Iain Learmonth

Electronics Research Group

Room 304, Fraser Noble Building

King's College

Aberdeen

Machine Sensing Architecture

A sensor is a converter that measures a physical quantity and converts it into a signal which can be read by an observer. With electronic sensors, these signals can be read by the controller they are connected to via a hardware interface. Electronic sensors measuring temperature, humidity, wind speed, GPS location and a vast number of other properties are already widely deployed although few record their data using the Semantic Sensor Network Ontology and even fewer do so using the open standards that were used by this project.

With low-power networking becoming more accessible through the introduction of open standards such as IEEE 802.15.4, the number of deployments is growing. The majority of these systems currently use proprietary standards for the reporting of data and the data is typically stored in a relational DBMS. The proprietary standards used for reporting result in a vendor lock-in as both the sensor nodes and the logging server will need to be replaced in order to use an alternative system.

The wireless sensor network that was implemented used open standards in order to submit data to the logging server which then stored the data as part of an RDF graph. This allows the data to be accessed by other systems using open semantic web standards such as RDF/XML or SPARQL.

Architecture Overview

This first part of the project aims to acquire data from machine sensors using emerging open standards and store it using semantic web technologies. This application can be split into two distinct parts, the acquisition of data and the storage of the data. By splitting the application in this way and using open standards to communicate between them it allows for each module to do only a single task and do it well and also allows for either module to be replaced easily as the communication protocols are clearly defined and readily available.

The data acquisition part consists of wireless sensor nodes that submit their observations

to a server application and the storage part consists of a triplestore with a SPARQL interface. The SPARQL interface is used to bridge the data acquisition process with the storage. A diagram representing the architecture can be found in figure 1.1.

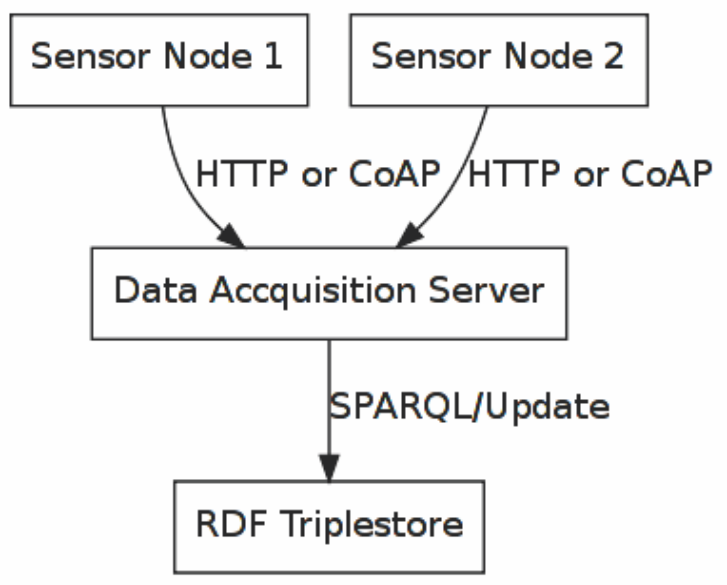

Figure 1.1: A diagram representing the architecture of the wireless machine sensing application

The Sensor Nodes

A large number of sensor network deployments use wireless "motes". These are purpose built wireless sensor platforms consisting of a microcontroller, transceiver, external memory, power source and one or more sensors. The disadvantage of using such platforms is that the price is very expensive compared to the general-purpose alternatives.

The Zigduino platform[1] is an Arduino-compatiable board with an integrated IEEE 802.15.4 radio. IEEE 802.15.4 defines a MAC layer for low-power low-bitrate wireless networks and 6LoWPAN, an IETF standard, allows for IPv6 to be used over IEEE 802.15.4. Communication with the server is facilitated by a USB 802.15.4 radio, the Jackdaw[2], which is supported by modern Linux kernels as a network interface allowing for either bridging, routing or simply "sniffing the wire". A picture of the Zigduino platform can be found in figure 1.2 and a picture of the Contiki Jackdaw can be found in figure 1.3.

The Zigduino has an integrated temperature sensor and also a series of GPIO pins, ADC pins and an I2C interface allowing arbitrary sensors to be connected to the system. The Zigduino has also been targeted by a port of the Contiki operating system[3] which provides an IPv6 networking stack and commonly used functions that will aid greatly in the development of applications targeting the Zigduino platform.

[1] http://logos-electro.com/zigduino/
[2] http://contiki.sourceforge.net/docs/2.6/a01799.html
[3] http://www.contiki-os.org/

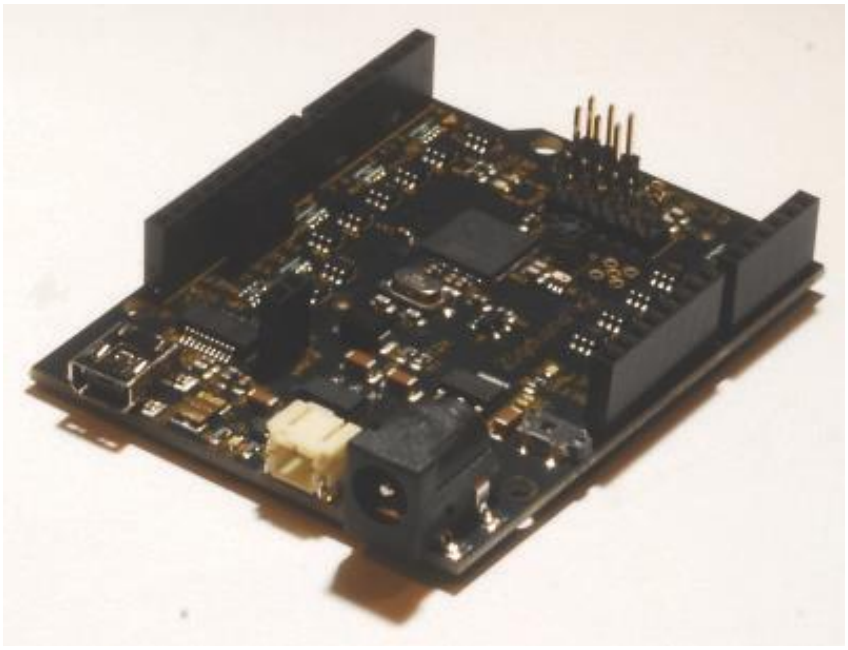

Figure 1.2.: A photo of the Zigduino platform

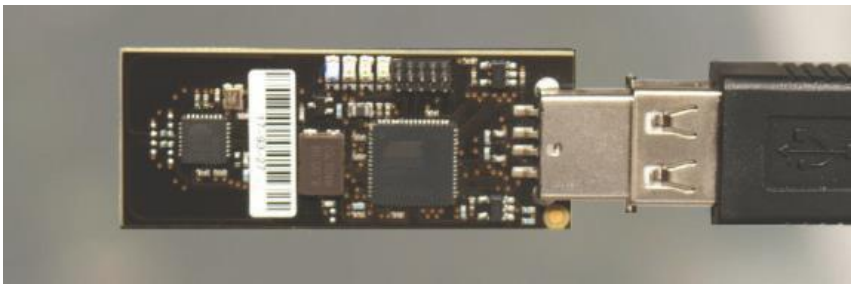

Figure 1.3.: A photo of the Contiki Jackdaw USB dongle

Storage System

RDF Vocabularies

The storing of sensor information and details of sensor observations within an RDF graph was the focus of the W3C Semantic Sensor Network Incubator Group[4] and the details of their findings are detailed in their final report. In order to record sensor information and the sensor observations almost all the required vocabulary is provided by the Semantic Sensor Network Ontology however the ability to store the numeric value and unit of the observation is not provided. It is common for the QUDT ontology to be used to provide these terms.

An application specific ontology was also created to extend these ontologies, providing

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

specific classes as subclasses of the SSN ontology class for each type of sensor whilst allowing applications to still reason with the data. A graph showing the relationships between these ontologies is shown in figure 1.4. The Units ontology is considered to be a part of the QUDT ontology.

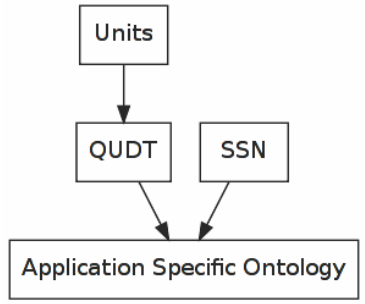

Figure 1.4.: A graph showing the import relationships between ontologies used for storing machine sensing data in the RDF graph

Semantic Sensor Network Ontology

The semantic sensor network ontology provides the ability to describe sensors and their observations. Each sensor node is represented as a ssn:SensingDevice which the ontology defines as "a device that implements sensing". Each sensor node observes a ssn:Property (property) of a ssn:FeatureOfInterest (feature of interest). In the sample application, the feature of interest is defined as a postcode district - the geographic area sharing the first half of a postcode - and the property being observed of each feature of interest is the outdoor temperature.

Each observation of the sensor nodes is represented as an ssn:Observation. The sensor nodes are linked to their observations through the ssn:madeObservation property and its inverse property of ssn:observedBy. Observations have values represented as instances of ssn:ObservationValue which are linked by the ssn:hasValue property. The values themselves require the use of the QUDT ontology to describe. An overview of the use of the SSN ontology is shown in figure 1.5.

Quantities, Units and Data Types Ontology

Each observation value has a numeric value and a unit. The QUDT property qudt:numericValue allows for a double literal to represent the numeric value for the observation and the QUDT property qudt:unit allows for the representation of the units used by the numeric value. The Units ontology, part of the QUDT ontology, provides units:DegreeFarenheit and units:DegreeCelcius to be used for the URI linked by the qudt:unit property. Figure 1.6 shows an overview of how the QUDT ontology is used.

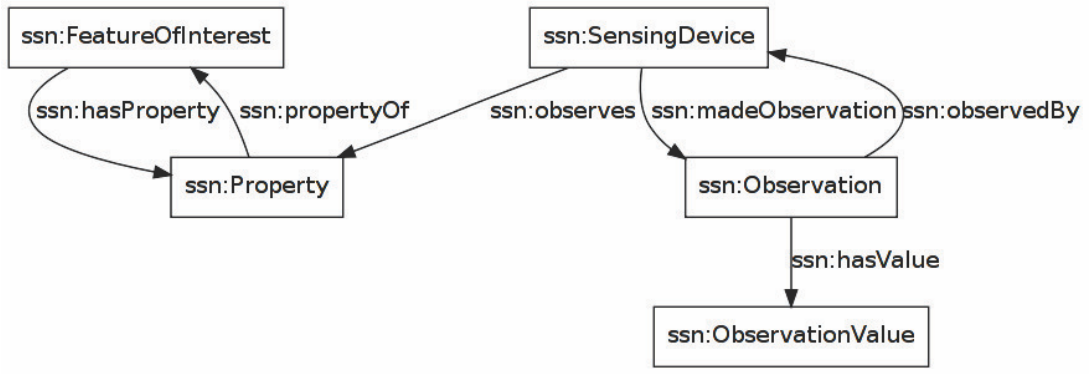

Figure 1.5.: An overview of the use of the SSN ontology in machine sensing

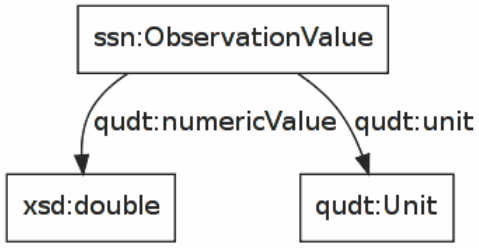

Figure 1.6.: An overview of the use of the QUDT ontology in machine sensing

An Example Graph using the SSN ontology and QUDT

The Turtle document in listing 1.1 shows an example RDF graph using the ontologies that were used in this project. It is representative of the RDF graph generated by the implementation of the wireless sensor network.

```
@prefix db:      <> .
@prefix xsd:     <http://www.w3.org/2001/XMLSchema#> .
@prefix geo:     <http://www.w3.org/2003/01/geo/wgs84_pos#> .
@prefix qudt: <http://qudt.org/1.1/schema/qudt#> .
@prefix rdfs: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix ssn:     <http://www.w3.org/2005/Incubator/ssn/ssnx/ssn#> .

db:feature/Aberdeen a ssn:FeatureOfInterest;
  rdfs:label "Aberdeen, Scotland";
  ssn:hasProperty db:feature/Aberdeen/Temperature;
  geo:location [
    geo:latitude "57.13"^^xsd:double;
    geo:longitude "2.1"^^xsd:double
    ];
  .

db:feature/Aberdeen/Temperature a ssn:Property;
  rdfs:label "Temperature in Aberdeen, Scotland";
  ssn:isPropertyOf db:feature/Aberdeen;
  .
```

```
21
22 db:sensor/1 a ssn:SensingDevice;
23   rdfs:label "Sensing Device 1";
24   ssn:observes db:feature/Aberdeen/Temperature;
25   .
26
27 db:observation/1 a ssn:Observation;
28   rdfs:label "Sensor Observation 1";
29   ssn:observedBy db:sensor/1;
30   ssn:observedProperty db:feature/Aberdeen/Temperature;
31   ssn:hasValue [ qudt:numericValue "23.0"^^xsd:double ];
32   ssn:hasSamplingTime "2013-01-01T20:05:00+00:00"^^xsd:dateTime;
33   .
34
35 db:observation/2 a ssn:Observation;
36   rdfs:label "Sensor Observation 2";
37   ssn:observedBy db:sensor/1;
38   ssn:observedProperty db:feature/Aberdeen/Temperature;
39   ssn:hasValue [ qudt:numericValue "22.8"^^xsd:double ];
40   ssn:observationSamplingTime "2013-01-01T20:10:00+00:00"^^xsd:dateTime
       ;
41   .
42
43 db:observation/3 a ssn:Observation;
44   rdfs:label "Sensor Observation 3";
45   ssn:observedBy db:sensor/1;
46   ssn:observedProperty db:feature/Aberdeen/Temperature;
47   ssn:hasValue [ qudt:numericValue "23.0"^^xsd:double ];
48   ssn:observationSamplingTime "2013-01-01T20:15:00+00:00"^^xsd:dateTime
       ;
49   .
```

Listing 1.1: Example RDF graph using the SSN ontology and QUDT ontology representative of the RDF graph generated by the implementation of the wireless sensor network

The Triplestore

The storage and linked data interface are two seperate applications, although typically deployed together. The storage provided by Apache Fuseki[5], an RDF triplestore, is a standalone server and the Pubby[6] linked data frontend is a Java web application that requires a servlet container.

The Apache Fuseki server was deployed on a Linux machine and set up with a configuration file. In order to not conflict with Apache Tomcat, which would be needed later, it was configured to listen on the TCP port 3030 and only on the loopback interface of the machine as if it were listening on any external interface it

[5] https://jena.apache.org/documentation/serving_data/index.html
[6] http://wifo5-03.informatik.uni-mannheim.de/pubby/

would therefore be possible for anyone with access to the TCP port to make arbitrary SPARQL/Update queries on the dataset.

The server is configured to have timeouts on SPARQL queries in order to protect itself. It may be possible to execute excessively large queries through the SPARQL endpoint that would place the server under heavy load rendering it almost useless to others attempting to use the service.

A single Fuseki service is defined for sensor observations. In RDBMS terms, this would be equivalent to a database running on the server being a self-contained collection of data that may be configured to behave differently to other databases within the server but still being served by the same application.

The Fuseki service was configured to use TDB[7] as its storage engine. A triplestore storage engine is almost identical in functionality to a storage engine for a RDBMS in that it provides the persistent storage on disk and a transaction engine to ensure data consistency. TDB is a component of Jena and thus readily available in Apache Fuseki.

With Fuseki running and configured, a RDF triplestore with persistent storage, a SPARQL query and SPARQL/Update query endpoint to the stored RDF graph was available.

Linked Data Interface

The Pubby linked data frontend was deployed as an application in Apache Tomcat[8].

Other than prefix declarations, there are no real restrictions on the data that can be stored within the triplestore and presented via the linked data frontend. Even where prefixes haven't been declared, it would be possible to use full URIs and still store the data. This is, in contrast to an RDBMS where a database schema would have been necessary by this point, limiting the scope of the data that could be usefully stored within the database.

As Apache Tomcat also gave access to the manager application which should not be accessiable from the outside world, it was also configured to listen on only the loopback interface. The Apache HTTP Server[9] was used along with mod_proxy[10] and mod_proxy_http[11] to reverse proxy incoming requests to the path for Pubby on the Apache Tomcat server.

[7] https://jena.apache.org/documentation/tdb/index.html
[8] https://tomcat.apache.org/
[9] https://httpd.apache.org/
[10] https://httpd.apache.org/docs/2.2/mod/mod_proxy.html
[11] https://httpd.apache.org/docs/2.2/mod/mod_proxy_http.html

Testing

Whilst these solutions were considered stable by their distributors, to ensure that no mistakes had been made during the configuration and deployment stages some limited testing was performed.

The first test performed was to check that the SPARQL endpoints were functioning as expected. The Apache Fuseki server contains a built-in webserver that provides a pair of HTML forms with one each for both SPARQL queries and SPARQL/Update queries. This was used to insert data into the triplestore using an INSERT DATA query and then the data was retrieved by performing a SELECT query. This worked successfully.

The second test performed was to check that the triplestore's storage was persisting as expected. The server was terminated and it was ensured that this had happened by checking for any remaining processes belonging to the server. Upon restarting, the same SELECT query was performed as in the previous test. The data was again retrieved showing that the persistent store was working correctly.
All the tests completed as expected and no problems were found.