Web Semantics: Science, Services and Agents on the World Wide Web 16 (2012) 42–51

Contents lists available at SciVerse ScienceDirect

Web Semantics: Science, Services and Agents on the World Wide Web

journal homepage: www.elsevier.com/locate/websem

A middleware framework for scalable management of linked streams[☆]

Danh Le-Phuoc[*], Hoan Quoc Nguyen-Mau, Josiane Xavier Parreira, Manfred Hauswirth

Digital Enterprise Research Institute, National University of Ireland, Galway, Ireland

ARTICLE INFO

Article history:
Received 30 January 2012
Received in revised form
23 May 2012
Accepted 12 June 2012
Available online 23 June 2012

Keywords:
Sensors
Linked stream data
Linked data
Semantic web
Middleware

ABSTRACT

The Web has long exceeded its original purpose of a distributed hypertext system and has become a global, data sharing and processing platform. This development is confirmed by remarkable milestones such as the Semantic Web, Web services, social networks and mashups. In parallel with these developments on the Web, the Internet of Things (IoT), i.e., sensors and actuators, has matured and has become a major scientific and economic driver. Its potential impact cannot be overestimated – for example, in logistics, cities, electricity grids and in our daily life, in the form of sensor-laden mobile phones – and rivals that of the Web itself. While the Web provides ease of use of distributed resources and a sophisticated development and deployment infrastructure, the IoT excels in bringing real-time information from the physical world into the picture. Thus a combination of these players seems to be the natural next step in the development of even more sophisticated systems of systems. While only starting, there is already a significant amount of sensor-generated, or more generally dynamic information, available on the Web. However, this information is not easy to access and process, depends on specialised gateways and requires significant knowledge on the concrete deployments, for example, resource constraints and access protocols. To remedy these problems and draw on the advantages of both sides, we try to make dynamic, online sensor data of any form as easily accessible as resources and data on the Web, by applying well-established Web principles, access and processing methods, thus shielding users and developers from the underlying complexities. In this paper we describe our Linked Stream Middleware (LSM, http://lsm.deri.ie/), which makes it easy to integrate time-dependent data with other Linked Data sources, by enriching both sensor sources and sensor data streams with semantic descriptions, and enabling complex SPARQL-like queries across both dataset types through a novel query processing engine, along with means to mashup the data and process results. Most prominently, LSM provides (1) extensible means for real-time data collection and publishing using a cloud-based infrastructure, (2) a Web interface for data annotation and visualisation, and (3) a SPARQL endpoint for querying unified Linked Stream Data and Linked Data. We describe the system architecture behind LSM, provide details of how Linked Stream Data is generated, and demonstrate the benefits and efficiency of the platform by showcasing some experimental evaluations and the system's interface.

1. Introduction

Sensors are already present in a wide range of applications, for example, environmental monitoring, smart cities, smart grids, and mobile phone applications. Sensor devices are also becoming more powerful, not only generating data but also offering processing capabilities, thus becoming smart devices. Moreover, advances in network technologies, e.g., the creation of the Constrained Application Protocol (CoAP), allow these devices to communicate and publish data on the Web, fostering the Internet of Things (IoT) paradigm and narrowing the gap to the Web. It is interesting to see that the IoT is increasingly using (semantic) Web technologies, to lower the technological threshold for access to such information, for example, CoAP, which is essentially HTTP for resource constrained devices enables RESTful services down to the sensor level and the ontologies proposed by the W3C's Semantic Sensor Networks Incubator Group (SSN-XG[1]) [1]. An example of the application of all these technologies is given in [2] and it is likely that even more technologies from the Web will be adapted. This development greatly benefits both the IoT and the Semantic Web:

[☆] This research has been supported by Science Foundation Ireland under Grant No. SFI/08/CE/I1380 (Lion-II), by the Irish Research Council for Science, Engineering and Technology (IRCSET), and by the European Commission under contract number FP7-2007-2-224053 (CONET).

[*] Corresponding author.
E-mail address: danh.lephuoc@deri.org (D. Le-Phuoc).

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

Data generated by sensor devices is a growing Web data resource and users have direct access to these streams of data, and in some cases, they can also control the data sources remotely. Additionally, the use of Web technologies makes access simple and will boost the development of applications using sensor streams due to the large base of Web developers.

However, until now applications involving both (static) Web data and (dynamic) sensor data are still limited to single domains. The heterogeneous nature of streams – in terms of data, data types, and access mechanisms – makes their use and integration with other data sources a difficult and labour-intensive task, which currently requires a lot of "hand-crafting". To address these problems, there have been some recent efforts to simplify the integration of sensor data with data from other sources, by providing semantic descriptions for sensor sources and sensor data streams, e.g., by the SSN-XG. These descriptions follow the standards proposed for Linked Data [3], and the semantically enriched data being generated is known as *Linked Stream Data* [4]. Linked Stream Data is being pursued in a number of recent projects, e.g., [5–7,2,8], and it promotes an easy and seamless integration of Linked Data collections with (and among) heterogeneous sensor data, enabling a new range of "real-time" applications.

Despite its enormous potential, Linked Stream Data is still not widely explored. Although based on the Linked Data principles, its real-time nature contrasts with standard quasi-static Linked Data collections, and existing technologies for Linked Data management cannot be directly applied. Moreover, there is currently a lack of support for publishing Linked Stream Data on the Web, so that it can be available to other applications.

To overcome these challenges we have developed the *Linked Stream Middleware* (LSM), a platform that brings together sensor data (or more generally time-dependent data or data streams) from the real world and the Semantic Web. The LSM provides an extensive range of functionalities: It provides a wide range of wrappers (which can be extended by the user) to access sensor stream sources and transform the raw data into Linked Stream Data; data annotation and visualisation are possible through an intuitive Web interface; and live querying over unified Linked Stream Data and data coming from the Linked Open Data cloud is enabled by two types of query processors—a standard SPARQL query processor and CQELS (Continuous Query Evaluation over Linked Streams) [9], a native and adaptive query processor for unified query processing over Linked Stream Data and Linked Data.

An LSM deployment is available at http://lsm.deri.ie/ and currently enables access to over 100,000 sensor data sources from different domains, providing also unified tools for processing, combining and analysing data, regardless of their original source. In this paper we first describe the system architecture behind LSM and provide details of how Linked Stream Data is generated in Section 2. Then Section 3 describes a large-scale deployment and demonstrates the benefits of the platform by showcasing its interface and functionalities along with data from experimental evaluations. The section concludes with a report on the lessons learned during the implementation and deployment of LSM and the remaining open challenges. In Section 4, we overview related work and put them into context with LSM. In Section 5, we conclude with a summary of our findings.

For simplicity, we will use the term "sensor data" and "sensor streams" for any data source which dynamically produces data and enables access to it. Thus, the scope of data producers we look at ranges from single sensing devices, e.g., a temperature sensor, to complex, dynamic aggregations of sensor data.

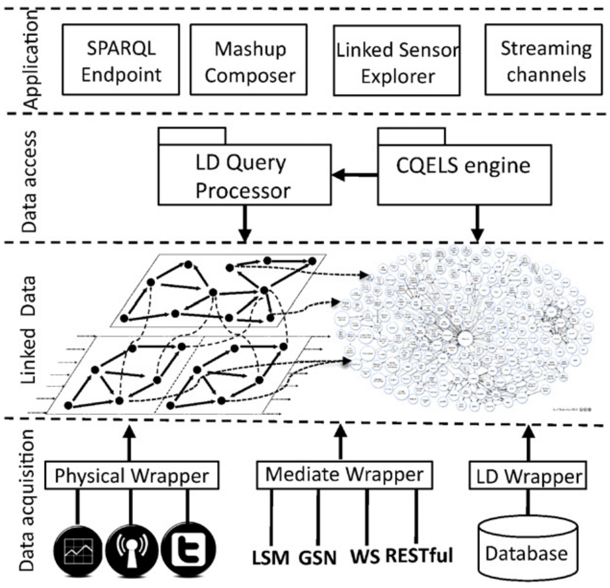


Fig. 1. The LSM layered architecture.

2. LSM architecture

The LSM is structured as a layered architecture, which increases flexibility, scalability, and facilitates maintenance. The layers together cover the entire process, from data acquisition, to Linked Data publishing and access, until applications. The LSM architecture is illustrated in Fig. 1. Next, we describe each of the individual layers in detail.

2.1. Data Acquisition layer

The Data Acquisition layer is responsible for collecting data from stream data sources. The data is acquired by the system via the wrappers. As there is a variety of interfaces for accessing sensor readings such as physical connections, middleware APIs, and database connections, the middleware provides different wrappers to cover a wide range of input formats. However, they all output the data in a unified format, following the data layout described in the Data Access Layer. Each wrapper is pluggable at runtime so that users can develop their own wrappers to connect new types of sensors into a live system.

For sensors that can be directly accessed via physical connections such as serial interfaces, IP connections, and wireless ad-hoc networks, the middleware provides Physical Wrappers. Based on the specification of each sensor platform, the respective Physical Wrapper is able to handle the communication protocol with the sensor and decode the raw data format of its output. Each sensor reading fed into the system is annotated with the meaning driven by ontologies. In the initialisation phase of the wrapper, a configuration file specifies the associated meaning of every output reading. The configuration can be provided manually or via a user-friendly wizard.

Even though Physical Wrappers provide direct access to sensor sources, in most of the deployed sensor systems there is already a middleware or a sensor data management system responsible for the sensor data acquisition. In these cases the data can be accessed via interfaces such as Web services or RESTful APIs. For instance, this is true for most of the weather services such as NOAA,[2] Weather Underground,[3] WeatherBug, and Yahoo Weather.

[2] National Oceanic and Atmospheric Administration: http://www.noaa.gov.

[3] Weather Underground: http://www.wunderground.com.

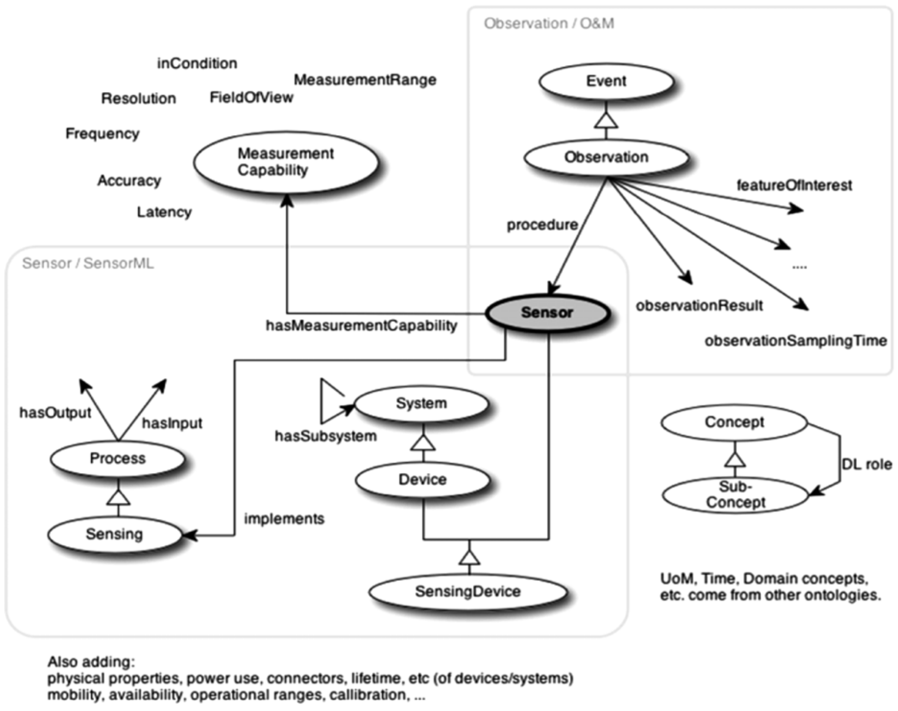


Fig. 2. Semantic Sensor Network Ontology.

In addition, some sensor middlewares like GSN [10] can publish data via Web services or push-based protocols on HTTP. To mediate the access to such systems, LSM provides Mediate Wrappers. The Mediate Wrappers use data transformation rules to map data in a given format into RDF. For example, sensor data in XML can be transformed to RDF using XSLT transformations[4] and the meanings of the sensor readings contained in the XML tags are annotated with concepts in the ontology via an XSLT transformation rule.

Sensor data can also be collected and stored in relational databases. To access these, LSM provides Linked Data Wrappers that expose data from relational databases in RDF via mapping rules. The rules act as "annotation templates" to add semantic meaning to the sensor data stored in relational tables. They can be represented using mapping languages such as D2R[5] and R2ML.[6] The Linked Data Wrappers use the mapping rules to generate the queries that are sent to the database in order to retrieve sensor data as RDF triples.

2.2. Linked Data Layer

The Data Acquisition Layer collects raw stream data and transforms it into triples. The Linked Data Layer, in turn, is responsible for making the data composable: Global identifiers are added to the data items, following the Linked Data publishing principles [11], and an ontology that captures the data model is used in the triple-based data representation. More specifically, we use the Semantic Sensor Network (SSN) Ontology.[7] Fig. 2 illustrates the modular view of the SSN Ontology. We represent the generated Linked Stream Data in a layered graph layout, using the vocabulary provided by the ontology. An example is given in Fig. 3. The layout serves as a guideline for the annotation process in the Data Acquisition Layer.

We divide the generated Linked Stream Data into two layers: sensor metadata and stream data. The sensor metadata corresponds to the time-independent information, which is static. It captures the context in which the sensor readings are obtained. In the example of Fig. 3, the sensor metadata is describing a "*weather station* that *observes* the *temperature* and *humidity* at *Dublin Airport*".

The stream data contains the dynamic, graph-based stream data from the time-varying sensor readings. These readings have links to their meanings, e.g."*tempValue* (18 *Celsius*) is the *temperature* of *Dublin Airport* at 21:32:52, 09/08/2011".

To enrich the sensor descriptions and support better integration with other data sources, the Linked Data Layer also facilitates the addition of outgoing links to data in the Linked Data Cloud. These links can be provided in the annotation module. For instance, we can add relevant links to DBpedia, GeoNames and LinkedGeoData via spatial relationships, for instance, point of interests nearby a sensor location.

2.3. Data Access layer

The Data Access layer supports declarative queries on top of the Linked Data layer. Queries over Linked Stream Data can be executed either in a pull-based or push-based fashion. For that, LSM provides two query processors as described below. Apart from the online data access, both sensor metadata and stream data can *optionally* be archived in a triple storage, facilitating access to historical information, if required. The triple storage also provides a Linked Data query processor that supports traditional pull-based queries via the SPARQL 1.1 query language. Spatial and temporal queries are also supported as an extension of SPARQL 1.1. Moreover, the local storage can be integrated with remote SPARQL endpoints via the federation extension of SPARQL 1.1. This facilitates the integration of the sensor datasets available in our middleware with other datasets in the Linked Open Data Cloud.

As we mentioned before, standard query processing techniques for Linked Data cannot be directly applied for processing Linked Stream Data, due to its dynamic nature: While traditional Linked

4 http://www.w3.org/TR/xslt.
5 http://www4.wiwiss.fu-berlin.de/bizer/d2rmap/d2rmap.htm.
6 http://www.w3.org/TR/r2rml/.
7 http://purl.oclc.org/NET/ssnx/ssn.

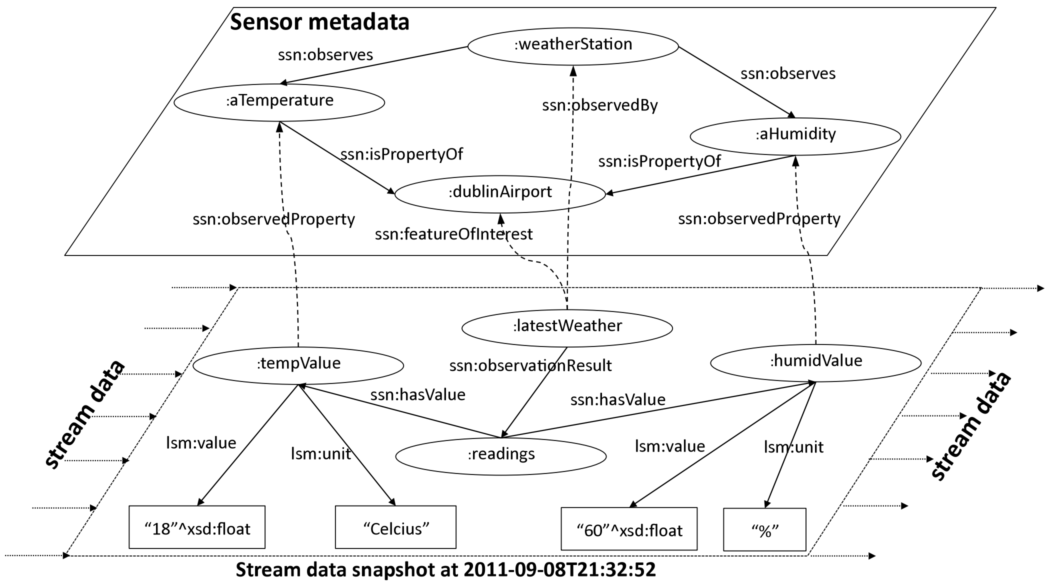


Fig. 3. Layered graph layout for Linked Stream Data.

Data queries are executed once over the entire collection and discarded after the results are produced, queries over Linked Stream Data are continuous. Continuous queries are first registered in the system, and continuously executed as new data arrives, with new results being output as soon as they are produced. A detailed discussion of the issues is provided in [9], which is omitted here due to space constraints.

For processing continuous queries over Linked Stream Data, the LSM provides the CQELS engine [9]. The query processing in CQELS is done in a push-based fashion, i.e., data entering the query engine triggers the processing. As soon as any result is found, it is sent to the query listeners, which are used to push the result output data stream of a continuous query to the delivery channels. A delivery channel can be an application, a message bus or a notification/streaming channel (see Section 2.4). The continuous queries are expressed in the CQELS language, which is an extension of SPARQL 1.1.

2.4. Application layer

Through the query processing capabilities of the Data Access layer, the Application layer provides application development support. It offers a SPARQL Endpoint, a Linked Sensor Explorer, a Mashup Composer and Notifications/Stream channels. Sensor metadata and historical sensor readings (if configured to be recorded) can be accessed/queried via the SPARQL endpoint. Continuous queries over combinations of static and dynamic data can be registered in the system, and their outputs are sent to the user-defined notification or streaming channels. Notification channels such as email, Twitter, and SMS create messages based on the new query results. Streaming channels continuously output the results stream using existing streaming protocols, such as PubSubHubbub,[8] XMPP[9] and WebSockets.[10]

PubSubHubbub stream channels deliver data to stream subscribers using the PubSubHubbub protocol, via a hub such as the Google App Hub server.[11] XMPP stream channels use the XMPP protocol for streaming data to Instant Messaging systems such as Jabber.[12] WebSocket stream channels are used for Web browsers to enable online Web applications using sensor data sources via the standardised WebSocket API and protocol.

The use of these streaming technologies provides great flexibility and ensures compatibility with existing systems. Other protocols can be added easily.

Since LSM represents all data sources in a unified triple-based model, we have extended the DERI Pipes [12] to build a Mashup Composer for LSM. The Mashup Composer enables the combination of existing data sources into new sensor data sources following a data stream paradigm. A visual editor provides an extensive range of ways to combine triple-based operators to mashup the input data sources. Different from DERI Pipes, which uses SPARQL, the Mashup Composer uses the CQELS language and query processor.

Finally, the Linked Sensor Explorer enables the exploration of existing data sources using Linked Data visualisation and data exploration techniques. A faceted browsing functionality helps users to filter sensor data based on specific properties, such as location and sensor type. More details on the Linked Sensor Explorer, as well as other functionalities of the middleware are presented in the next section, which describes our LSM deployment.

3. LSM deployment, evaluation, and lessons learned

In this section we describe the deployment architecture and some of the features available in the live LSM deployment at http://lsm.deri.ie/. We also present the results of initial performance tests and discuss the lessons learned from a large-scale, real world implementation.

3.1. Deployment architecture

In the longer term, our deployment aims at a large-scale sensor web [13]. The current deployment is the first of multiple steps towards this goal, which are necessary to learn how real users use a combination of static and dynamic data sources in real

8 http://code.google.com/p/pubsubhubbub/.
9 http://xmpp.org/.
10 http://dev.w3.org/html5/websockets/.
11 http://pubsubhubbub.appspot.com/.
12 http://www.jabber.org/.

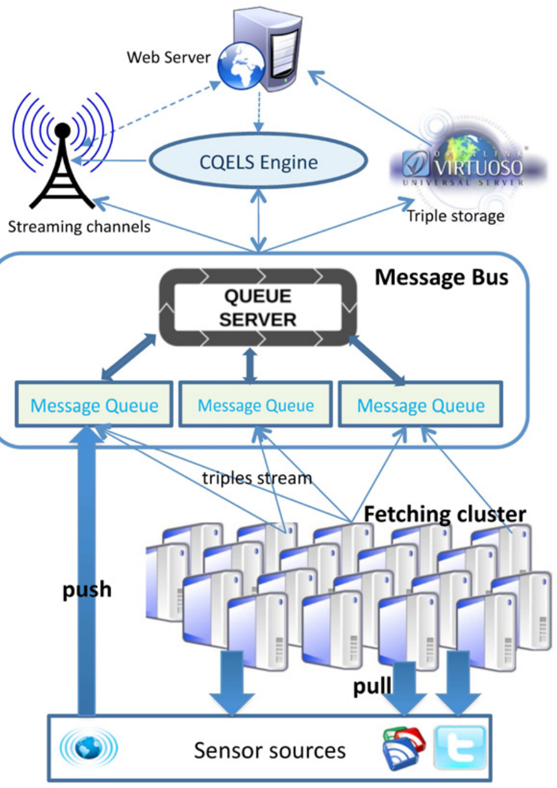


Fig. 4. Deployment architecture.

applications. This enables us to understand query and data load profiles to improve LSM. In this, we follow the accepted evidence-driven approach based on practical experiments for large-scale information systems to improve the infrastructure. Our current deployment provides access to over 110,000 sensor data streams from all continents. The architecture of the deployment is shown in Fig. 4, realising the layered architecture of Fig. 1.

The majority of the data sources in our deployment are published through HTTP as Web services or RESTful APIs, which are pull based mechanisms. For these sources, the wrappers periodically fetch each data source to collect raw sensor data and transform it into triples. The fetching operations are built as asynchronous tasks, scheduled in the fetching cluster. We use Hadoop as our fetching cluster, and we currently schedule fetching tasks for over 70,000 data sources. We use Hadoop to minimise the fetching cycles. Another set of around 20,000 sources is also pull based, but they stream media content that is not used in the query processing, e.g., traffic camera images, and because of that they are fetched on demand, outside the fetching cycles. The remaining 20,000 sources are streamed to LSM in a push-based fashion, i.e., the wrappers receive the data via a network connection and push the data into the Message Bus, as described below. In the current deployment, we did not encounter any scalability issue with these data sources.

In the Linked Data layer, LSM buffers the triples received in the Message Bus and multiplexes the data streams for further processing. After a detailed evaluation of existing message bus implementations, we chose RabbitMQ[13] for the Message Bus. RabbitMQ is a messaging publish/subscribe platform in which the triples are encoded messages exchanged between publishers and subscribers. A wrapper is an example of a publisher. The data processing operations consuming the data from the Message Bus are the subscribers. For instance, if data is to be archived, the archiving operation subscribes to the Message Bus, in order to receive sensor readings, and then inserts them into the triple storage as historical readings (this is optional and depends on the user's requirements). We use Virtuoso[14] for the triple storage. Besides the historical data, Virtuoso also stores sensor metadata and Linked Data collections. The live triple streams themselves are consumed directly from the Message Bus by the CQELS engine (without using Virtuoso). Conceptually, the Linked Data layer is comprised of the buffer windows of the CQELS engine and the triple storage.

Virtuoso also provides a SPARQL query engine with a spatial extension (Data Access Layer) and a SPARQL Endpoint (Application Layer). The Linked Sensor Explorer is deployed in the Web server. The Web socket server and the PubSubHubbub hub are also integrated into the Web server to optimally serve the respective stream channels. The RabbitMQ server natively supports XMPP, so XMPP channels are configured as subscribers to the Message Bus.

3.2. *Supported features*

LSM provides a *sensor data exploration interface* which uses a map overlay to display the sensor information as shown in Fig. 5. Several types of sensor data are available, such as flight status (1), weather (2), nearby metro stations (3), and street cameras (4). In addition to the live stream data, the historical data of a particular source (if available) can also be visualised and downloaded in RDF format. Fig. 5 shows an example of data history for a temperature sensor (2a and 2b).

Besides the map overlay, the data can also be accessed via an interactive faceted search interface, as shown in Fig. 6. The faceted search allows users to filter the information displayed based on sensor type (given by a sensor taxonomy) and/or sensor location, by choosing an area in the map, or also based on the sensor specification (physical context, accuracy, etc.).

LSM also provides a user-friendly wizard for users who have sensor sources they would like to publish as Linked Stream Data. Fig. 7(a) demonstrates the process of annotating sensor data from a source in XML format. After the user chooses the type of the sensor to be added (step 1), LSM parses the input source to extract the properties (step 2), which can then be selected and annotated (step 3). Step 4 allows users to add extra sensor descriptions. Moreover, users can also import external ontologies into the middleware, to link the available data back to concepts of their domain.

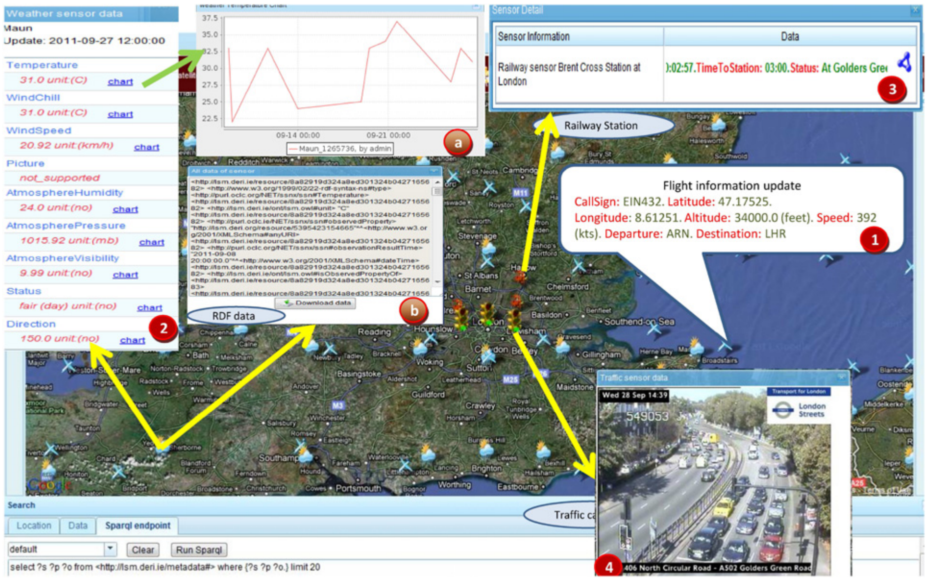
Besides SPARQL/CQELS endpoints, LSM also provides tools to create notifications or sensor feeds from live data sources. An example is given in Fig. 7(b), where a sensor feed is expressed in CQELS [9]. The feed created can be fed easily into any application. To demonstrate the simplicity of building applications with sensor feeds, we created an example applications as a blue-print for users, which overlays the up-to-date location of a mobile phone on a map, as shown in Fig. 7b. The application runs on Android and contains of only a few lines of code. The source code and application can be downloaded at http://code.google.com/p/deri-lsm/.

3.3. *Performance evaluation*

The performance of the deployment architecture from Section 3.1 depends heavily on the software and hardware settings. Hence, we have conducted tests to investigate how the system performs with different settings. We have used the following setup: The Web server, the Virtuoso storage, and the RabbitMQ

13 http://www.rabbitmq.com/.

14 http://virtuoso.openlinksw.com/.



Maun
Update: 2011-09-27 12:00:00
Temperature: 31.0 unit:(C) chart
WindChill: 31.0 unit:(C) chart
WindSpeed: 20.92 unit:(km/h) chart
Picture: not_supported
AtmosphereHumidity: 24.0 unit:(no) chart
AtmospherePressure: 1015.92 unit:(mb) chart
AtmosphereVisibility: 9.99 unit:(no) chart
Status: fair (day) unit:(no) chart
Direction: 150.0 unit:(no) chart

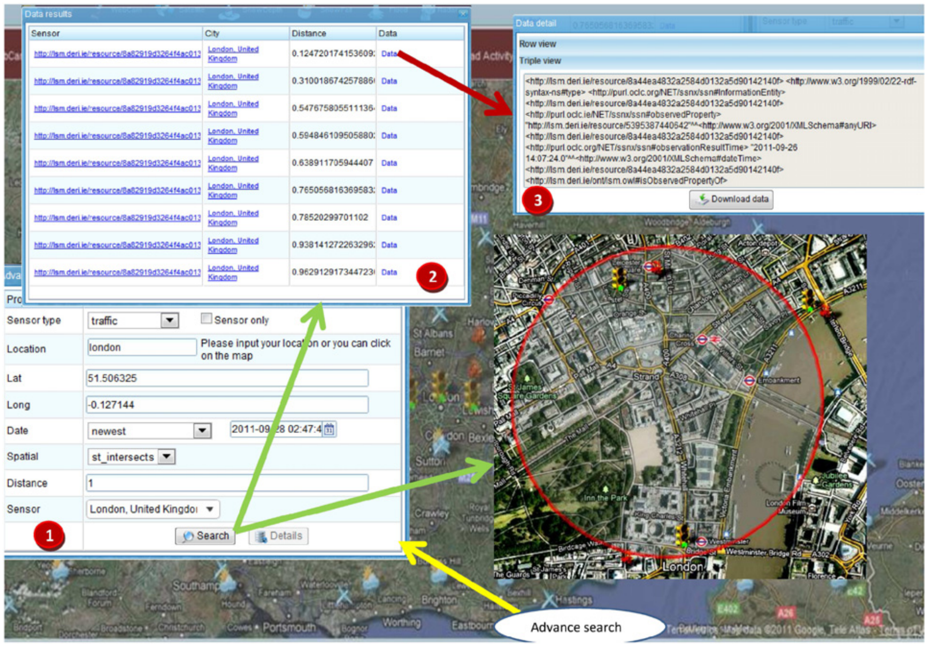
Railway sensor Brent Cross Station at London |):02:57.TimeToStation: 03:00.Status: At Golders Gree

Flight information update
CallSign: EIN432. Latitude: 47.17525.
Longitude: 8.61251. Altitude: 34000.0 (feet). Speed: 392 (kts). Departure: ARN. Destination: LHR

Traffic sensor data
Wed 28 Sep 14:39
406 North Circular Road - A502 Golders Green Road

select ?s ?p ?o from <http://lsm.deri.ie/metadata#> where {?s ?p ?o.} limit 20

Fig. 5. Sensor data exploration interface.



Sensor	City	Distance	Data
http://lsm.deri.ie/resource/8a82919d3264f4ac013	London, United Kingdom	0.124720174153609	Data
http://lsm.deri.ie/resource/8a82919d3264f4ac013	London, United Kingdom	0.310018674257886	Data
http://lsm.deri.ie/resource/8a82919d3264f4ac013	London, United Kingdom	0.547675805511136	Data
http://lsm.deri.ie/resource/8a82919d3264f4ac013	London, United Kingdom	0.594846109505880	Data
http://lsm.deri.ie/resource/8a82919d3264f4ac013	London, United Kingdom	0.638911705944407	Data
http://lsm.deri.ie/resource/8a82919d3264f4ac013	London, United Kingdom	0.765056816369583	Data
http://lsm.deri.ie/resource/8a82919d3264f4ac013	London, United Kingdom	0.78520299701102	Data
http://lsm.deri.ie/resource/8a82919d3264f4ac013	London, United Kingdom	0.938141272263296	Data
http://lsm.deri.ie/resource/8a82919d3264f4ac013	London, United Kingdom	0.962912917344723	Data

Sensor type	traffic ☐ Sensor only
Location	london — Please input your location or you can click on the map
Lat	51.506325
Long	-0.127144
Date	newest 2011-09-28 02:47:4
Spatial	st_intersects
Distance	1
Sensor	London, United Kingdo
	Search Details

```
<http://lsm.deri.ie/resource/8a44ea4832a2584d0132a5d90142140f> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://purl.oclc.org/NET/ssnx/ssn#InformationEntity>
<http://lsm.deri.ie/resource/8a44ea4832a2584d0132a5d90142140f>
<http://purl.oclc.org/NET/ssnx/ssn#observedProperty>
"http://lsm.deri.ie/resource/5395378440642"^^<http://www.w3.org/2001/XMLSchema#anyURI>
<http://lsm.deri.ie/resource/8a44ea4832a2584d0132a5d90142140f>
<http://purl.oclc.org/NET/ssnx/ssn#observationResultTime> "2011-09-26
14:07:24.0"^^<http://www.w3.org/2001/XMLSchema#dateTime>
<http://lsm.deri.ie/resource/8a44ea4832a2584d0132a5d90142140f>
<http://lsm.deri.ie/ont/lsm.owl#isObservedPropertyOf>
```

Fig. 6. Faceted search.

run in dedicated computers, each with the following configuration: 1 × Quad Core Intel Xeon E5410 2.33 GHz, 8 GB memory, 2 × 500 GB Enterprise SATA disks, Ubuntu 11.04/x86_64, Java version 1.6, Java HotSpot (TM) 64-Bit Server VM. The same configuration is also used for the processing nodes in the Hadoop cluster. The bandwidths are 1 Gb/s for the local networks, and 84 Mb/s for the Internet connection.

3.3.1. Data consumption

We have carried out tests on the data consumption for the 70,000 sensor data sources available as Web services or RESTful APIs as this is the major load contributor due to scheduling of fetching tasks. The data consumption process consists of fetching the sensor readings and then buffering the readings in the Message Bus. The fetching operations are scheduled as MapReduce tasks that run on the Hadoop cluster. We define *Feed time* as the time spent to fetch data from all sources once (fetching cycle). A fetching cycle is scheduled as concurrent processing tasks (one task for each data source) in the cluster's processing queue. The working threads of the cluster continuously send the outputs to the Message Bus as soon as they finish each fetching task. The Feed time depends mainly on two factors: The number of cluster nodes, and the number of concurrent MapReduce tasks running on each node. We have varied the number of nodes from 3 to 32 and measured the Feed time with 10 and 15 concurrent MapReduce tasks per node. We have also computed the *Insert time*, i.e. the time it takes to insert the output of a fetching cycle into the triple storage. Results are shown in Figs. 8 and 9.

We can see that in both charts the feed and insert times drop sharply when the number of nodes increases from 3 to 8, while the drop is smoother from 8 to 32 nodes. This is caused by the bandwidth limitation and the capacity of the servers hosting the sensor data sources. We can also observe that there is no significant difference between the 10 and 15 tasks configurations. To better

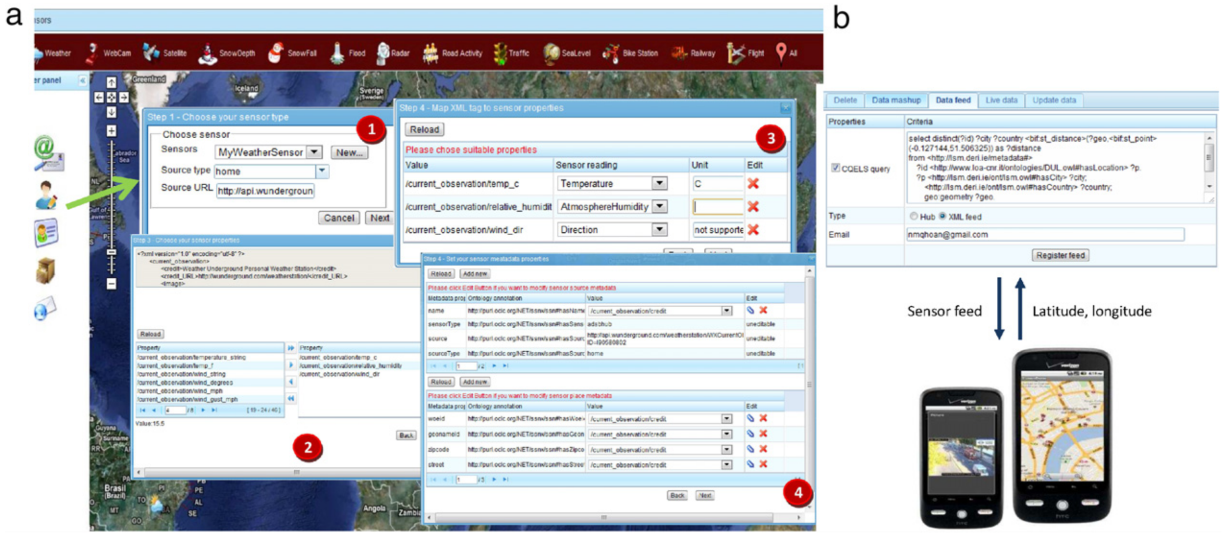


Fig. 7. Sensor data annotation (a), and an example of a sensor data feed for a mobile device (b).

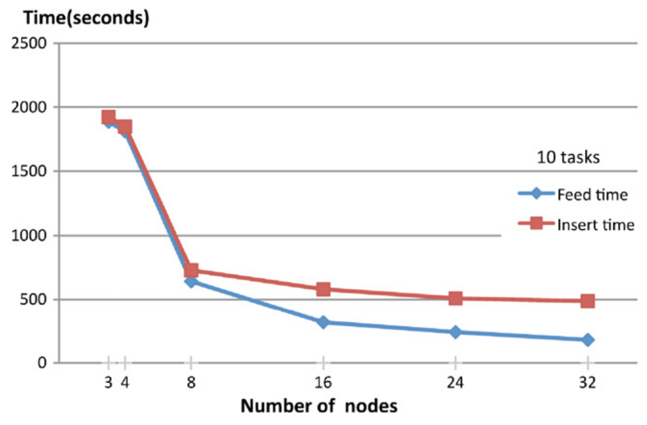


Fig. 8. Feed and insert times with 10 tasks per cluster node.

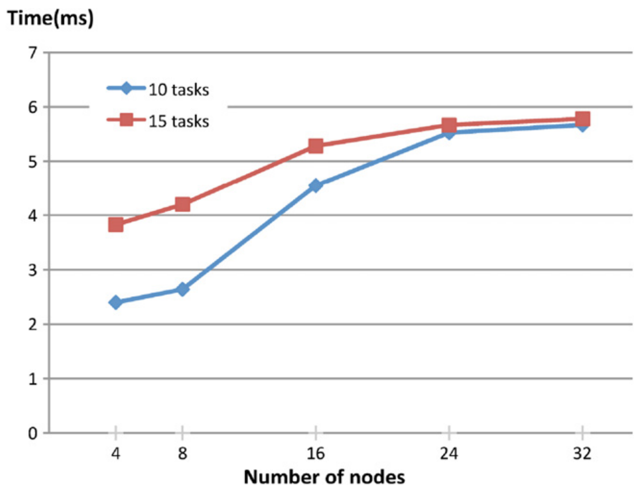


Fig. 10. Fetching delay for 10 and 15 tasks per cluster node.

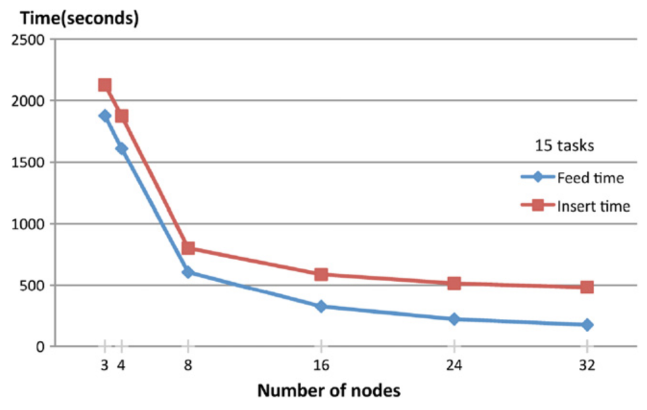


Fig. 9. Feed and insert times with 15 tasks per cluster node.

understand these phenomena, we have measured the delay added by the cluster in a fetching task. The results in Fig. 10 reveal that increasing the number of tasks can introduce more delay for each processing task. This extra delay is produced by the overhead of scheduling and hosting the asynchronous threads for additional tasks. However, the chart also shows that adding more processing nodes to the cluster reduces the delay difference between both cases. These results may sound simple but nevertheless determine the practical scalability of a deployment.

Another important observation in Figs. 8 and 9 is that the system takes less time to finish a fetching cycle than to flush the output into the triple storage. To find out the difference between the throughputs of the fetching and inserting operations, we have measured them separately. The results are given in Fig. 11. We can see that feed rates are much higher than insert rates, especially when the number of nodes increases beyond 16. This shows that the triple storage acts as a bottleneck. This is a common issue of triple storages, since they are not designed for write-intensive applications. An alternative would be to use the stream processing engine to filter and aggregate high-update-rate data on the fly before storing them in the triple storage, therefore reducing the number of insertions, which we will investigate further in coming versions of LSM.

3.3.2. Query execution

We have also evaluated LSM in terms of query execution time. For continuous queries, the current version of the CQELS engine can handle stream throughputs from 5000 up to 20,000 triples per second. Hence, CQELS can cope with the feed rates shown in Fig. 11. The delay for delivering results is about 0.5–200 ms depending on

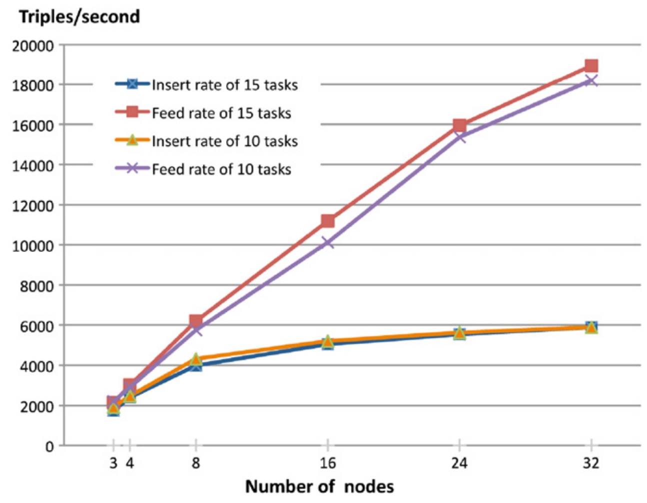


Fig. 11. Feed and insert rates for 10 and 15 tasks per cluster node.

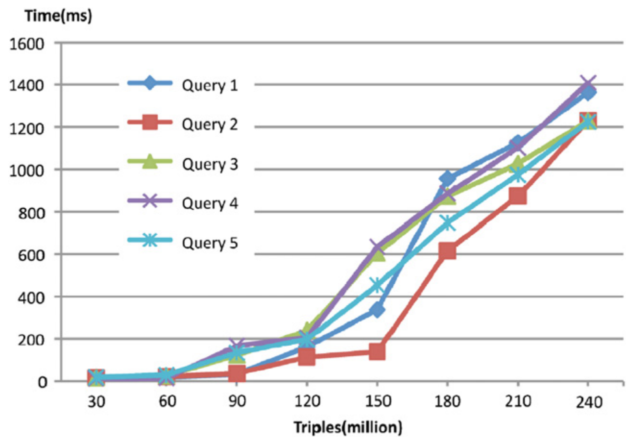


Fig. 12. Query execution time with increasing number of triples.

the query complexity and the sizes of static data involved. CQELS can also handle thousands of concurrent queries as shown in the evaluation of CQELS in [9].

While CQELS can cope with highly dynamic data streams, when combining stream and static data (e.g., historical and metadata), CQELS's performance is naturally limited by the query response time of the triple storage that hosts the static input. The triple storage also dictates the amount of historical readings that can be handled. This is outside the possible optimisations of CQELS as these are outside linked data sources.

To evaluate the query processing performance over historical data in the LSM, we incrementally fetch weather readings from 70,000 sources and insert them into the triple storage. We repeat this process every 30 min, and measure the query execution time with the increased number of triples. While this may look like an artificially limited scenario, it nevertheless is the most common practical deployment scenario in distributed sensor web systems. We have chosen the following five queries, since they represent the most popular queries over historical readings and metadata:

- *Query* 1: Get the latest reading of a sensor.
- *Query* 2: Retrieve all the readings within a time interval.
- *Query* 3: Find all the sensors within an area that have a reading in a specific value range.
- *Query* 4: Get the latest reading of the nearest sensor that can measure a certain type of reading, e.g., temperature.
- *Query* 5: Show a reading, e.g., temperature, of a location, e.g., "Times Square" (the name is provided by geodata from DBpedia and LinkedGeoData).

The SPARQL representation of the queries is given in the Appendix. As usual in database benchmarks, we first warm up Virtuoso's cache by running each query 20 times. After that, we executed each query 100 times and averaged the execution times. At each execution the constant values in the queries were chosen randomly. We evaluated the queries every time 30 million new triples had been inserted in the storage (approximately after eight fetching cycles). Fig. 12 shows the results over a period of 30 h.

We can observe that the query execution time of all queries increases sharply after 150 million triples, which corresponds to approximately 37 readings per sensor source. At this point the triple storage performs poorly and crashes often. This creates a limitation on the number of historical readings that can be stored. In this setup, we would collect over 190 million triples in just one day, which would result in poor query response times. This limitation can be slightly improved adding more computing resources. For instance, if doubling the memory from 8 to 16GB, this setup can handle up to 500 million triples.[15] However, this solution is not applicable when the number of triples goes beyond billions, for example, historical readings for months. Thus the amount of collecting historical data must be monitored by the applications. The most common strategies would be to collect no historical data which is possible in many scenarios, regularly expire such data, or dynamically off-load it into a different storage not used in the online processing.

3.4. *Discussion and lessons learned*

Scalability is the crucial feature for the targeted deployment scenarios of LSM. Millions of sensor data sources are published via HTTP and must be supported by any infrastructure similar to LSM. For handling this huge amount of information, the data fetching process needs to run in a reliable, scalable and bottleneck-free infrastructure. In the presented deployment architecture of LSM, Hadoop allows us to increase the data consumption and transformation rate by simply adding more hardware, which is a proven strategy in most large-scale services. However, adapting the task scheduling to the variations on the stream rates and server throughputs is still an open challenge. On top that, more experiments to show how the performance of the consumption changes in a certain setup when adding new data sources are needed.

Besides the online data processing, it is also necessary to store the data, either for queries defined over a time period or for archiving purposes. We have observed that most of the triple storages cannot efficiently handle high insertion rates. The performance evaluation has shown that for just one day worth of data, at a very slow sampling rate of one reading every 30 min for each source, the size of the historical data could easily be in the range of billions of triples given the current growth rate in the number of sources. Such live systems are expected to gather data at a much faster pace, therefore more efficient data storages are clearly needed. This is an open challenge for research on triple storages. We are investigating how to apply the solutions from Streaming Data Warehouses [14] to triple storages. For instance, to faster insert and query over continuously update and massive datasets, the datasets can be split into smaller partitions so that the updates/expirations only affect smaller chunks of data. Therefore, the burden of updating the indexes is lifted to enable a much higher throughput of insert and a faster query speed.

15 How many triples can a single server handle? http://virtuoso.openlinksw.com/dataspace/dav/wiki/Main/VOSVirtuoso6FAQ.

As an alternative to triple storages, we have tried to use relational tables to store the historical data. However, this requires mapping the data from relations to triples. Virtuoso provides a mapping language as well as a query rewriter that enables querying relational data from SPARQL in a transparent way, but we have encountered performance issues with more advanced queries, because the rewritten queries were too complex. Hence, better mapping languages and optimised query rewriters are needed. Looking closely at the queries in the Appendix, the complexity of the queries on historical data comes from classes and properties that capture sensor readings. This raises the challenge of designing sensor ontologies that could improve the Linked Stream Data processing. A further line of investigation is the use of column storages.

4. Related work

Traditionally, sensor data fusion applications require prior knowledge about the sensor systems as well as knowledge about the environment for which the system was built, which limits considerably the development of new applications and hinders the data integration process. Preliminary efforts to support flexible reuse of sensor data were the definition of standards such as the standards for transducers (IEEE 1451), the Radiation Detection Standards (ANSI N42), the Open Geospatial Consortium (OGC) Sensor Web Enablement [15], and the Extended Environments Markup Language (EEML) [16]. However, most of these standards are oversimplified, and either too general or too domain specific, so only a few systems currently conform to them. The Sensor Web Enablement has achieved a bigger impact and is used in projects such as the 52° North [17] project, the NASA/JPL Sensor Webs project [18], and the European Space Agency [19].

Advancements in sensor and network technologies enables sensor systems to share their data. Most of the existing approaches for sensor data sharing provide simple interfaces for publishing and retrieving data via centralised portals. SensorBase[16] enables the publication of sensor data via HTTP POST and provides Web services to query sensor data from its relational database. In a similar fashion, Pachube[17] provides a RESTful interface to stream live sensor data. Historical data and live data can be retrieved via feeds in XML (EEML) or JSON formats. Along with Web services to publish and request historical and online sensor data, SensorMap [20] also provides an explorer for sensor data on maps. Another approach that uses HTTP for making sensor data accessible is [21]. It combines the advantages of the Web feed and multimedia streaming paradigms. Each sensor stream has a URL with associated parameters to query and filter historical and live data. The Global Sensor Networks (GSN) [10] provides a zero-programming, declarative middleware to publish sensor data directly from physical sensors connected to a PC via interfaces like USB, Bluetooth, UDP and TCP/IP, or from virtual sensors. Sensor data is accessible through HTTP with querying parameters. GSN supports a wide range of sensors and offers a nice development environment with Web interfaces and visualisation functionalities.

This increasing number of available sensors also led to a number of projects to generate and explore sensor metadata information. Sensor metadata is crucial for sensor source selection and for the integration with other data sources. The OGC proposed sensor and observation models based on XSD schemas for specifying interoperability interfaces and metadata encodings, in order to enable the online integration of heterogeneous sensor Webs. However, there is a gap between the syntactic XSD of OCG's Sensor Web Enablement (SWE) and the RDF/OWL-based metadata, which is commonly used for representing domain knowledge. To bridge this gap, Sheth et al. [22] proposed to use RDFa to annotate ontological concepts and properties of SWE by using XLink [23]. This approach requires the underlying sensor system to capture semantic data along with the raw sensor readings, and to provide sensor stream data management and support for some semantic functionalities. While the concepts have been proposed, so far no such system has been implemented.

Other approaches use semantic descriptions of sensor data streams to automatically compose sensor applications and services. Semantic Streams [7] allows users to specify queries based on the semantics of the sensor data streams. The semantics are described using Prolog-based logic rules. Due to issues of scalability and decidability of Semantic Streams, Bouillet et al. [5] propose to use OWL to represent the sensor data stream and the processing elements for composing applications. However, these approaches also assume that semantic descriptions of sensor data are already available.

SenseWeb [24] is a platform for leveraging data from available sensor data streams across the Web. It relies on sensor metadata to help in the selecting process of appropriate sensor sources for each application. As an early version of LSM, SensorMasher [25] was the first platform for publishing sensor data as Linked Data. Its mashup engine enables the user to visually explore and combine sensor data resources in a homogeneous triple-based data model. SemSorGrid4Env [26] also explores semantic annotations to support well-informed interactions among heterogeneous sensor data. Similar to LSM, it uses another Linked Data stream processing engine, called SPARQL$_{stream}$ [27]. However, it mainly focuses on data integration, whilst the data acquisition, performance and scalability are not mentioned in the literature.

5. Conclusions and further work

Data streams add completely new functionalities to Web applications by providing up-to-date information and a link to the physical world if a stream comprises sensor information. Possible applications range from simple information services up to sophisticated real-time information and planning systems as in the case of smart cities, which enable efficient decision-making of people, public administrations, and businesses. The technical challenges are not trivial but solvable as demonstrated by the LSM platform described in this paper. We have already a range of approaches to address the issues of data and platform heterogeneity and integrated stream processing according to the Linked Data paradigm. If supported well, Linked Stream Data will become as significant as Linked Data and will allow Web technologies also to penetrate other areas such as the Internet of Things and mobile applications. Some of the open challenges are shared with other domains, for example, designing triple storages optimised for writing-intensive operations. Others, specifically on the security and privacy side are open problems and require in-depth investigation. Global linking of information together with real-world information bears significant risks along with the huge innovation potential that exists. However, this is an issue shared with any information dissemination and integration technology and will have to be addressed comprehensively. On the Web side, the integration of sensor information and social networks also seems an interesting new research direction: Sensors can provide valuable information for social networks, for example, physical presence integrated with virtual presence in the social network, while social networks and people can be used as sensors (opportunistic sensing, citizen sensors).

16 http://sensorbase.org/.
17 http://pachube.com/.

Appendix. Evaluation queries over historical data

Query 1

```
select ?s ?p ?o
where{
 {
  select ?x where
    {?x <http://purl.oclc.org/NET/ssnx/ssn#observedBy>
     <http://lsm.deri.ie/resource/8a8291b73215690e01321576ff047d51>.
      ?x <http://purl.oclc.org/NET/ssnx/ssn#observationResultTime> ?time.}
     order by desc(?time) limit 1  }
 ?s <http://lsm.deri.ie/ont/lsm.owl#isObservedPropertyOf> ?x. ?s ?p ?o.}
}
```

Query 2

```
select ?s ?p ?o
where{
  {
   select ?observation where
     { ?observation <http://purl.oclc.org/NET/ssnx/ssn#observedBy>
       <http://lsm.deri.ie/resource/8a8291b73215690e01321576ff047d51>.
       ?observation <http://purl.oclc.org/NET/ssnx/ssn#observationResultTime> ?time.
       filter( ?time > "2012-01-11"^^xsd:date &&?time < "2012-01-13"^^xsd:date). }
  }
  ?s <http://lsm.deri.ie/ont/lsm.owl#isObservedPropertyOf> ?observation.  ?s ?p ?o.
}
```

Query 3

```
select ?s ?value
where{
  {
    select ?sensorId  from <http://lsm.deri.ie/metadata#>
    where{
        ?sensorId  <http://www.loa-cnr.it/ontologies/DUL.owl#hasLocation> ?p.
        ?p geo:geometry ?geo.
    filter (<bif:st_intersects>(?geo,<bif:st_point>(-2.900....,47.5499...),5)).
    }order by <bif:st_distance>(?geo,<bif:st_point>(-0.127144,51.506325))
  }
  ?obs <http://purl.oclc.org/NET/ssnx/ssn#observedBy>  ?sensorId.
  ?s <http://lsm.deri.ie/ont/lsm.owl#isObservedPropertyOf> ?obs.
  ?s rdf:type <http://lsm.deri.ie/ont/lsm.owl#WindSpeed>.
  ?s <http://lsm.deri.ie/ont/lsm.owl#value> ?value.
  filter( ?value > 15 && ?value < 20).
}
```

Query 4

```
  select ?temp ?value ?times
  where{
  {
    select ?sensorId  from <http://lsm.deri.ie/metadata#>
    where{
        ?sensorId  <http://www.loa-cnr.it/ontologies/DUL.owl#hasLocation> ?p.
        ?p geo:geometry ?geo.
    filter (<bif:st_intersects>(?geo,<bif:st_point>(-2.900...,47.5499...),5)).
    }order by <bif:st_distance>(?geo,<bif:st_point>(-0.127144,51.506325)) limit 1
  }
  ?obs <http://purl.oclc.org/NET/ssnx/ssn#observedBy> ?sensorId.
  ?temp <http://lsm.deri.ie/ont/lsm.owl#isObservedPropertyOf> ?obs.
  ?temp rdf:type <http://lsm.deri.ie/ont/lsm.owl#AirTemperature>.
  ?temp <http://lsm.deri.ie/ont/lsm.owl#value> ?value.
  ?temp <http://purl.oclc.org/NET/ssnx/ssn#observationResultTime> ?time
}order by desc(?time) limit 1
```

Query 5

```
select ?temp
where{
{
    select ?sensorId  from <http://lsm.deri.ie/metadata#>
    where{
        ?sensorId  <http://www.loa-cnr.it/ontologies/DUL.owl#hasLocation> ?p.
        ?p geo:geometry ?geo. ?p <http://www.geonames.org/ontology/#nearby> ?poi.
        ?poi rdf:label  'Time Square'. }
     ?obs <http://purl.oclc.org/NET/ssnx/ssn#observedBy> ?sensorId.
     ?temp <http://lsm.deri.ie/ont/lsm.owl#isObservedPropertyOf> ?obs.
     ?temp rdf:type <http://lsm.deri.ie/ont/lsm.owl#AirTemperature>.
     ?temp <http://lsm.deri.ie/ont/lsm.owl#value> ?value.
}
```

References

[1] M. Compton, P. Barnaghi, L. Bermudez, R.G. Castro, O. Corcho, S. Cox, J. Graybeal, M. Hauswirth, C. Henson, A. Herzog, V. Huang, K. Janowicz, W.D. Kelsey, D. Le-Phuoc, L. Lefort, M. Leggieri, H. Neuhaus, A. Nikolov, K. Page, A. Passant, A. Sheth, K. Taylor, The SSN Ontology of the Semantic Sensor Networks Incubator Group, Web Semantics: Science, Services and Agents on the World Wide Web (2012).

[2] D. Pfisterer, K. Römer, D. Bimschas, O. Kleine, R. Mietz, C. Truong, H. Hasemann, A. Kröller, M. Pagel, M. Hauswirth, M. Karnstedt, M. Leggieri, A. Passant, R. Richardson, Spitfire: toward a semantic web of things, IEEE Communications Magazine 49 (11) (2011) 40–48.

[3] C. Bizer, T. Heath, T. Berners-Lee, Linked data—the story so far, International Journal on Semantic Web and Information Systems 5 (3) (2009) 1–22.

[4] J.F. Sequeda, O. Corcho, Linked stream data: a position paper, in: SSN Workshop, 2009.

[5] E. Bouillet, M. Feblowitz, Z. Liu, A. Ranganathan, A. Riabov, F. Ye, A semantics-based middleware for utilizing heterogeneous sensor networks, in: DCOSS, 2007, pp. 174–188.

[6] A.P. Sheth, C.A. Henson, S.S. Sahoo, Semantic sensor web, IEEE Internet Computing 12 (4) (2008) 78–83.

[7] K. Whitehouse, F. Zhao, J. Liu, Semantic streams: a framework for composable semantic interpretation of sensor data, in: EWSN, 2006, pp. 5–20.

[8] M. Leggieri, A. Passant, M. Hauswirth, inContext-Sensing: LOD augmented sensor data, in: ISWC (Posters and Demos), 2011.

[9] D. Le-Phuoc, M. Dao-Tran, J.X. Parreira, M. Hauswirth, A native and adaptive approach for unified processing of linked streams and linked data, in: ISWC, 2011, pp. 370–388.

[10] K. Aberer, M. Hauswirth, A. Salehi, Infrastructure for data processing in large-scale interconnected sensor networks, in: MDM, 2007, pp. 198–205.

[11] T. Heath, C. Bizer, Linked Data: Evolving the Web into a Global Data Space, first ed., Morgan & Claypool, 2011.

[12] D. Le-Phuoc, A. Polleres, M. Hauswirth, G. Tummarello, C. Morbidoni, Rapid prototyping of semantic mash-ups through semantic web pipes, in: WWW, 2009, pp. 581–590.

[13] M. Balazinska, A. Deshpande, M. Franklin, P. Gibbons, J. Gray, S. Nath, M. Hansen, M. Liebhold, A. Szalay, V. Tao, Data management in the worldwide sensor web, IEEE Pervasive Computing 6 (2) (2007) 30–40.

[14] L. Golab, M.T. Özsu, Data stream management, Synthesis Lectures on Data Managemen, 2010, pp. 1–73.

[15] Sensor web enablement dwg. http://www.opengeospatial.org/projects/groups/sensorweb.

[16] Extended environments markup language, http://www.eeml.org.

[17] 52° north, http://52north.org.

[18] Nasa/jpl sensor webs project, http://sensorwebs.jpl.nasa.gov.

[19] European space agency, http://www.esa.int/esaCP/index.html.

[20] S. Nath, J. Liu, F. Zhao, Sensormap for wide-area sensor webs, IEEE Computer 40 (7) (2008) 90–93.

[21] R.F. Dickerson, J. Lu, J. Lu, K. Whitehouse, Stream feeds—an abstraction for the world wide sensor web, in: IOT, 2008, pp. 360–375.

[22] A. Sheth, C. Henson, S.S. Sahoo, Semantic sensor web, IEEE Internet Computing 12 (2008) 78–83.

[23] Xml linking language. http://www.w3.org/TR/xlink/.

[24] A. Kansal, S. Nath, J. Liu, F. Zhao, Senseweb: an infrastructure for shared sensing, IEEE MultiMedia 14 (2007) 8–13.

[25] D. Le Phuoc, Sensormasher: publishing and building mashup of sensor data, in: Triplification Challenge, I-Semantics'09, 2009.

[26] A.J.G. Gray, R. García-Castro, K. Kyzirakos, M. Karpathiotakis, J.-P. Calbimonte, K. Page, J. Sadler, A. Frazer, I. Galpin, A.A.A. Fernandes, N.W. Paton, O. Corcho, M. Koubarakis, D. De Roure, K. Martinez, A. Gómez-Pérez, A semantically enabled service architecture for mashups over streaming and stored data, in: ESWC, 2011, pp. 300–314.

[27] J.P. Calbimonte, O. Corcho, A.J.G. Gray, Enabling ontology-based access to streaming data sources, in: ISWC'10, 2010, pp. 96–111.