-time streams. Microblogs such as Twitter[1], weather monitoring site such as AccuWeather[2], traffic monitoring sites such as Waze[3] are few representative examples.

Streams, being unbounded sequences of time-varying data elements, should not be treated as persistent data to be stored (forever) and queried on demand, but rather as transient data to be consumed on the fly by continuous queries. Continuous queries, after being registered, keep analyzing such streams, producing answers triggered by the streaming data and not by explicit invocation. Such a paradigmatic change have been largely investigated in the last decade by the database community [15]. Specialized Data Stream Management Systems (DSMS) have been developed (e.g., STREAM [2], Aurora/Borealis [1] and Stream Mill [6]). Several startups such as StreamBase[4] are commercializing DSMS, and features of DSMS are becoming supported by major database products, such as Oracle and DB2.

Motivated by the availability of real-time streams on the Web and by the lack of Web-based approaches to process them, we have been working since 2008 on an extension to SPARQL[20] for continuous querying over streams of RDF and static RDF graphs (namely C-SPARQL [7, 9]).

[1] http://twitter.com/
[2] http://www.accuweather.com/
[3] http://world.waze.com/
[4] http://www.streambase.com/

LDOW2010, April 27, 2010, Raleigh, North Carolina.

Listing 1 shows an example of C-SPARQL query that, given a static description of brokers and a stream of financial transactions for all brokers, computes the amount of transactions for Swiss brokers within the last hour.

```
1  REGISTER STREAM TotalAmountPerBroker COMPUTE EVERY 10m AS
2  PREFIX ex: <http://example/>
3  CONSTRUCT {?broker ex:hasTotalAmount?total .}
4  FROM <http://brokerscentral.org/brokers.rdf>
5  FROM STREAM <http://stockex.org/market.trdf>
6    [RANGE 1h STEP 10m]
7  WHERE {
8          ?broker ex:from ?country .
9          ?broker ex:does ?tx .
10         ?tx ex:with ?amount .
11     FILTER (?country = "CH" )
12 }
13 AGGREGATE { (?total, SUM(?amount), ?broker) }
```

Listing 1: Example of C-SPARQL which allows dealing with streams of RDF triples as well as static RDF graphs

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 an RDF stream. The COMPUTE EVERY clause states the frequency of every new computation, in the example every 10 minutes. At line 5, the clause FROM STREAM defines the RDF stream of financial transactions, used within the query. Next, line 6 defines the **window** of observation of the RDF stream. 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 RDF triples produced in the last 1 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. Finally, at line 13, the AGGREGATE function asks the C-SPARQL engine to include in the result set a new variable ?total which is bound to the sum of the amount of the transaction of each broker.

Our C-SPARQL Engine [9] treats non-RDF DSMSs as virtual RDF streams and graphs. It allows to register queries that continuously combine (virtual) RDF streams and RDF graphs. Under this respect, our C-SPARQL Engine is similar to D2RQ [12] that treats non-RDF databases as virtual RDF graphs. In our previous works [7, 8, 9] we develop an engine for registering and continuously executing C-SPARQL queries. With this position paper, we propose an extension of our C-SPARQL Engine that publishes data streams as Linked Data. Such an extension complements the work done so far and lowers the entry barrier for external

(Semantic) Web application to consume data streams.

The rest of the paper is organized as follows. In Section 2 we describe the design principles that inspire our proposal for *Streaming Linked Data*. Section 3 explains how to publish a single data stream as an RDF stream. In the same section we also present a vocabulary to describe the time interval in which the published data are valid. The URI schema that allows to control the Window behavior is presented in Section 4. In Section 5, we describe the RESTful [21] services which allow to control the C-SPARQL query that continuously computes the published RDF stream. Finally, Section 6 and 7 present some related work and draw some conclusions, respectively.

2. DESIGN PRINCIPLE

The design principle that inspires our approach is illustrated in Figure 1. Our C-SPARQL engine is able to process data streams and RDF streams in combination with RDF graphs. In our previous work, we use in memory connection between our C-SPARQL engine and local C-SPARQL clients. However, we anticipate a rapidly growing need of mashing up results of our C-SPARQL engine with SPARQL- and RDF-based linked data clients. A Streaming Linked Data Server is a special local C-SPARQL Client that connects in memory to a C-SPARQL engine and exposes as Linked Data the results of continuous queries registered in the C-SPARQL engine.

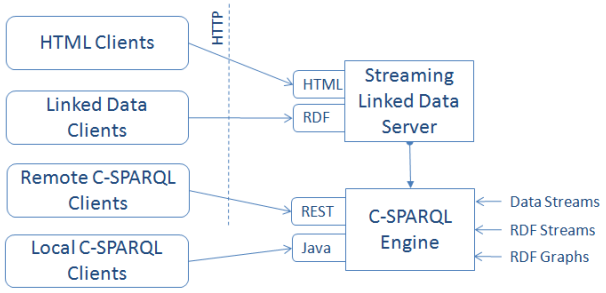

Figure 1: Architectural solution of our approach to publish Streaming Linked Data

By using our C-SPARQL engine as a one-to-one mapper from data streams to RDF streams, we can make available to Linked Data Clients a raw data stream (see Section 3). Moreover, we offer an interface to remotely control the behavior of the window which the stream is observed through (see Section 4). Finally, we make available RESTful services that implement a remote C-SPARQL Client (see Section 5). Such services provide full control (i.e, beyond window behavior) on the C-SPARQL queries whose results are served as Linked Data by the Streaming Linked Data Server.

3. PUBLISHING A STREAM

A data stream is defined as an ordered sequence of pairs, where each pair is made of a tuple and its timestamp τ. For instance, the stream of financial transactions used in the example in Listing 1 could contain a transaction `tr1` done by `broker1` for $ 1000 registered at τ_i, and two transactions at τ_{i+1}: `tr2` done by `broker1` for $ 3000 and `tr3` done by `broker2` for $ 2000.

$$(\langle Transaction(tr1, broker1, \text{"\$1000"})\rangle\ ,\ \tau_i)$$
$$(\langle Transaction(tr2, broker1, \text{"\$3000"})\rangle\ ,\ \tau_{i+1})$$
$$(\langle Transaction(tr3, broker2, \text{"\$2000"})\rangle\ ,\ \tau_{i+1})$$

In a similar way, we define an RDF stream [7] as an ordered sequence of pairs, where each pair is made of an RDF triple and its timestamp τ. By mapping the data stream above in RDF using D2RQ mapping language [10], we obtain the following RDF stream:

$$(\langle broker1\ does\ tr1\ .\rangle\ ,\ \tau_i)$$
$$(\langle tr1\ with\ \text{"\$1000"}\ .\rangle\ ,\ \tau_i)$$
$$(\langle broker1\ does\ tr2\ .\rangle\ ,\ \tau_{i+1})$$
$$(\langle tr2\ with\ \text{"\$3000"}\ .\rangle\ ,\ \tau_{i+1})$$
$$(\langle broker2\ does\ tr3\ .\rangle\ ,\ \tau_{i+1})$$
$$(\langle tr3\ with\ \text{"\$2000"}\ .\rangle\ ,\ \tau_{i+1})$$

We propose to represent RDF streams in RDF using *named graphs* [13]. We distinguish between two kind of named graphs: the *Stream Graphs* (shortly s-graphs) and the *Instantaneous Graphs* (shortly i-graphs). In our proposal, an RDF Stream can be represented using one s-graph and several i-graphs, one for each timestamp.

A s-graphs is a metadata graph that describes the current content of the window over the RDF Stream. The most important part of an s-graph are the triples that refer to the i-graphs using `rdfs:seeAlso`[5] and those that describe when each i-graph was received using the property `receivedAt`.

Few other metadata complete the description of an s-graph. The property `lastUpdate` describes the last time the graph was updated. The property `expires` allows to indicate a Linked Data Client that the information in the graph will expire in a given moment in future. The properties `sld:windowType` and `windowSize` describe the window through which the stream is observed (see Section 4 for more information).

For instance, if the data stream exemplified above was the current content of a window over the stream of financial transactions, it can be represented using the s-graph in Listing 2 and the two i-graphs in Listing 3 and 4.

```
1  @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
2  @prefix sld:   <http://www.streaminglinkeddata.org/schema#> .
3  @prefix : <http://example/> .
4
5  :sgraph1 sld:lastUpdate "τ_{i+1}"^^xsd:dataTime ;
6           sld:expires "τ_{i+2}"^^xsd:dataTime ;
7           sld:windowType sld:logicalTumbling ;
8           sld:windowSize "PT1H"^^xsd:duration .
9
10 :sgraph1 rdfs:seeAlso :igraph1 .
11 :igraph1 sld:receivedAt "τ_i"^^xsd:dataTime .
12
13 :sgraph1 rdfs:seeAlso :igraph2 .
14 :igraph2 sld:receivedAt "τ_{i+1}"^^xsd:dataTime .
```

Listing 2: Example of Stream Graph linking two Instantaneous Graphs

```
1  @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
2  @prefix sld:   <http://www.streaminglinkeddata.org/schema#> .
3  @prefix : <http://example/> .
4
5  :igraph1 sld:receivedAt "τ_i"^^xsd:dataTime ;
6           rdfs:seeAlso :sgraph1 .
7
8  :broker1 :does :tr1 .
9  :tr1 :with "$ 1000" .
```

Listing 3: The Instantaneous Graph timestamped with τ_i.

[5]We choose to link s-graphs to i-graphs using the property `rdfs:seeAlso`, because it has been largely adopted to link named graphs (see for instance the usage of `rdfs:seeAlso` in Sindice [19] and in the Semantic Web Client [17])

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix sld:   <http://www.streaminglinkeddata.org/schema#> .
@prefix : <http://example/> .

:igraph2 sld:receivedAt "τ_{i+1}"^^xsd:dataTime ;
         rdfs:seeAlso :sgraph1 .

:broker1 :does :tr2 .
:tr2 :with "$ 3000" .
:broker2 :does :tr3 .
:tr3 :with "$ 2000" .
```

Listing 4: The Instantaneous Graph timestamped with τ_{i+1}.

Following the guidelines on *cool URIs* [5], we propose to give to s-graphs and i-graphs an IRI using the following schemata:

```
s-graph: http://ex.org/%stream-name%
         e.g., http://stockex.org/transactions
i-graph: http://ex.org/%stream-name%/URLeconde(%timestamp%)
         e.g., http://stockex.org/transactions/2010-02-12T13%3A34%3A41Z
```

Moreover, following the best practice on how to publish Linked Data on the Web [11] in terms of content negotiation, when IRIs, which follow the schemata shown above are dereferenced, the Streaming Linked Data Server dereferences an information resource appropriate for the client (using HTTP content negotiation):

- Linked Data Clients are redirected to

```
http://ex.org/trdf/%stream-name%
http://ex.org/trdf/%stream-name%/URLeconde(%timestamp%)
```

- HTML Clients are redirected to

```
http://ex.org/page/%stream-name%
http://ex.org/page/%stream-name%/URLeconde(%timestamp%)
```

4. CONTROLLING THE WINDOW

As we have explained in the previous section, streams are intrinsically infinite. In C-SPARQL, we introduce the notion of windows over streams. In Section 3, we focus on the general approach to publish a data stream rather than on the notion of window. However, we foresee the need for a consumer of Streaming Linked Data to be able to control the behavior of the window through which the stream is observed.

Types and characteristics of windows in C-SPARQL are inspired by those of the windows defined in continuous query languages for relational streaming data, such as CQL[3]. Windows are expressed in C-SPARQL within the `FROM STREAM` clause, whose syntax is as follows:

FromStrClause ⟶ 'FROM' ['NAMED'] 'STREAM' *StreamIRI* '[RANGE' *Window* ']'
Window ⟶ *LogicalWindow* | *PhysicalWindow*
LogicalWindow ⟶ *Number TimeUnit WindowOverlap*
TimeUnit ⟶ 'ms' | 's' | 'm' | 'h' | 'd'
WindowOverlap ⟶ 'STEP' *Number TimeUnit* | 'TUMBLING'
PhysicalWindow ⟶ 'TRIPLES' *Number*

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).

Logical windows are *sliding* [16] when they are progressively advanced of a given `STEP` (i.e. a time interval that is shorter than the window's time interval); they are *non-overlapping* (or `TUMBLING`) when they are advanced of exactly their time interval at each iteration. With tumbling windows every triple of the stream is included exactly into one window, whereas with sliding windows some triples can be included into several windows.

We believe that consumers of Streaming Linked Data would largely benefit from controlling the window of a running C-SPARQL query. Therefore we propose the following IRI schemata:

- physical windows can be controlled replacing **`%size%`** with the number of triples (e.g., the last 1000 triples)

```
Schema:  http://ex.org/%stream-URI%/physical/%size%
Example: http://stockex.org/transactions/physical/1000
```

- logical windows can be controlled replacing **`%size%`** with the a time interval[6] (e.g., `PT1H` meaning 1 hour) and replacing **`%step%`** either with the keyword `tumbling` or with a time interval (e.g., `PT10M` meaning 10 minutes).

```
Schema:  http://ex.org/%stream-URI%/logical/%size%/%step%
Example: http://stockex.org/transactions/logical/PT1H/PT10M
```

Notably, each of these IRIs are translated to an equivalent C-SPARQL query that processes the data stream. For instance, the example above is equivalent to the following C-SPARQL query.

```
REGISTER STREAM transactions COMPUTE EVERY 10m AS
PREFIX : <http://example/>
CONSTRUCT *
FROM STREAM <http://stockex.org/market.trdf>
  [RANGE 1h STEP 10m]
WHERE { ?s ?p ?o . }
```

5. CONTROLLING C-SPARQL QUERIES

In this Section, we describe the RESTful [21] services which allow one to control each C-SPARQL query that continuously computes each RDF stream published with our approach.

As we explained above, C-SPARQL queries have to be *registered* in the C-SPARQL Engine. As soon as a query is registered, the C-SPARQL engine *starts* to compute it. An explicit *stop* command is required to stop the processing of a registered query. Similarly an *unregister* command allows for deleting a C-SPARQL query.

We desinged a RESTful interface that uses the HTTP methods to controll the C-SPARQL queries:

- `PUT`, with a C-SPARQL query as parameter, allows to register a query that generates a certain RDF stream,
- `POST`, with `start` or `stop` command as parameters, is used to start or stop a registered query, and
- `DELETE` can be used to unregister a query.

6. RELATED WORK

Two previous works [14, 22] address the need for publishing data streams as Linked Data.

In [14], Corcho introduce the concept of Linked Stream Data, a way in which the Linked Data principles can be

[6]The lexical space of such an interval is the same as xsd:duration, i.e., the format `PnYnMnDTnHnMnS` defined by ISO 8601 [18]

applied to stream data and be part of the Web of Linked Data. At a first glance, his proposal could appear similar to our one. Both his and our proposal use named graphs and define IRI schemata. However, his approach does not take into account the nature of streams, that, being unbounded sequences of time-varying data elements, should not be treated as persistent data to be stored (forever) and queried on demand, but rather as transient data to be consumed on the fly by continuous queries. His proposal allows for opening a window starting from and ending into any moment in time (see listing below). This is incompatible with the principle to keep a window open on the latest data that has to be consumed on the fly. It requires the Linked Stream Data server to store the stream for an indefinite time period.

```
http://www.domain.org/sensor/name/%start time%,%end time%
```

In [22], Rodríguez et al. introduce the notion of Time-Annotated RDF (TA-RDF) that allows for representing time-series data, especially streaming data, using the Semantic Web approach. (TA-RDF) is an extension of the RDF model where resources are optionally annotated with a time value, i.e, a time-annotated resource is a pair of the form `resource[time]` (see listing below for an example).

```
<urn:OHARE> <urn:hasRainSensor> <urn:sensor1> .
<urn:sensor1>["2009-01-01Z-06:00"^^xsd:date] <urn:hasReading> "0" .
<urn:sensor1>["2009-01-01Z-06:05"^^xsd:date] <urn:hasReading> "5" .
...
<urn:sensor1>["2009-01-31Z-10:00"^^xsd:date] <urn:hasReading> "15" .
```

A TA-RDF graph can be represented as a set of RDF graphs using two special properties: `belongsTo`, which indicates a data element in a stream, and *hasTimestamp*, which points toward the timestamp of the data element.

As for the previous related work, TA-RDF proposal looks very similar to our one, but still it lacks the paradigmatic change from persistent data to transient data. In TA-RDF streams are supposed to be stored indefinitely.

Finally, the two proposal do not consider the rich types of windows proposed in DSMS. They do not propose a vocabulary to describe the window type (i.e., `lsd:physical` vs. `lsd:logical`) and the size of the window (i.e., the equivalent of our property `windowSize`). The properties `lastUpdate` and `expires`, which in our vocabulary allows to indicate a Linked Data Client when the graph was updated and when it will expire, are not present.

7. CONCLUSION

Distributing and presenting information in real-time streams is becoming a best practice on the Web. The nature of streams requires a paradigmatic change from persistent data to be stored, and queried on demand, to transient data, to be consumed on the fly by continuous queries.

In our previous work we investigated C-SPARQL as an approach to treat non-RDF DSMSs as virtual RDF streams and graphs. With this position paper, we propose an extension of our C-SPARQL Engine that publishes data streams as Linked Data. In this paper, we described the principle that inspires our approach and we explain how to publish RDF streams continuously generated by C-SPARQL queries. Such a best practice introduces the concepts of Stream Graph (or s-graph) and Instantaneous Graph (or i-graph) as well as a small vocabulary that allows to describe which part of the stream has been published and when the information will expire. A RESTful service to control the C-SPARQL queries that generates the RDF streams is also detailed.

We believe that our proposal can lower the entry barrier for external (Semantic) Web application to consume data streams. Our next step is to complete the prototypical implementation of our Streaming Linked Data Server and evaluate it against several use cases. We are currently considering the synthetic Linear Road Benchmark [4], a well established benchmark for Data Stream Management Systems, and several real source of streams that we are already experimenting with (see for instance, the social media streams in [8] or the Milan traffic streams in [9]).

8. ACKNOWLEDGMENTS

The work described in this paper has been partially supported by the European project LarKC (FP7-215535).

9. REFERENCES

[1] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Çetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. S. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik. The Design of the Borealis Stream Processing Engine. In *Proc. Intl. Conf. on Innovative Data Systems Research (CIDR 2005)*, 2005.

[2] A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, I. Nishizawa, J. Rosenstein, and J. Widom. STREAM: The Stanford Stream Data Manager (Demonstration Description). In *Proc. ACM Intl. Conf. on Management of Data (SIGMOD 2003)*, page 665, 2003.

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

[4] A. Arasu, M. Cherniack, E. F. Galvez, D. Maier, A. Maskey, E. Ryvkina, M. Stonebraker, and R. Tibbetts. Linear road: A stream data management benchmark. In M. A. Nascimento, M. T. Özsu, D. Kossmann, R. J. Miller, J. A. Blakeley, and K. B. Schiefer, editors, *VLDB*, pages 480–491. Morgan Kaufmann, 2004.

[5] D. Ayers and M. Vlkel. Cool uris for the semantic web. World Wide Web Consortium, Note NOTE-cooluris-20081203, December 2008. Available on line at: http://www.w3.org/TR/2008/NOTE-cooluris-20081203/.

[6] Y. Bai, H. Thakkar, H. Wang, C. Luo, and C. Zaniolo. A Data Stream Language and System Designed for Power and Extensibility. In *Proc. Intl. Conf. on Information and Knowledge Management (CIKM 2006)*, pages 337–346, 2006.

[7] D. F. Barbieri, D. Braga, S. Ceri, E. Della Valle, and M. Grossniklaus. C-SPARQL: SPARQL for Continuous Querying. In *Proc. Intl. Conf. on World Wide Web (WWW)*, pages 1061–1062, 2009.

[8] D. F. Barbieri, D. Braga, S. Ceri, E. Della Valle, and M. 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.

[9] D. F. Barbieri, D. Braga, S. Ceri, and M. Grossniklaus. An Execution Environment for C-SPARQL Queries. In *Proc. Intl. Conf. on Extending Database Technology (EDBT)*, 2010.

[10] C. Bizer. D2R MAP - A Database to RDF Mapping Language. In *WWW (Posters)*, 2003.

[11] C. Bizer, R. Cyganiak, and T. Heath. How to publish linked data on the web. Web page, 2007. Revised 2008. Accessed 07/08/2009.

[12] C. Bizer and A. Seaborne. D2RQ - Treating Non-RDF Databases as Virtual RDF Graphs. In *ISWC2004 (posters)*, November 2004.

[13] J. J. Carroll, C. Bizer, P. J. Hayes, and P. Stickler. Named graphs, provenance and trust. In A. Ellis and T. Hagino, editors, *WWW*, pages 613–622. ACM, 2005.

[14] O. Corcho. Linked stream data: A position paper. In *The 2nd International Workshop on Semantic Sensor Networks 2009*, 2009.

[15] M. Garofalakis, J. Gehrke, and R. Rastogi. *Data Stream Management: Processing High-Speed Data Streams (Data-Centric Systems and Applications)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[16] L. Golab and M. T. Özsu. Processing Sliding Window Multi-Joins in Continuous Queries over Data Streams. In *Proc. Intl. Conf. on Very Large Data Bases (VLDB 2006)*, pages 500–511, 2003.

[17] O. Hartig, C. Bizer, and J. C. Freytag. Executing sparql queries over the web of linked data. In A. Bernstein, D. R. Karger, T. Heath, L. Feigenbaum, D. Maynard, E. Motta, and K. Thirunarayan, editors, *International Semantic Web Conference*, volume 5823 of *Lecture Notes in Computer Science*, pages 293–309. Springer, 2009.

[18] International Organization for Standardization. Data elements and interchange formats — information interchange — representation of dates and times. ISO 8601, December 2004. Available on line at: http://xml.coverpages.org/ISO-FDIS-8601.pdf.

[19] E. Oren, R. Delbru, M. Catasta, R. Cyganiak, H. Stenzhorn, and G. Tummarello. Sindice.com: a document-oriented lookup index for open linked data. *IJMSO*, 3(1):37–52, 2008.

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

[21] L. Richardson and S. Ruby. *RESTful Web Services*. O'Reilly, Beijing, 2007.

[22] A. Rodriguez, R. McGrath, Y. Liu, and J. Myers. Semantic Management of Streaming Data. In *Proc. Intl. Workshop on Semantic Sensor Networks (SSN)*, 2009.