XGSN: An Open-source Semantic Sensing Middleware for the Web of Things

Jean-Paul Calbimonte, Sofiane Sarni, Julien Eberle and Karl Aberer

Faculty of Computer Science and Communication Systems, EPFL, Switzerland.
firstname.lastname@epfl.ch

Abstract. We present XGSN, an open-source system that relies on semantic representations of sensor metadata and observations, to guide the process of annotating and publishing sensor data on the Web. XGSN is able to handle the data acquisition process of a wide number of devices and protocols, and is designed as a highly extensible platform, leveraging on the existing capabilities of the Global Sensor Networks (GSN) middleware. Going beyond traditional sensor management systems, XGSN is capable of enriching *virtual sensor* descriptions with semantically annotated content using standard vocabularies. In the proposed approach, sensor data and observations are annotated using an ontology network based on the SSN ontology, providing a standardized queryable representation that makes it easier to share, discover, integrate and interpret the data. XGSN manages the annotation process for the incoming sensor observations, producing RDF streams that are sent to the cloud-enabled Linked Sensor Middleware, which can internally store the data or perform continuous query processing. The distributed nature of XGSN allows deploying different remote instances that can interchange observation data, so that virtual sensors can be aggregated and consume data from other remote virtual sensors. In this paper we show how this approach has been implemented in XGSN, and incorporated to the wider OpenIoT platform, providing a highly flexible and scalable system for managing the life-cycle of sensor data, from acquisition to publishing, in the context of the semantic Web of Things.

1 Introduction

From wearable devices for health monitoring to geospatial and environmental sensors, we are surrounded by objects or things which are susceptible to be present in the Web, in one way or another. Sensed data on the web is a need and a reality in many real-life use cases and scenarios nowadays. The gap between the real and virtual world is narrowing and there is an increasing necessity to identify everyday life entities in the Web, and let them interact among them, as well as with real people. Many of these challenges have converged towards concepts such as the Internet of Things and the Web of Things, which have gathered enormous attention from academia and the industry [3].

However, when comes the time to expose these data in the Web, there are several problems that data providers may encounter on the way. One is the heterogeneity of the data sources. Starting from the devices themselves, there is an enormous range of gadgets and equipment with different capabilities, accuracy, range or frequency. There also exist numerous possible IoT protocols and technologies that devices can use to publish data to the Web (CoAP, XMPP, MQTT, DDS, etc.) each targeting different use cases. Many of these challenges have been addressed in previous years from different perspectives [4]. Through a middleware system, applications and users may access data from interconnected objects and things, hiding the internal communication and low-level acquisition aspects. As an example, the GSN middleware has already provided an extensible protocol-agnostic mechanism to acquire data from sensing devices, using configurable wrappers [1], and implementing some of these protocols.

However, these technical difficulties at the lower layers are only the tip of the iceberg, considering that even if they are addressed by platforms such as GSN, there is still a large heterogeneity problem when the data that is sensed needs to be interpreted and understood. For example, the number of possible *observed properties* that may be sensed by an entity, such as humidity, radiation, soil moisture, location detection, etc. can include almost any type of phenomenon or event in the surrounding world. Even if these elements can be abstracted in a domain-model, there are also different ways of exposing and publishing the data in the web, through different formats, under different data models and using different service abstractions. In the end, in many cases the result is a use-case-tailored system that gathers data from a particular set of sources, and exposes them using some ad-hoc data model, creating yet-another isolated silo of data in the web, with very few possibilities of re-use or integration.

One of the ways to tackle these heterogeneity issues is by following a semantics-based approach. Using semantically rich models (ontologies which can be extended for a particular use case), a number of systems [21, 19] have shown how very uneven data sources can be shared and be mutually understandable, while following emerging standards and principles such as Linked Data [7]. In the more specific case of sensor data, specific ontologies and vocabularies such as the SSN (Semantic Sensor Network) Ontology [10] have been created by the community, and have been adopted in a number of projects already [6, 14, 9, 16, 20]. Existing standards for publishing and accessing semantically annotated data (SPARQL[1], Linked Data Platform[2], etc.) are gaining adoption and establishing best practices for sharing data.

In this paper we describe XGSN, a middleware solution that handles the lifecycle of *virtual sensors* (devices, objects or people observing properties around them), providing semantic annotations for them and the observation data that they produce. The key idea is to provide an end-to-end semantic-enabled platform for IoT data management, in which XGSN plays the role of a fully distributed data acquisition middleware with semantic annotation capabilities. We describe the architecture of this system and its implementation, emphasizing on the distributed data processing that allows XGSN to produce different layers of aggregated observations. XGSN extends the successful Global Sensor Net-

[1] W3C Recommendation SPARQL 1.1 `http://www.w3.org/TR/sparql11-query/`
[2] W3C Candidate Recommendation LDP: `http://www.w3.org/TR/ldp/`

works [1] system with the semantics-aware capabilities described in this paper, and is available as an open-source package that can be used and extended. The existing community of developers and users inherited from GSN positions this software project as one of the most comprehensive and extensible tools for IoT data management, as it has been shown in several real-life deployments and environmental scientific research. XGSN is a ready-to-use system[3], also available as part of the OpenIoT platform[4], and has also been integrated with the Linked Sensor Middleware (LSM) [16], showing that it can be plugged to an RDF-enabled data store. The remainder of the paper is structured as follows: In Section 2 we describe the general approach of XGSN. Then we present the ontology management aspects and annotation process in Section 3. The architecture is described in Section 4 and the distributed virtual sensor management and experimentation in Section 5. We discuss the related work in Section 6 before concluding.

2 The XGSN Approach for Semantic Data Management

XGSN is built as an extended fork of the GSN middleware [1], which already implemented pluggable sensor data acquisition mechanisms, combined with a distributed stream processing layer. GSN is an inherently decentralized system where different instances can exist in a distributed deployment (see Figure 1), and interchange observation data as needed. The distribution can be based on geographical, economic, privacy or scalability constraints, and each instance can expose a number of different *virtual sensors*. These virtual sensors can be logical abstractions of one or more real sensors or objects or any entity that captures data. They can also be aggregators or filters applied to other virtual sensors, which can be deployed locally or remotely. The interface between devices, sensors or inter-connected *things* and a virtual sensor is a *wrapper*, of which different implementations can co-exist. Different wrappers are already available in the system (e.g. UDP, serial, HTTP, etc.) and creating a new one is generally a simple extension task. Once the data is captured by the wrapper, GSN also provides an extendable processing layer which can be programmed to store the observation data, annotate it, apply correction algorithms over it, etc.

Although GSN already dealt with the problem of handling heterogeneity at the device and acquisition level, it was not able to provide higher level abstractions over the virtual sensors, so that applications could interpret and reuse the data without an external entity deciphering it. In XGSN we follow a semantics-based approach, annotating the virtual sensors with relevant metadata using an extension of the SSN ontology. Two main types of semantic annotations have been added in XGSN. The first are metadata annotations, related to sensors, sensing devices and their capabilities[5], which could not be described before in GSN. These are typically linked to the virtual sensors declared in an XGSN

[3] GSN: `http://gsn.epfl.ch`

[4] OpenIoT: `http:/openiot.eu`

[5] Related to the Measuring and Measuring Capability modules in the SSN ontology

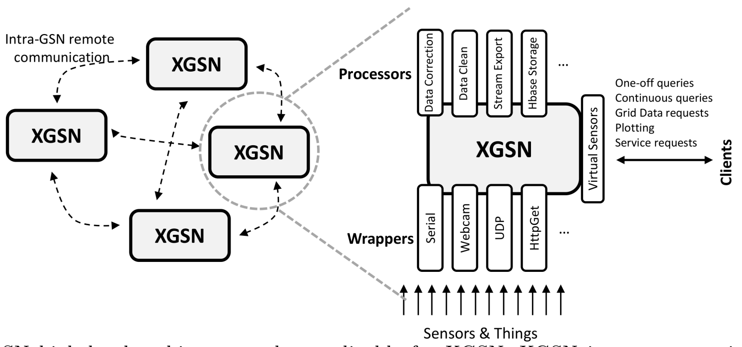


Fig. 1: GSN high level architecture, also applicable for XGSN. XGSN instances may interchange observation data remotely. Each one acquires data through a set of wrappers, and offer continuous data handling capabilities through extensible processors.

instance: e.g. describe the sensing device that produces the data in a particular virtual sensor, its location, the type of observation it produces, the responsible person or organization, the source type, etc. The other type of annotations are related to the observations or measurements produced continuously by the sensors. This includes the semantic information that describes the time and context when the observation happened, the observed property, unit, the values themselves, etc. We will see in Section 3 how these metadata and observation annotations can be exposed as Linked Data using an RDF-enabled cloud system such as LSM.

These explicit semantics in the virtual sensor representation facilitate the tasks of discovery and search in an IoT environment. Also for actual observations, XGSN can provide different levels of semantic annotations to them, depending on the type of virtual sensor that is exposed. For example, in an air quality scenario, XGSN can annotate each measurement made by a sensor (including value, unit, data type, etc.) as an observation of a property (e.g. NO_2), as depicted in Figure 2. However, for some use cases low level annotations are not useful or relevant, so XGSN can aggregate, filter or process several observations over time or space. This will produce indicators in a higher level virtual sensor, each of which can be annotated with even higher-level concepts of a domain ontology (e.g. a “low air quality” observation). Furthermore, even more complex correlations, and processing including external data sources or data from other XGSN instances, can lead to annotations that denote actionable and human-comprehensible concepts like alerts or activities.

In order to make this possible, XGSN relies on ontologies for sensor and observation representation, with three main extensibility points: at the model, data acquisition and processing levels, as we will detail next.

3 Ontologies and Annotation in XGSN

The basis of the abstract model used by XGSN for sensing entities and observations in the web of things, is the SSN ontology[6]. This ontology is not limited

[6] SSN Ontology: `http://purl.oclc.org/NET/ssnx/ssn`

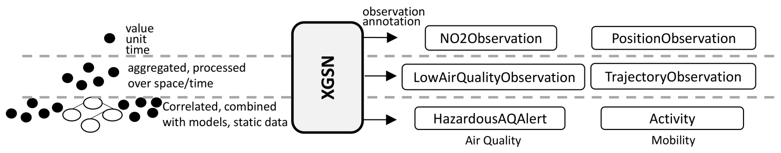


Fig. 2: XGSN annotations at different abstraction levels. From annotation of particular observations to high level concepts that aggregate, summarize or combine more data sources, the processing capabilities of XGSN allow defining annotation at different levels depending on the IoT use cases (i.e. air quality, mobility use cases, etc.)

to sensors thought as devices like thermistors or wind anemometers but more generally to any entity capable of observing a property of a feature of interest [10]. Therefore, interconnected objects and things can provide information about the events, facts and observations surrounding them. The SSN ontology is designed to be extended depending and according to the domain of use. With this in mind, XGSN takes this model as a the core of the metadata and observation annotations. We can see a summary of the main concepts of the ontology in Figure 3 along with extensions and examples of domain-specific vocabularies. The first important extension point is at the sensor level. Any virtual sensor (independently of being a device or other type of entity) is annotated in XGSN as a instance of ssn:Sensor[7] or a sub-class of it, like a thermistor or a capacitive bead. Individuals can also be sensors, as they can also observe events and observations surrounding them. The other two key extension points are related to the *observed property* and to which *feature of interest* it is associated. XGSN requires each sensor to observe at least one property, which can be a domain specific instance, such as the cf-prop:air_temperature of the dim:Temperature quantity in Figure 3. Each of these observed properties of a sensor corresponds to a field defined in a XGSN virtual sensor. Accordingly, each of these properties is associated to a certain feature of interest, e.g. the air or water surface in some geographical region, an observed person moving in a defined area, etc. XGSN also considers the location of the virtual sensor, which is annotated with geo-location vocabularies. Finally, other specific metadata such as accuracy, operating range, and other capabilities can be added as we will see in the following subsections.

In the case of observations, XGSN considers that every tuple generated by a virtual sensor includes one observation per field, considering that every field corresponds to an observed property, in terms of the SSN Ontology. Nevertheless, the data from virtual sensors can range from low level measurements to complex events built on top of other virtual sensors and external data, as seen in Section 2. We provide a summary of the main ontology concepts used at the observation level by XGSN in Figure 4. While for low level observations (e.g. a particular NO_2 measurement at a certain point in time) XGSN can annotate values using the quantities ontology, for higher level concepts the observation may be symbolic and represent an alert or an actionable event. In the following

[7] For brevity, we represent ontology URIs in its prefixed form, e.g. ssn: denotes `http://purl.oclc.org/NET/ssnx/ssn#`.

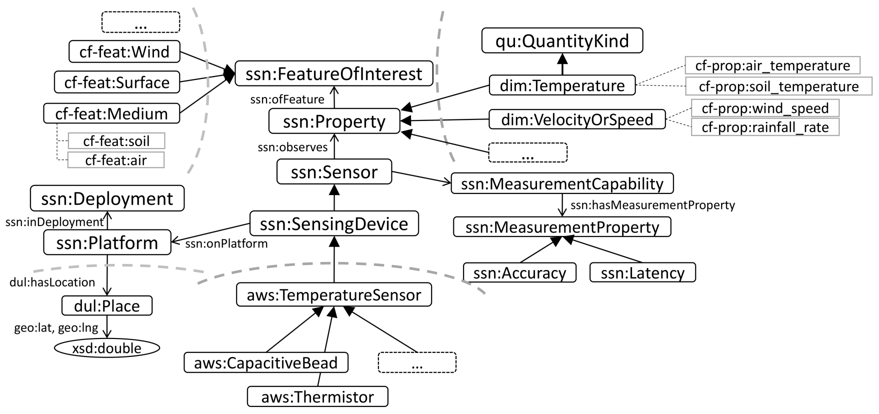


Fig. 3: Excerpt of some of the main ontology elements used by XGSN, based on the SSN ontology and QU/CF domain ontologies [17]. Notice the different extensions, marked by grayed dotted lines, with specialized ontologies defining specific sensors, features or observed properties.

sections we describe in more detail how this process occurs in XGSN, within the life-cycle of virtual sensors.

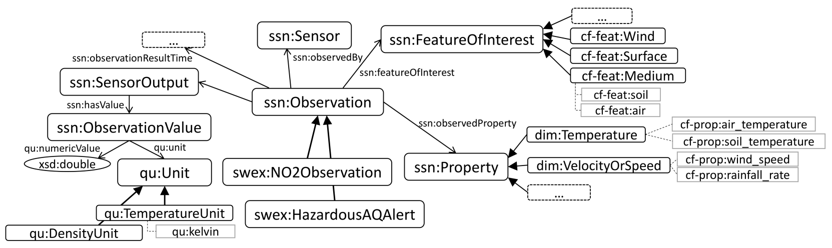


Fig. 4: Main ontology concepts used for observations in XGSN, based on the SSN ontology.

3.1 Virtual Sensor Registration

Virtual sensors in XGSN are set up using a configuration descriptor (an XML document as in Listing 1), and can be deployed and started at any time in an XGSN container. The output-structure element in the descriptor defines a set of fields for the virtual sensor. These fields are associated to observed properties according to the SSN ontology, and also correspond to the fields in the *query* element in the configuration. The wrapper information is also specified in this descriptor, and their parameters vary from wrapper to wrapper: e.g. the address of the data source, pull rates, security parameters, etc.

```
<virtual-sensor name="sens1" priority="10" >
  <processing-class>
    <class-name>org.openiot.gsn.vsensor.LSMExporter</class-name>
      ...
      <output-structure>
```

```
        <field name="temperature" type="double" />
        <field name="humidity" type="double" />
      </output-structure>
  </processing-class>
  <streams>
    <stream name="input1">
      <source alias="source1" sampling-rate="1" storage-size="1">
        <address wrapper="csv">
          ...
        </address>
        <query>select * from wrapper</query>
      </source>
      <query>select temp as temperature,humid as humidity, timed from source1</query>
    </stream>
  </streams>
</virtual-sensor>
```

Listing 1: Virtual sensor sample configuration in XGSN.

Each virtual sensor in a XGSN container has an associated sensor instance in an RDF cloud store (managed by the LSM middleware [16]) , i.e. a URI that uniquely identifies it. As we have seen in Figure 3, each sensor instance is connected to the respective properties, features of interest, location and other metadata needed by the system. All these metadata properties can be provided attached to the virtual sensor configuration, as in the example in Listing 2. In XGSN, we have limited the existing XML configuration solely to internal wrapper and processing class parameters, while all the high-level metadata of the sensors themselves is managed as RDF, and hence can be later shared as Linked Data. In the example, the sensor URI `http://openiot.eu/test/id/sensor/5010` observes air temperature, and has a number of other attributes including location, authorship, feature of interest, etc. Notice that the metadata can be extensible although XGSN internally requires only a handful of these, mainly the observed property, unit, feature and sensor type.

```
@base <http://openiot.eu/test/id/> .

<sensor/5010> rdf:type aws:CapacitiveBead,ssn:Sensor;
              rdfs:label "Sensor 5010";
              ssn:observes aws:air_temperature ;
              phenonet:hasSerialNumber <sensor/5010/serial/serial2> ;
              ssn:onPlatform <site/narrabri/Pweather> ;
              ssn:ofFeature <site/narrabri/sf/sf_narrabri> ;
              ssn:hasMeasurementProperty <sensor/5010/accuracy/acc_1> ;
              prov:wasGeneratedBy "AuthorName";
              DUL:hasLocation <place/location1>;
              lsm:hasSensorType <sensorType1>;

<sensor/5010/serial/serial2> rdf:type phenonet:SerialNumber;
                             phenonet:hasId "5010" .

<site/narrabri/Pweather>  rdf:type ssn:Platform ;
                          ssn:inDeployment <site/narrabri/deployment/2013> .
<site/narrabri/deployment/2013> rdf:type ssn:Deployment.

<sensor/5010/accuracy/acc_1> rdf:type ssn:Accuracy ;
                             qu:numericalValue "0.3"^^xsd:double ;
                             DUL:hasParameter phenonet:degreeCelsius .
```

Listing 2: Excerpt of a virtual sensor sample semantic descriptor in RDF used by XGSN. Prefixes ommitted.

The set up of the virtual sensor configuration and its annotation with the ontology constitutes the registration process, as illustrated in Figure 5. The registration and update of metadata can be performed using RESTful services provided by XGSN, simply by providing an RDF document with the required contents. In practice, the RDF metadata of virtual sensors is exposed as Linked Data by LSM, and can be queried or discovered by external applications and the upper layer components of the OpenIoT platform. Once the virtual sensor is registered and its metadata is available, it can produce (annotated) observations.

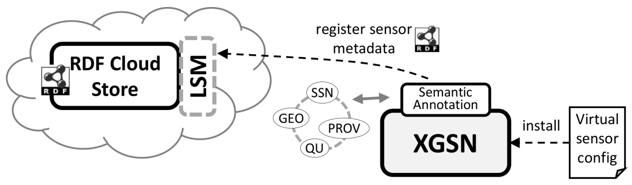


Fig. 5: Registration of a virtual sensor and annotation process performed in XGSN, storing the metadata through LSM.

3.2 Streaming Observation Annotations

Each time that the virtual sensor produces a value, XGSN annotates it and produces the corresponding observation according to the ontology model, as illustrated in Figure 6. Essentially, every time a tuple is produced in the virtual sensor stream (through the wrapper), a processing class automatically generates the RDF annotations that can be later transmitted to an RDF-aware data store or query processor. In the OpenIoT implementation, this processor is the LSM middleware, but it could even be processed by an RDF Stream Processor (RSP) such as CQELS [15], which is capable of evaluating continuous queries, extending the SPARQL language. Feeding any other continuous RDF query processor would follow a similar path: XGSN can feed the stream of RDF observations of an RSP. The advantage of this approach is that it decouples data acquisition from query processing, although it does add the complexity of having to manage both an RSP and XGSN. Also, RDF can be too verbose for certain stream processing tasks, depending on the volume and velocity of the data stream. Notice that XGSN observations could also be stored or processed through other channels, depending on the logic included in the virtual sensor processing class. For example, as it has been shown in [9], XGSN could expose a SPARQL-like (SPARQLStream) interface using R2RML mappings, through query rewriting techniques. Although for the OpenIoT reference implementation only the LSM integration has been wired, these types of extensions can be added in future stages.

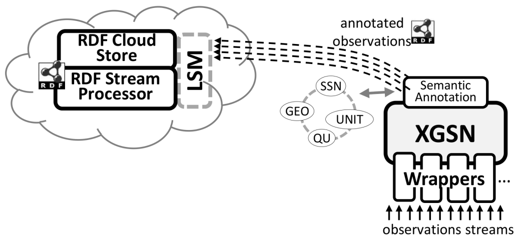


Fig. 6: XGSN annotation of observations produced by the virtual sensor, to the LSM middleware.

4 Architecture and Implementation

As already explained, the core abstraction in XGSN is the virtual sensor, which is hosted and deployed in a XGSN container. Each container is independent and runs its own set of virtual sensors, although different containers or instances may interchange data between them in a peer-to-peer fashion. Each container is structured in different layers, as detailed in Figure 7.

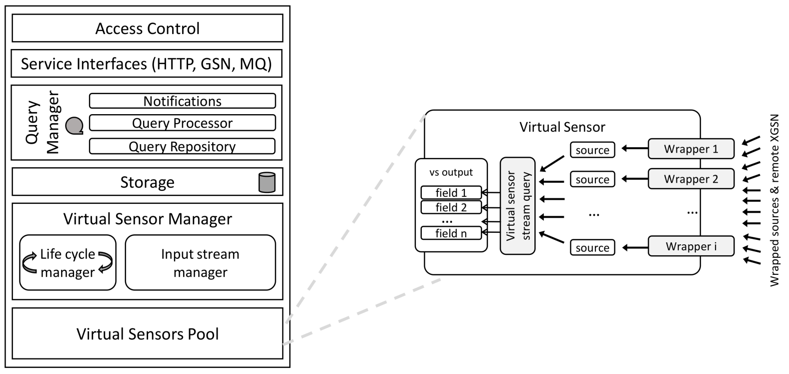


Fig. 7: XGSN container architecture, and virtual sensor acquisition and data stream provision [1].

A pool of deployed virtual sensors is administered by the virtual sensor manager. This includes handling the life-cycle of a virtual sensor (initialization, interactions, resources, disposal, etc.) and managing the incoming streams provided through the wrapper. The streams produced by each virtual sensor have an output structure, composed of one or more fields, which can be defined in terms of a continuous query that operates over one or more sources, each of them getting data through a wrapper. Notice that a wrapper can also encapsulate other (local or remote) virtual sensors, opening the possibility of having layered streams of data, as discussed in Section 2. Once the data is ready for processing, the storage layer handles persistent or temporary storage of the incoming data streams, depending on the virtual sensor configuration parameters. For some use cases where observations need to be archived this may include storage in a relational database. Alternatively, in stream processing scenarios this can be handled in a memory-only database or a stream processor. Next, at the query manager layer, the system can host running queries that are continuously evaluated by a processor, acting directly on the streams produced at the lower layers. The query capabilities are exposed though the service interfaces, currently implemented as an HTTP RESTful interface that can be accessed by external applications. Moreover, each XGSN instance can be accessed through a native interface (inter-XGSN communication) implemented on top of ØMQ (ZeroMQ, see Section 5)[8]. Finally, there is an access layer on top of the services, that allows defining permissions over the virtual sensors and the observations they produce. More details about the internal architecture of XGSN can be found in [1].

The system has been implemented mainly in Java, while some out-of-the-box wrappers are implemented in other languages. The entire project is open-source, and is available in Github, as a standalone project[9] and also as part of the OpenIoT platform[10], with an existing and growing community of users and developers. The project documentation in the Github site provides more detailed information about the installation, deployment, development and production use of the system.

5 XGSN in a Distributed Environment

As we have mentioned, one of the main features of XGSN is its capability to work on a fully distributed mode, in such a way that data processing is as close as possible to the data sources. At the same time, this allows virtual sensors in one XGSN instance to be fed from other remote virtual sensors, enabling the definition of high level events that can be semantically annotated. We have experimented in a controlled environment how a network of XGSN instances works in this distributed scenario. We were interested in the generation rates, processing rates and network usage in our experimentation.

First, we used an XML-based protocol of exchange of observations between instances, and then we implemented an alternative and more efficient mechanism based in ZeroMQ. The first protocol is available in two versions: a push-based one that works in a publish-subscribe manner, and a pull-based that can work even if the client XGSN is behind a NAT (does not have a public IP address). The main advantage of this protocol is that is easier to debug, as it is human-readable, and that is based on well supported standards (XML and HTTP), but the overhead for processing the data distribution is not negligible. The alternative protocol, using ØMQ and the Java serialization library Kryo, is similar to the push version of the HTTP wrapper, using a PUB and SUB sockets, as shown on Figure 8. A proxy takes care of forwarding the subscriptions and the data to allow the simultaneous use of internal (*inproc* sockets) and external connections (by IP

[8] ZeroMQ: `http://www.zeromq.org/`

[9] GSN github repository: `https://github.com/LSIR/gsn/`

[10] OpenIoT github repository: `https://github.com/OpenIotOrg/openiot`

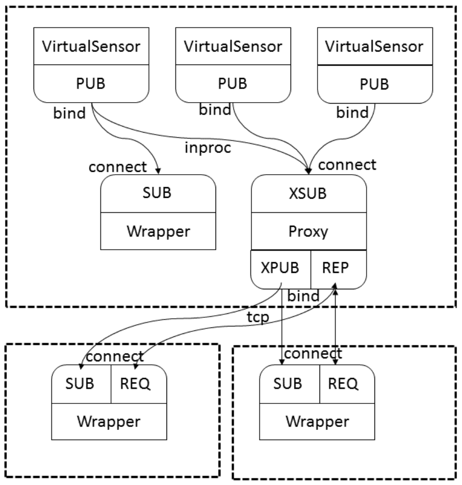


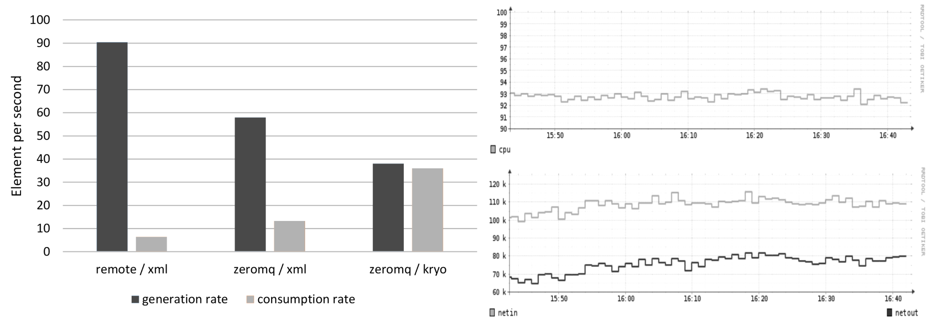
Fig. 8: XGSN ZeroMQ communication through asynchronous publish-subscribe sockets. Wrappers can subscribe to data of local or remote virtual sensors through a proxy.

address). It also serves as a directory, listing the available sensors and their data structure for external connections.

To evaluate the inter-server communication, we set up a use-case where each server is receiving or generating a stream of data and need to share it with all the other servers, in a kind of worst-case scenario. We deployed 24 XGSN servers, each on its own virtual machine, distributed over 9 physical machines. The virtual machines were provisioned with 2 cores (2.66GHz) and 3GB of RAM. Each server had 25 virtual sensors: one generating data every 10 ms and 24 connected to the other instances (including itself). The storage was kept in memory using the H2:mem database to reduce the disk writing overload.

In the first experiment, the remote HTTP XML wrapper was used to connect the 24 virtual sensors, in the second one the ZeroMQ wrapper was used with XML serialization and finally in the last experiment ZeroMQ with Kryo serialization. Each experiment lasted around 20 minutes during which two snapshots were taken.

In the first execution of the test-bed, using the remote wrapper and XML serialization, the CPU load of the virtual machines stayed around 36% and the network traffic was around 120kbps. The counter on the virtual sensor generating data indicated a rate of 90 element per seconds (Figure 9 (a)). This means XGSN needed 1 ms to generate the element and then wait for 10 ms before generating the next one. The network traffic is perfectly symmetric as every server is sending and receiving to, respectively from, all the others. From this results, it is clear that the network is saturated and cannot follow the element production. The second and third run, using the ZeroMQ wrappers presented a similar behavior regarding the CPU load and network traffic. CPU was almost at its maximum and network showed some differences in incoming versus outgoing



(a) **(b)**

Fig. 9: Generation of data items in one XGSN instance (a); and CPU and network usage during the experiment with ZeroMQ (b).

traffic (Figure 9 (b)). This can be explained by the distribution of the generation rate among the servers. All XGSN instances received the same amount of data (same incoming traffic), but the ones generating less elements had also less data to send (lower outgoing traffic).

In the experiment using XML serialization, the communication protocol being lighter than HTTP, it was possible to send and process twice as much elements per seconds per virtual sensor (see Figure 10). But for processing those elements, the CPU was also more solicited (almost 100%) and was not able to keep the production rate. Finally using the Kryo serialization, the network was saturated, and similarly to the previous experiment the CPU had less time producing elements, around 38 per second in each instance. In this last experiment we almost reached the maximum performance possible with our virtual machines limitation: 860 elements sent, received and processed per second. In summary, we see that we can reach a fairly reasonable processing throughput, even more with the ZeroMQ implementation, although at the cost of losing reliability (relaxing packet loss guarantees).

6 Related Work & Discussion

Several systems have been devised to provide access to data streams on the Web in the form of Linked Data. Early approaches, including the architectures described in [18] and [13], rely on bulk-import operations that transform the sensor data into an RDF representation that can be queried using SPARQL in memory, lacking scalability and real-time querying capabilities. The Semantic Sensor Web [20] pioneered in bringing sensor data to the Linked Open data cloud, although it served more as a static repository without streaming or dynamic change in the observation data. Semantic annotations have also been considered at the service layer, for example for registering new sensors to observation services in [8]. In [12] an SOS service with semantic annotations on sensor data is

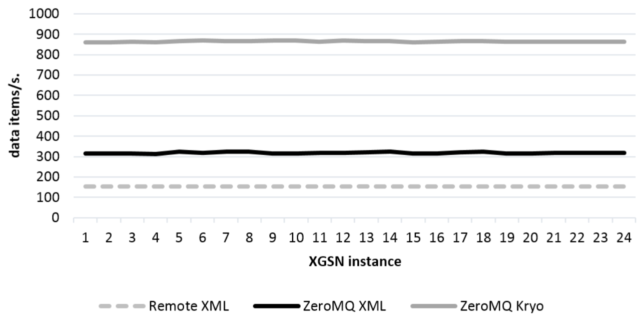


Fig. 10: Throughput in terms of data items received and processed per second, for the three configurations: Remote XML, ZeroMQ XML and ZeroMQ Kryo.

proposed, embedding terminology from an ontology in the XML of O&M and SensorML documents. In a different approach, the framework presented in [16] provides sensor data readings annotated with metadata from the Linked Data Cloud. This framework evolved into the LSM middleware that is now part of the OpenIoT platform, and that is used in conjunction with XGSN to manage the annotated sensor data and metadata. In most cases these systems have helped bringing sensor data to the (semantic) web, but resulted mainly in off-line archives of Linked Data, as opposed to the live annotation of sensor observations in XGSN. Moreover, we provide an end-to-end solution that manages the data from the acquisition layer up to the RDF and SPARQL data provision, through the LSM integration.

Other works have focused in the problem of continuous processing and querying over RDF streams, including CQELS [15], SPARQLStream [9], CSPARQL [5] or EP-SPARQL [2]. As explained in Section 3 these systems could be used in conjunction with XGSN, which can delegate the processing of the annotated observations to these systems, simply by implementing a processing class in a virtual sensor.

Nevertheless, even if our approach is capable of providing a solid semantic layer over IoT deployments, there are still many open challenges to tackle the problem of efficient stream processing. In the current OpenIoT implementation individual stream elements are annotated as they arrive, generating a non-negligeable volume of RDF which may be prohibitive for certain workloads. Stream compression techniques or virtualized RDF views over native data streams [9] are possible alternatives that have shown interesting results in other scenarios.

For handling continuous queries over streams, several Data Stream Management Systems (DSMS) have been designed and built in the past years, exploiting the power of continuous query languages and providing pull and push-based data access. Other systems, cataloged as complex event processors (CEP), emphasize on pattern matching in query processing and defining complex events from basic ones through a series of operators [11]. In recent years several commercial systems have implemented the CEP paradigms, including Oracle CEP[11], StreamBase[12], Microsoft StreamInsight[13], IBM InfoSphereStream[14] and Esper[15]. Some of these systems provide similar (or alternative) streaming data techniques as those of XGSN, and they could even be used as an alternative processing class for XGSN virtual sensors. However, none of them provides semantically rich annotation capabilities on top of the query interfaces. XGSN could allow plugging different types of commercial CEPs, replacing its internal data streaming core, but the lack of query standards (such as SQL in the database world) makes it difficult to design such a mechanism.

Finally, there has been a large amount of work in the IoT community regarding suitable protocols for device-to-device and device-to-server communication. While XGSN is designed as a protocol-agnostic middleware (new protocols can be supported through new wrappers) it will be important in the immediate future to natively support these protocols. For this it is also envisaged to allow deploying a constrained version of XGSN inside sensors and mobile devices, so that these *things* can transparently communicate with standard XGSN instances and therefore with the Web.

7 Conclusions

We have presented XGSN, an open-source middleware that is capable of collecting data from sensors and things in the real world, abstracted as virtual sensors, process them and publish the data using a semantic model based on the SSN ontology. We have shown in detail how the annotation process has been designed and implemented, for both the sensor metadata and the produced observations. We have also described a multi-layered scheme for defining observations at different levels of abstraction, for which XGSN provides a very flexible, extendable and scalable infrastructure. We have shown how the system goes beyond other existing sensor middleware, by adding the semantic aspect at its core, and by integrating its existing features and complementing them with the LSM framework for RDF storage and querying. XGSN is a fully functional and ready-to-use system, with a growing community of users and developers, and is now part of the wider OpenIoT platform. XGSN has been shown to be effective and useful in several different types of use cases, including air quality monitoring, environmental alpine experimentation, participatory sensing, smart agriculture, intelligent manufacturing, etc.

We plan to add several features to XGSN in the near future. First, we plan to make use of the semantic annotations of virtual sensors to allow enhancing the M2M communication among XGSN instances or even other *semantics-aware* applications. We also plan to integrate the semantic features not only with LSM

[11] Oracle:`http://www.oracle.com/technetwork/middleware/complex-event-processing/overview`
[12] StreamBase: `http://www.streambase.com/`
[13] StreamInsight: `http://msdn.microsoft.com/en-us/sqlserver/ee476990`
[14] InfoSphereStream: `http://www-01.ibm.com/software/data/infosphere/streams/`
[15] Esper: `http://esper.codehaus.org/`

but with other backends (not only cloud-based but also local-based). Another future direction is exploring parallelized execution of streaming data algorithms over the observation data (e.g. Spark[16] or Storm[17]), and how these can be combined with our system. There is also room for work integrating this system with mobile sensing and participatory sensing, where the mixture of incentives and privacy can be a challenging problem. While there is a need for having accurate data from a crowdsensing community, it is also important to protect privacy of individuals contributing to the dataset. Finally, we are re-designing the web services interfaces of XGSN, expanding its functionalities (e.g. including discovery, exploiting the semantic annotations for linkage, provenance support, etc.) and adhering to the Linked Data Platform.

Acknowledgments Partially supported by the OpenIoT FP7-287305 and the SNSF-funded Nano-Tera OpenSense2 projects.

References

1. Aberer, K., Hauswirth, M., Salehi, A.: A middleware for fast and flexible sensor network deployment. In: Proc. 32nd International Conference on Very Large Data Bases VLDB, pp. 1199–1202. VLDB Endowment (2006)
2. Anicic, D., Fodor, P., Rudolph, S., Stojanovic, N.: EP-SPARQL: a unified language for event processing and stream reasoning. In: Proc. 20th International Conference on World Wide Web, pp. 635–644 (2011)
3. Atzori, L., Iera, A., Morabito, G.: The internet of things: A survey. Computer networks 54(15), 2787–2805 (2010)
4. Bandyopadhyay, S., Sengupta, M., Maiti, S., Dutta, S.: Role of middleware for internet of things: A study. International Journal of Computer Science and Engineering Survey 2(3), 94–105 (2011)
5. Barbieri, D.F., Braga, D., Ceri, S., Della Valle, E., Grossniklaus, M.: Incremental reasoning on streams and rich background knowledge. In: Proc. 7th Extended Semantic Web Conference, pp. 1–15 (2010)
6. Barnaghi, P., Wang, W., Henson, C., Taylor, K.: Semantics for the internet of things: early progress and back to the future. International Journal on Semantic Web and Information Systems (IJSWIS) 8(1), 1–21 (2012)
7. Bizer, C., Heath, T., Berners-Lee, T.: Linked data-the story so far. International Journal on Semantic Web and Information Systems 5(3), 1–22 (2009)
8. Bröring, A., Janowicz, K., Stasch, C., Kuhn, W.: Semantic challenges for sensor plug and play. In: Proc. 9th International Symposium on Web and Wireless Geographical Information Systems. vol. 5886, pp. 72–86. Springer (2009)
9. Calbimonte, J.P., Jeung, H., Corcho, O., Aberer, K.: Enabling query technologies for the semantic sensor web. International Journal On Semantic Web and Information Systems (IJSWIS) 8(1), 43–63 (2012)
10. Compton, M., Barnaghi, P., Bermudez, L., García-Castro, R., Corcho, O., Cox, S., Graybeal, J., Hauswirth, M., Henson, C., Herzog, A., Huang, V., Janowicz, K., Kelsey, W.D., Phuoc, D.L., Lefort, L., Leggieri, M., Neuhaus, H., Nikolov, A., Page,

[16] `https://spark.apache.org/streaming/`

[17] `http://storm.incubator.apache.org/`

K., Passant, A., Sheth, A., Taylor, K.: The SSN ontology of the W3C semantic sensor network incubator group. Journal of Web Semantics 17, 25–32 (2012)

11. Cugola, G., Margara, A.: Processing flows of information: From data stream to complex event processing. ACM Computing Surveys 44(3), 15:1–15:62 (2011)
12. Henson, C., Pschorr, J., Sheth, A., Thirunarayan, K.: SemSOS: Semantic Sensor Observation Service. In: Proc. International Symposium on Collaborative Technologies and Systems CTS 2009. pp. 44–53. IEEE (2009)
13. Huang, V., Javed, M.: Semantic sensor information description and processing. In: Proc. 2nd International Conference on Sensor Technologies and Applications SENSORCOMM 2008. pp. 456–461. IEEE (2008)
14. Janowicz, K., Scheider, S., Pehle, T., Hart, G.: Geospatial semantics and linked spatiotemporal data–past, present, and future. Semantic Web 3(4), 321–332 (2012)
15. Le-Phuoc, D., Dao-Tran, M., Xavier Parreira, J., Hauswirth, M.: A native and adaptive approach for unified processing of linked streams and linked data. In: Proc. 10th International Semantic Web Conference ISWC, pp. 370–388. Springer (2011)
16. Le-Phuoc, D., Nguyen-Mau, H.Q., Parreira, J.X., Hauswirth, M.: A middleware framework for scalable management of linked streams. Web Semantics: Science, Services and Agents on the World Wide Web 16, 42–51 (2012)
17. Lefort, L., Henson, C., Taylor, K.: Semantic sensor network xg final report. Tech. rep., W3C SSN XG (2011), http://www.w3.org/2005/Incubator/ssn/XGR-ssn/
18. Lewis, M., Cameron, D., Xie, S., Arpinar, B.: ES3N: A semantic approach to data management in sensor networks. In: Proc. 1st International Workshop on Semantic Sensor Networks SSN 2006 (2006)
19. Pfisterer, D., Romer, K., Bimschas, D., Kleine, O., Mietz, R., Truong, C., Hasemann, H., Kroller, A., Pagel, M., Hauswirth, M., et al.: Spitfire: toward a semantic web of things. Communications Magazine, IEEE 49(11), 40–48 (2011)
20. Sheth, A., Henson, C., Sahoo, S.S.: Semantic sensor web. Internet Computing, IEEE 12(4), 78–83 (2008)
21. Song, Z., Cárdenas, A.A., Masuoka, R.: Semantic middleware for the internet of things. In: Internet of Things (IOT), 2010. pp. 1–8. IEEE (2010)