A semantic platform for integrated human and machine sensing

Iain R. Learmonth
<iain.learmonth.09@aberdeen.ac.uk>

Supervisor: Prof. Peter Edwards
Co-Supervisor: Prof. Gorry Fairhurst (School of Engineering)

Computing Science Department
University of Aberdeen
Kings's College
Aberdeen AB24 3UE

Honours Project, Ac. Year 2012/13

Acknowledgements

I would like to thank my supervisor, Prof. Pete Edwards, for accepting my self-submitted project and for his guidance in the completion of this project. I would also like to thank Chris Baillie who was present at the weekly supervisor meetings for his input to discussions relating to this project, especially in the area of ontology development.

I would like to thank the project's co-supervisor, Prof. Gorry Fairhurst, for his input on the technical networking details and for the use of the wireless sensing equipment.

I would like to thank the denizens of Room 205, Meston Building, for helpful discussions. Cyril Danilevski was particularly useful and also proofread some of the earlier writing.

Finally, I would like to thank Bram Moolenaar for creating Vim and Leslie Lamport for creating LaTeX as without these software packages this report would have been a lot harder to compile.

Declaration

I declare that this document and the accompanying code has been composed by myself, and describes my own work, unless otherwise acknowledged in the text. It has not been accepted in any previous application for a degree. All verbatim extracts have been distinguished by quotation marks, and all sources of information have been specifically acknowledged.

Signed: ______________________________

Iain R. Learmonth

Date: ______________________________

Contents

Acknowledgements **ii**

Declaration **iii**

1. Introduction **1**

1.1. Project Overview . . . 1
1.2. Project Motivation . . . 1
1.3. The Internet of Things . . . 2
1.4. Project Constraints . . . 2

2. Semantic Web Technologies **3**

2.1. The Semantic Web . . . 3
2.2. Resource Description Framework . . . 3
2.3. An Example Graph . . . 3
2.4. The Semantic Sensor Network Ontology . . . 4
2.5. Complementary Ontologies . . . 5

3. A Machine Sensing Architecture **6**

3.1. Architecture Overview . . . 6
3.2. The Sensor Nodes . . . 7
3.3. Storage System . . . 8
3.3.1. RDF Vocabularies . . . 8
3.3.2. The Triplestore . . . 11
3.3.3. Linked Data Interface . . . 12
3.3.4. Testing . . . 13
3.4. Hypertext Transfer Protocol Client . . . 13
3.4.1. An Analysis of HTTP Versions . . . 13
3.4.2. Design and Implementation Detail . . . 15
3.4.3. Testing . . . 16
3.5. Constrained Access Protocol Client . . . 17
3.5.1. Benefits of CoAP over HTTP . . . 17
3.5.2. Design and Implementation Detail . . . 18
3.5.3. Testing . . . 18

4. Integrated Sensing **20**

4.1. The uktemp.net Application . . . 20
4.2. Architecture Overview . . . 20

4.3. Ontology Development . . . 21
4.4. Data Acquisition Modules Design . . . 23
4.4.1. Twitter . . . 23
4.4.2. Weather Underground . . . 24
4.5. Web Frontend Design . . . 24
4.6. Implementation Detail . . . 25
4.6.1. Base Architecture . . . 25
4.6.2. Data Acquisition Modules . . . 25
4.6.3. Web Frontend . . . 26
4.7. Testing . . . 26
4.8. Results . . . 27

5. Evaluation 31
5.1. HTTP and CoAP for Wireless Sensor Platforms . . . 31
5.2. Suitability of Semantic Web Technologies . . . 33
5.3. Crowd-sourcing from Twitter . . . 35

6. Conclusion 37
6.1. Achievements . . . 37
6.2. Future Work . . . 37
6.3. Summary . . . 38

. Bibliography 38

A. Wireless Sensor Network Deployment Manual 41
A.1. Pre-Requisites . . . 41
A.2. Configuring and Flashing Sensor Nodes . . . 41
A.3. Configuring the logging server . . . 42

B. uktemp.net Application - User Manual 43
B.1. Using the uktemp.net map frontend . . . 43
B.2. Submitting an observation via Twitter . . . 43
B.3. Using the SPARQL endpoint and Linked Data interface . . . 44

C. uktemp.net Application - Maintainence Manual 47
C.1. Pre-requisites . . . 47
C.2. Installation Guide . . . 47
C.2.1. RDF Triplestore Setup . . . 47
C.2.2. Optional: Linked Data Frontend . . . 48
C.2.3. The uktemp.net Application . . . 48
C.2.4. Reverse-proxy Configuration . . . 49
C.3. Creating a new module . . . 49
C.3.1. Data Acquisition Module . . . 49
C.3.2. Presentation Module . . . 51
C.4. Notes . . . 51
C.4.1. Deploying a new version . . . 51

C.4.2. Running multiple instances 52
C.5. Code Overview 52

D. Fuseki Configuration File 68

E. Pubby Configuration File 69

F. Apache VirtualHost Configuration File 71

G. Human Sensing Ontology Documentation 72
G.1. Introduction 72
G.2. Classes 73
G.2.1. HumanTemperatureSensingDevice 73
G.2.2. HumanPainSensingDevice 73
G.2.3. HumanAccelerationSensing 73
G.2.4. HumanVisualSensingDevice 73
G.2.5. HumanTouchSensingDevice 74
G.2.6. HumanIlluminanceSensingDevice 74
G.2.7. HumanTasteSensing 74
G.2.8. HumanImageRecognitionSensing 74
G.2.9. HumanTimeSensingDevice 75
G.2.10. HumanImageRecognitionSensingDevice 75
G.2.11. HumanVisualSensing 75
G.2.12. HumanSoundIntensitySensing 75
G.2.13. HumanSoundRecognitionSensing 76
G.2.14. HumanTouchSensing 76
G.2.15. HumanAudioSensingDevice 76
G.2.16. HumanLuminousIntensitySensing 76
G.2.17. HumanSmellSensing 77
G.2.18. HumanAudioSensing 77
G.2.19. HumanTemperatureSensing 77
G.2.20. HumanIlluminanceSensing 77
G.2.21. HumanAccelerationSensingDevice 78
G.2.22. HumanTimeSensing 78
G.2.23. HumanBalanceSensing 78
G.2.24. HumanSensingDevice 78
G.2.25. HumanSmellSensingDevice 79
G.2.26. HumanTasteSensingDevice 79
G.2.27. HumanPainSensing 79
G.2.28. HumanSoundRecognitionSensingDevice 79
G.2.29. HumanSoundIntensitySensingDevice 80
G.2.30. HumanBalanceSensingDevice 80
G.2.31. HumanLuminousIntensitySensingDevice 80

H. Qualitative Units Ontology Documentation **81**
H.1. Introduction . . . 81
H.2. Classes . . . 81
H.2.1. QualitativeInformationEntropyUnit . . . 81
H.2.2. QualitativeHumanCountUnit . . . 82
H.2.3. QualitativeCurvatureUnit . . . 82
H.2.4. QualitativeDataRateUnit . . . 82
H.2.5. QualitativeAngularMassUnit . . . 82
H.2.6. QualitativeTimeUnit . . . 83
H.2.7. QualitativeUnit . . . 83
H.2.8. QualitativeIlluminanceUnit . . . 84
H.2.9. QualitativeMomementumUnit . . . 84
H.2.10. QualitativeMassUnit . . . 84
H.2.11. QualitativeCountUnit . . . 85
H.2.12. QualitativePermeabilityUnit . . . 85
H.2.13. QualitativeForceUnit . . . 85
H.2.14. QualitativeDynamicViscosityUnit . . . 86
H.2.15. QualitativeVelocityUnit . . . 86
H.2.16. QualitativeRespiratoryRateUnit . . . 86
H.2.17. QualitativeTurbidityUnit . . . 86
H.2.18. QualitativeConcentrationUnit . . . 87
H.2.19. QualitativePlaneAngleUnit . . . 87
H.2.20. QualitativeConductanceUnit . . . 87
H.2.21. QualitativeAreaUnit . . . 87
H.2.22. QualitativeTemperatureUnit . . . 88
H.2.23. QualitativeWorkUnit . . . 88
H.2.24. QualitativeResistanceUnit . . . 88
H.2.25. QualitativeLengthUnit . . . 89
H.2.26. QualitativeFrequencyUnit . . . 89
H.2.27. QualitativeBendingMomentUnit . . . 89
H.2.28. QualitativeHeartRateUnit . . . 89
H.2.29. QualitativeAccelerationUnit . . . 90
H.2.30. QualitativeLuminousIntensityUnit . . . 90
H.2.31. QualitativeVolumeUnit . . . 90
H.2.32. QualitativeVideoFrameRateUnit . . . 90
H.2.33. QualitativeThermalConductivityUnit . . . 91
H.3. Data Properties . . . 91
H.3.1. qualitativeValue . . . 91

I. UKTemp.net Units Ontology Documentation **92**
I.1. Introduction . . . 92
I.2. Classes . . . 92
I.2.1. UKPostcodeDistrict . . . 92
I.2.2. WundergroundTemperatureSensingDevice . . . 93
I.2.3. TwitterHumanTemperatureSensingDevice . . . 93

List of Figures

3.1. A diagram representing the architecture of the wireless machine sensing application . . . 7
3.2. A photo of the Zigduino platform . . . 8
3.3. A photo of the Contiki Jackdaw USB dongle . . . 8
3.4. A graph showing the import relationships between ontologies used for storing machine sensing data in the RDF graph . . . 9
3.5. An overview of the use of the SSN ontology in machine sensing . . . 10
3.6. An overview of the use of the QUDT ontology in machine sensing . . . 10

4.1. An overview of the uktemp.net application architecture . . . 21
4.2. An example of the use of the Qualitative Units Ontology . . . 22
4.3. The temperature as reported by Weather Underground in AB24 over the 10 day trial of the uktemp.net application . . . 28
4.4. Bar chart showing word frequencies for human qualitative observations of temperature in AB24 . . . 29
4.5. Box plots showing the distribution of machine measured temperatures close to qualitative observations . . . 30

B.1. Screenshot of Twitter web frontend with the box used to compose new tweets highlighted . . . 44
B.2. An example tweet submitting an observation to the uktemp.net application 45

List of Tables

2.1. Complementary vocabularies used 5

3.1. Minimum request sizes for `GET` and `POST` requests for HTTP versions 0.9, 1.0 and 1.1 14

Listings

2.1. Example RDF graph with a single node describing the author 4

3.1. Example RDF graph using the SSN ontology and QUDT ontology representative of the RDF graph generated by the implementation of the wireless sensor network 10

4.1. SPARQL query used to select temperatures and sampling times of observations of outdoor temperature in AB24 by Weather Underground 27

E.1. Pubby Configuration File 69

F.1. Apache VirtualHost Configuration File 71

1. Introduction

This report will document the design and construction of a semantic platform for integrating the "sensor observations" from both traditional electronic sensors and crowd-sourced human sensors and the findings from experimentation with the platform.

1.1. Project Overview

The goal of this project is to produce a software solution to allow for the acquisition of data, in the form of sensor observations, from both electronic and human sensors and automatically add semantic annotations to these observations relating to the sensors that provided them. Semantic reasoning on the data should be able to be performed in exactly the same way for both electronic and human sensors, requiring no modification of existing products that work with elecronic sensors to now also work with human sensors.

The software solution will contain a storage solution that provides an interface for querying observations, inserting observations and annotating them. Using this interface, bridges will exist to allow both machines and humans to submit their observations to the storage.

The end result will allow for observations close to the same temporal and geographical points to be matched. This will allow for the production of a "best-guess" based on the available data and also the production of a mapping between the language used in qualitative values submitted by humans and the quantitative values submitted by machines where both are available.

1.2. Project Motivation

Emerging semantic web technologies, low-level networking technologies and low-cost wireless sensor platforms are now all coming to maturity and starting to see deployment. The primary motivation behind this project is to evaluate these technologies and their suitability for the application of semantic sensing.

The secondary motivation for this project is to explore the area of human sensing. Machine sensor deployments are not currently widespread and are still expensive enough for the price to be a barrier for many research projects. Humans are already widely deployed with access to smart devices that can be used to submit sensor observations.

1.3. The Internet of Things

This project leverages the Internet of Things throughout in collecting sensor observations. In the past, the Internet consisted mainly of servers and personal computers but increasingly now smaller, less powerful, devices are being connected to the Internet with new technologies. The Internet of Things allows these previously standalone systems to have their parts uniquely identified by an address and communicated with by any other host on the Internet.

These new devices can provide rich data about their environment which can be used to provide ambient intelligence and, given the two-way communication possible on the Internet, remote control of the devices too.

In the first part of this project, a wireless sensor network is deployed with each sensor node being Internet connected and using Internet technologies to communicate with the data logger. In the second part of this project, it is possible for humans to submit observations to the data logger by way of devices such as smartphones, tablets and in fact any platform capable of running a Twitter client.

1.4. Project Constraints

As an honours project, there are financial constraints on the project. As such, a mass deployment of electronic sensors is not an option, nor is providing any financial incentives to human participants. Whilst an actual use case of the platform documented here would likely have these options, they are not necessary to perform basic testing and experimentation and are also not necessary for the development of the platform itself.

2. Semantic Web Technologies

There has been vast amounts of research and development in the Semantic Web field thus, solutions to a number of problems that this project presents already exist.

2.1. The Semantic Web

According to the World Wide Web Consortium (W3C), "The Semantic Web provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries." The term was coined by Tim Berners-Lee[2], the inventor of the World Wide Web and director of the W3C, which oversees the development of proposed Semantic Web standards. He defines the Semantic Web as "a web of data that can be processed directly and indirectly by machines."

2.2. Resource Description Framework

A core semantic web technology, the Resource Description Framework (RDF), is a family of W3C specifications[3] originally designed as a metadata model. It has come to be used as a general method for conceptual description or modeling of information that is implemented in web resources, using a variety of syntax formats.

RDF represents knowledge as triples composed of a subject, a predicate and an object[18]. Examples would be "Alice knows Bob" or "Alice is 35". A triplestore is a purpose-built database designed specifically for the storage and retrieval of such triples. These triplestores can also be queried and updated using a query language, SPARQL[22], in a way similar to traditional relational databases.

Objects can either be RDF nodes or literals. If an object is linking to another RDF node, the URI of the node is given. If the object is a literal, it can either be a simple string or it can use a datatype from XML such as double, datetime or duration.

2.3. An Example Graph

In order to demonstrate how an RDF graph stores representations of real life objects, an example graph representing the author is shown in listing 2.1. The example is given in

Turtle format although other serialisations exist such as RDF/XML.

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .

<http://irl.sdf.org/rdf/#me> a foaf:Person;
    foaf:family_name "Learmonth";
    foaf:givenname "Iain";
    foaf:homepage <http://irl.sdf.org/>;
    foaf:mbox <mailto:i.learmonth@abdn.ac.uk>;
    foaf:mbox_sha1sum "c09c052342567b791bd5f6babe7a9c5f1437354b";
    foaf:name "Iain Learmonth";
    foaf:phone <tel:+441224272799>;
    foaf:schoolHomepage <http://www.abdn.ac.uk/>;
    foaf:title "Mr";
    foaf:workplaceHomepage <http://www.abdn.ac.uk/>;
    foaf:openid <http://irl.sdf.org/>;
    owl:sameAs <http://dbtune.org/last-fm/shiftout>;
    owl:sameAs <https://wm.sdf.org/status/index.php/user/188>;
    .
```

Listing 2.1: Example RDF graph with a single node describing the author

For properties such as *foaf:givenname*, a person's given name, a literal is used to simply provide a value. For properties such as *foaf:homepage*, a URI is used to refer to the object. In this case, it refers to a web page which is retrievable from the web but it is possible that semantic annotations of the webpage are also available. The URI used to describe the author is not a document available on the web, but a person in the physical world, and so when such as URI is requested it is typically redirected to a page describing the resource using a "See Other" redirect.

2.4. The Semantic Sensor Network Ontology

Within the semantic web, vocabularies define the concepts and relationships used to describe and represent an area of concern. It is greatly advantageous to the semantic web if the vocabularies used are standardised and adopted widely, as opposed to having multiple competing vocabularies as this allows for datasets to be easily linked with other datasets to produce larger ones.

The Semantic Sensor Network Ontology provides the vocabulary for describing sensors, their observations and some related concepts[24]. It is developed by the W3C Semantic Sensor Networks Incubator Group (SSN-XG).

Like most vocabularies in the semantic web, the Semantic Sensor Network Ontology is designed to be as general as possible to allow its application to the maximum number of use cases. A result of this generalisation is that it increases the complexity of the ontology. As such, an example of existing use[4] was used along the ontology's specification and the visualisations available in the SSN-XG's final report[17] in order to fully understand it before attempting any implementation using the ontology.

2.5. Complementary Ontologies

In order to keep the Semantic Sensor Network Ontology as general as possible, it does not describe domain concepts. This includes concepts such as time, location and the properties being measured by sensors. These are intended to be included from other ontologies[24].

As such, a number of other ontologies were selected to complement the Semantic Sensor Network Ontology. These other ontologies are listed in table 2.1.

Prefix	Name
xsd:	XML Datatypes[1]
qudt:	Quantities, Units, Dimensions and Data Types[2]
rdfs:	RDF Schema[3]

[1] `http://www.w3.org/2001/XMLSchema#`
[2] `http://qudt.org/1.1/schema/qudt#`
[3] `http://www.w3.org/1999/02/22-rdf-syntax-ns#`

Table 2.1.: Complementary vocabularies used

The XML datatypes ontology allows the use of datatypes such as doubles and a datatype representing time. The QUDT ontology allows the sensor observation values to be labelled as having numeric values and permits the annotation of those values with the units they are recorded in. The RDF schema provides some properties for giving human readable descriptions to objects within the RDF graph.

3. A 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[17].

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.

3.1. Architecture Overview

This first part of the project aims to accquire 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 accquisition 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 accquisition 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 accquisition process with the storage. A diagram representing the architecture can be found in figure 3.1.

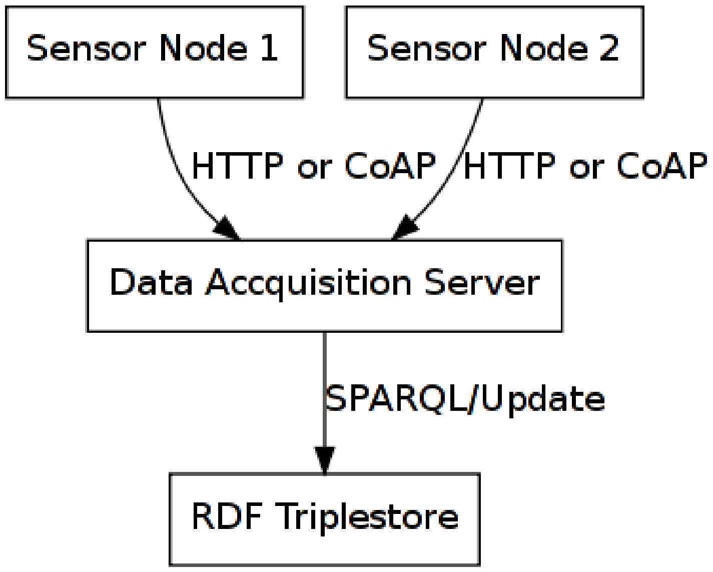

Figure 3.1.: A diagram representing the architecture of the wireless machine sensing application

3.2. 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 3.2 and a picture of the Contiki Jackdaw can be found in figure 3.3.

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/`

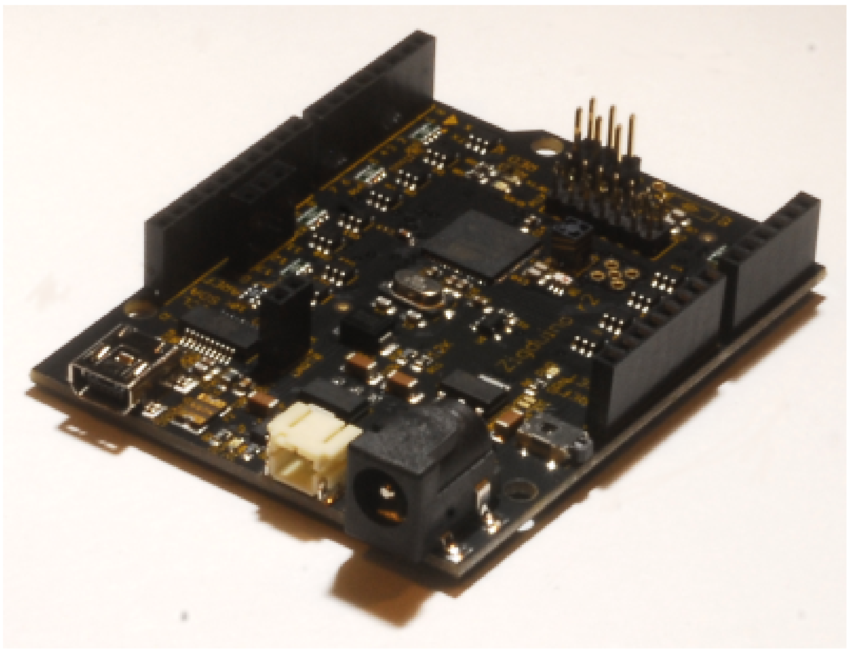

Figure 3.2.: A photo of the Zigduino platform

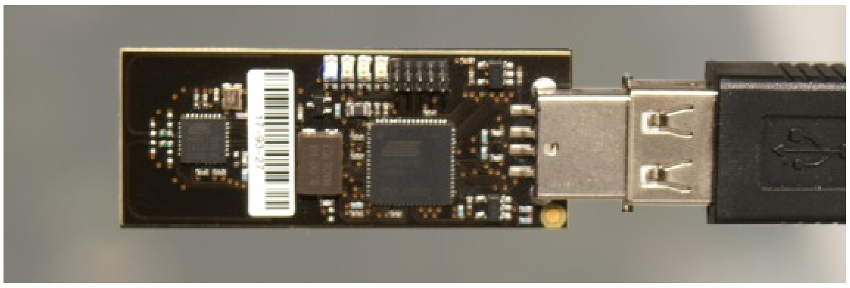

Figure 3.3.: A photo of the Contiki Jackdaw USB dongle

3.3. Storage System

3.3.1. 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[17]. 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[4].

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 3.4. The Units ontology is considered to be a part of the QUDT ontology.

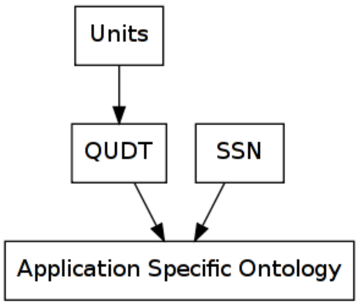

Figure 3.4.: A graph showing the import relationships between ontologies used for storing machine sensing data in the RDF graph

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 3.5.

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 3.6 shows an overview of how the QUDT ontology is used.

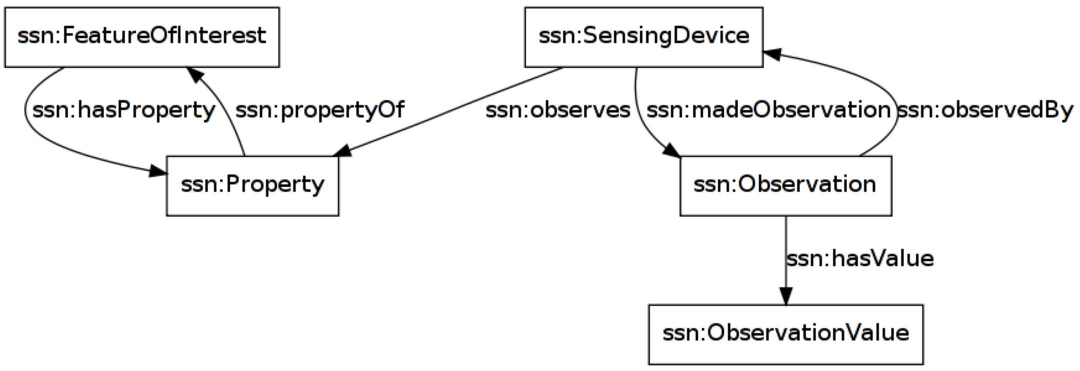

Figure 3.5.: An overview of the use of the SSN ontology in machine sensing

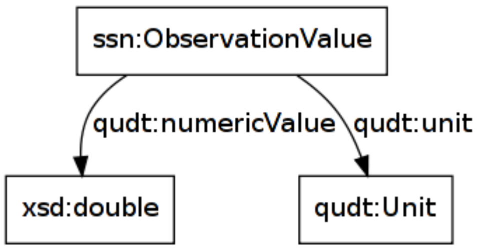

Figure 3.6.: An overview of the use of the QUDT ontology in machine sensing

An Example Graph using the SSN ontology and QUDT

The Turtle document in listing 3.1 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;
  .
```

```

db:sensor/1 a ssn:SensingDevice;
  rdfs:label "Sensing Device 1";
  ssn:observes db:feature/Aberdeen/Temperature;
  .

db:observation/1 a ssn:Observation;
  rdfs:label "Sensor Observation 1";
  ssn:observedBy db:sensor/1;
  ssn:observedProperty db:feature/Aberdeen/Temperature;
  ssn:hasValue [ qudt:numericValue "23.0"^^xsd:double ];
  ssn:hasSamplingTime "2013-01-01T20:05:00+00:00"^^xsd:dateTime;
  .

db:observation/2 a ssn:Observation;
  rdfs:label "Sensor Observation 2";
  ssn:observedBy db:sensor/1;
  ssn:observedProperty db:feature/Aberdeen/Temperature;
  ssn:hasValue [ qudt:numericValue "22.8"^^xsd:double ];
  ssn:observationSamplingTime "2013-01-01T20:10:00+00:00"^^xsd:dateTime
    ;
  .

db:observation/3 a ssn:Observation;
  rdfs:label "Sensor Observation 3";
  ssn:observedBy db:sensor/1;
  ssn:observedProperty db:feature/Aberdeen/Temperature;
  ssn:hasValue [ qudt:numericValue "23.0"^^xsd:double ];
  ssn:observationSamplingTime "2013-01-01T20:15:00+00:00"^^xsd:dateTime
    ;
  .
```

Listing 3.1: Example RDF graph using the SSN ontology and QUDT ontology representative of the RDF graph generated by the implementation of the wireless sensor network

3.3.2. 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 the configuration file found in Appendix D. 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.

3.3.3. Linked Data Interface

The Pubby linked data frontend was deployed as an application in Apache Tomcat[8]. The configuration file used for Pubby can be found in Appendix E.

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. The configuration used for the virtual host can be found in Appendix F.

[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

3.3.4. 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.

The third test was to check the linked data frontend. The full example RDF graph given in listing 3.1 was loaded using a `LOAD` query. Using the listing and visualisation as a reference, the linked data interface was navigated and found to be working as expected.

All the tests completed as expected and no problems were found.

3.4. Hypertext Transfer Protocol Client

The Hypertext Transfer Protocol (HTTP), the protocol behind the World Wide Web, is currently used by many systems to provide a stateless API. These systems range from simply providing data to logging and even control. As a widely deployed protocol, HTTP is certainly a candidate for use by a wireless sensor networks for pushing data from the sensors to a central server.

3.4.1. An Analysis of HTTP Versions

Wireless sensor nodes are typically constrained in both the speed they can transmit at and the amount of time the radios can be active due to power availability. It is important that the sensor can send its update in as few bytes as possible and then shut down its radio to conserve its power.

All HTTP versions are plain-text based protocols, i.e. no compression is performed and only printable characters can be used.

HTTP version 0.9[1], still supported by modern web servers including the Apache HTTP server version 2.4.4, the latest version at the time of writing, has a simple request protocol which consists of opening a TCP[21] connection and sending a plain-text string in the form

	HTTP/0.9	HTTP/1.0	HTTP/1.1
GET	26	37	56
POST	-	121	140

Table 3.1.: Minimum request sizes for `GET` and `POST` requests for HTTP versions 0.9, 1.0 and 1.1

of "`GET /url`" followed by a carrige return and a line feed. Headers are not supported for requests and only the GET method is available. There are also no headers attached to a response and so a zero-length response would simply result in the connection being closed. Parameters such as a sensor ID and sensor reading can be attached to the URL in order to be processed by the server.

HTTP version 1.0[1], again still supported by modern web servers including the Apache HTTP server, has a more complex request protocol. Once the TCP stream is opened a plain text string in the form of "`METHOD /url HTTP/1.0`" followed by two sets of carrige return and line feeds must be sent. The version number added to the request informs the server that the newer version of the protocol is in use. As HTTP/1.0 supports headers, a second carrige return and line feed must be present to indicate the end of the headers section.

HTTP version 1.1[8], the currently most widely deployed version of HTTP, adds futher complexity by requiring the presence of a "Host" header in all requests. HTTP versions 1.0 and 1.1 also support the POST method, where the parameters are passed as a request body, which requires the additional "Content-Type" and "Content-Length" headers.

The minimum request size, assuming an 8 character sensor ID, a 5 character sensor value, an 11 character hostname and the update script being at the root of the server, for each protocol is shown in table 3.1.

One of the stated uses for the `POST` method within the HTTP/1.0 specification is "extending a database through an append operation" which is exactly what submitting a sensor result should do. `GET` requests are meant only for retrieving information, whether from static resources or from data generating processes, not for performing any updates or changes and so with version 1.0 and 1.1 of the HTTP specifications the use of a GET request to submit sensor readings would break the specifications.

Zero-length responses from the server with versions 1.0 and 1.1 would in fact have the server respond with an HTTP status code of "204 No Content", a number of headers, and a human-readable message as an HTML document explaining the status code. This size of this would be likely to be significantly larger than the request, although the page returned in this case by the server is configurable. As the connection is already using TCP, there is nothing gained from the reciept of a response from the server by the sensor node.

Following this analysis of the HTTP versions, it seems that newer is not better. As the protocol has evolved, it has become more and more complex and thus the minimum packet size containing the same data has grown leading to the radio being in use for longer periods of time. Due to this, for the HTTP version of the sensor node, HTTP/0.9 was utilised to submit sensor readings.

The SPDY/2[12] and SPDY/3[13] protocols, whilst in current use, have not been considered for the same reason that HTTP over SSL[10] and HTTP over TLS[7] have not been considered. The overhead of these protocols is greatly disproportionate to the amount of data being communicated by the sensor and their usage would greatly increase the amount of time that the radio would need to be powered on the wireless sensor nodes. Encryption, if necessary, should be implemented at the MAC layer where it can be performed by hardware as opposed to software with less overhead and reduced power consumption.

HTTP is a plain-text protocol and HTTP/0.9 provides no support for authentication. It is assumed that properly configured firewalls will be in place to allow only the wireless sensors to access the server to submit updates and that the sensors themselves are trusted.

HTTP can be wrapped in either SSL or TLS to provide encryption and versions 1.0 and 1.1 support authentication, but there is nothing gained from the sensors being Internet accessiable due to the fact that they spend most of their time asleep and so deployment on a private network should not bring any disadvantages.

It should be noted that as sensors are required to identify themselves, a rouge sensor could send false readings for other sensors acting as the other sensor. Any password-style parameter also passed with the URL could be easily identified by a sensor by “sniffing the wire” and so would not provide any enhanced security. It would also be possible for a rouge sensor to launch a denial-of-service attack but this is an extremely hard threat to mitigate as all that would be necessary is to produce noise on the frequencies used by the radio to push the signal-to-noise ratio beyond its tolerance.

3.4.2. Design and Implementation Detail

As the property chosen to measure is ambient temperature, sensing continuously and transmitting updates continuously would not provide any benifits but would increase the amount of data that would need to be stored and processed and would greatly decrease the life of the sensor by placing considerable drain on the battery power available. As such, the wireless sensor node should have a timer to perform a sensor reading and transmit a result periodically.

The Zigduino client is based upon a snapshot[12] of the Contiki operating system port to the Zigduino by GitHub user *quikshot*.

Contiki is an event driven platform and supports lightweight threads. This allows for a timer to be implemented in a thread triggering an event when it expires to perform the sensing and submission. The way in which this is implemented in Contiki allows for the timer to trigger an interrupt, allowing for the processor to sleep until the timer

[12]`https://github.com/quikshot/contiki-avr-zigduino/commit/a0c8d1311562f7c3cf879131ec8d9f514d2179de`

expires in order to futher conserve battery life. The alternative option would have been to constantly poll the timer which would require the processor to run at full load for the whole time it is waiting which would have resulted in limited run time due to the limited battery power available being consumed more quickly.

Upon activation, the sensor would construct an HTTP request placing the data objects as URI parameters as this is the only method available when using HTTP/0.9, open a TCP connection and send the request. Once the server has recieved the request it will close the connection. Reading the temperature sensor is simply a matter of including the header file for the sensor that maps a struct to the memory address of the sensor. By performing a peek operation, the value can be read.

There are two events to which the HTTP client needs to respond to, the reciept of a packet via the radio and the expiration of a timer indicating the next reading should be sent. Whilst no actual data is being sent to the client it is still necessary to process incoming packets as part of maintaining the TCP connections used by the HTTP client. The existing web client in Contiki was used with modifications to add the reading of the temperature sensor and the update by timer whereas previously the client would simply fetch a file and quit.

The observation data would be recieved by a CGI script via the web server that would perform SPARQL/Update queries to the triplestore. PHP contained all the necessary functionality to perform the task of inserting the data and is widely deployed as a CGI system. The use of PHP as a CGI script meant that the handling of the HTTP protocol could be left to an existing web server product, in this case Apache HTTP Server 2.

A PHP script read in the values sent by the sensor node and constructed SPARQL/Update queries to commit the observations to the triplestore using the SSN and QUDT ontologies. Other than the *ssn:madeObservation* property, no properties were added for the sensors and if this system were to be used in a real deployment, it would be expected that the triplestore was bootstrapped with triples relating to descriptions of the sensors and their capabilities.

3.4.3. Testing

The IEEE 802.15.4 MAC layer and 6LoWPAN stack requires a number of conversions before the IPv6 packets can be relayed across networks using other stacks, such as the traditional IPv6 over Ethernet stack. To ensure that packets were reaching the server correctly, Wireshark was used along with dissectors for the protocols in the stack to ensure that the packets were correctly formatted and that their checksums were correct.

The Apache HTTP server logs proved useful in testing that, over an extended period of time, the timer was regularly triggering with negligable skew in the time between triggers and calling the update script on the server. Using the logs and cross-referencing them with the Pubby linked data interface it was possible to see that all the observations were being created in the RDF graph without errors and with all the expected annotations attached.

3.5. Constrained Access Protocol Client

Constrained Application Protocol (CoAP) is a software protocol intended to be used in very simple electronics devices that allows them to communicate interactively over the Internet. It is particularly targeted for small low power sensors, switches, valves and similar components that need to be controlled or supervised remotely, through standard Internet networks[23]. CoAP does not define the exact functionality required and as such it was necessary to define a protocol on top of CoAP to allow for the reporting of observations from the sensor node to the server for storage.

3.5.1. Benefits of CoAP over HTTP

As CoAP is explicitly designed to be used in scenarios similar to the one encountered in this application, there should be benefits of using it over using a more general purpose protocol like HTTP. The first problem it attempts to tackle is removing the overhead and complexity of TCP by operating entirely over UDP and implementing an optional acknowledgements system. This allows for messages to be transmitted with only best-effort and in cases of network congestion, a sensor reading will be lost instead of adding retransmission attempts into the already congested network. For most sensor network applications not related to security, eventual success is good enough.

The second advantage to CoAP over HTTP is the smaller amount of data to be transmitted. TCP requires a three-way handshake before any communication can even begin however this is not necessary with CoAP as data can be sent on the first packet. In this application, the client is not interested in the data contained within the replies and so also does not need to store any state about the connections and can simply discard incoming packets - excluding ICMP echo requests for testing purposes.

The amount of data to be transmitted is futher reduced by using a binary header as opposed to the ASCII header used by HTTP. Far more information can be transmitted with the radio being powered for a shorter time frame. The third advantage to CoAP over HTTP is that it can be used with multicast which allows the sensor nodes to send their updates to a multicast group as opposed to a single server. This was utilised to allow for auto-discovery of sensor nodes without the need for prior configuration.

Like HTTP, there is no encryption being used when passing data from the sensor nodes to the server, although it is possible to use Datagram TLS (DTLS) with CoAP it is beyond the scope of this project to implement it. No authentication is performed and so any rouge device could connect to the network and submit sensor readings. Any authentication used would need to be able to withstand replay attacks.

3.5.2. Design and Implementation Detail

The design of the overall system is very similar to that of the HTTP-based system but instead using CoAP as the communication protocol. Following private communication with one of the designers of the CoAP protocol, Carsten Bormann, it was decided that a `POST` request should be used to submit a representation of the resource to the server for it to be created. The semantics of the CoAP request methods are not defined in any more detail than with HTTP and so there is no definitive correct way to request a CoAP server to create a new resource, especially a semantic web resource, but the use of `POST` appears to be best current practice.

Although multiple resources would be created by the server from the single submission, it was decided that the payload of the `POST` request should be as small as possible and so, again, it only contained a sensor ID and a sensor output. With no real-time clock on board the Zigduino, the observation sampling time was left for the server to determine. If an HTTP post were used, it would have used the `application/x-form-url-encoded` which is not available for CoAP however there is a content-type of `application/json` in the CoAP content-type registry and so this was used instead to encode the representation of the sensor result.

Originally, it was intended that the CoAP packet would be represented in the code as a struct, with a linked list of options, with each option containing a pointer to its value. This would allow the easiest construction of CoAP packets and allow for header fields, options and payload to be added in any order. A problem was encountered when attempting to implement this in that there is a lack of built-in memory management for the Contiki Operating System.

Functions like `malloc()` that would be encountered in standard C libraries are not present and the length of many fields in the CoAP packet structure are not fixed. To work around this problem, the CoAP packet builder implemented required the use of a series of functions in a specific order to correctly build the packet structure inside a statically assigned `char` array. This meant that the construction of a packet had to follow strict rules and reduced the amount of flexibility in the code.

On the server side, using Java, the packet could be easily unpacked and the fields made available through getter methods on an object representing the packet.

3.5.3. Testing

Due to time constraints, the multicast CoAP system was not implemented in its entirety and so no integration testing could be performed. The packet builder on the Zigduino client was tested by manually checking hexdumps of automatically constructed packets against hexdumps of manually created packets in order to ensure it was working as expected. It was seen that packets were correctly built when the functions were called in the correct order, but the consequences of failing to follow the order were the creation of corrupted packets that at best would be ignored by a server and at worst would crash it.

Had the implementation been completed, state tracking would have been implemented to return an error when a function was called out of order.

4. Integrated Sensing

The main aim of this project was to collect and integrate data from both human and machine sources. After the design and implementation of the wireless sensing architecture, the way in which sensors interact with the logger, the level of information available and how that information could be stored using RDF and the SSN ontology was well understood. Using this knowledge an example application was devised to collect data from humans and machines and represent it in RDF in a way as compatible as possible with the current primary use case of the SSN ontology.

4.1. The uktemp.net Application

Whilst it was important to ensure that any solution remained as general as possible to accomodate the largest amount of use cases for integrated human and machine sensing, it would not have been possible to produce an application that would use every source possible for every possible property of any feature of interest. Temperature is a property that can be measured by both humans and machines

4.2. Architecture Overview

The system was once again based on the Apache Fuseki triplestore and the Pubby linked data frontend. Pubby was deployed in a servlet container and so it made sense that any modules written for this application would also be deployed in the same servlet container. The first two modules considered were the data acquisition modules. These run as background threads in the web application to acquire data from each source. Once some data was acquired, the third module considered was the presentation module that provided a web frontend to the data collected.

The architecture has been designed in such a way that whilst the modules can be developed as part of the same Java web application, a low-level of coupling was aimed for between modules with each module performing a specific task and having communication only with the triplestore. This allows for any number of sources to be used for the acquisition of data and any number of presentation modules to be used to perform reasoning on the data and display it to an end-user, or another machine system.

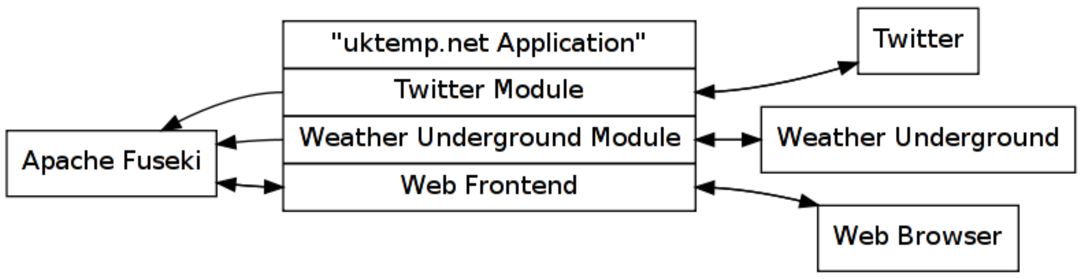

Figure 4.1.: An overview of the uktemp.net application architecture

4.3. Ontology Development

For capturing machine generated data, the existing SSN ontology and QUDT ontologies are sufficient to represent all the data that was accquired from the Weather Underground data acquisition module. Problems were encountered when it came to describing human sensing however. This was not due to the lack of data available from the Twitter API but due to the differences between machines and humans.

The first problem encountered was not even considered when building the wireless sensor network, what is the sensor? For the machine sensing, common sense indicates that the sensor node is the sensor. It produces observations based on a sensor result that was in turn based on a stimulus. In humans, temperature sensation from thermoreceptors enters the spinal cord along the axons of Lissauer's tract that synapse on second order neurons in grey matter of the dorsal horn. The axons of these second order neurons then decussate, joining the spinothalamic tract as they ascend to neurons in the ventral posterolateral nucleus of the thalamus.

The SSN ontology is extremely flexible. This is good as it allows for adoption in the greatest amount of use cases but also bad as it allows for data representation to become very complicated very quickly. If we used the same logic as was applied to the machine sensing system, it is the thermoreceptors that are the sensors as they are producing the sensor results in response to the stimulus. This is clearly impractical given the number of thermoreceptors a human has for the amount of data generated would be beyond the ability for the human to report.

The next idea considered was to define the sensor as the skin, which would contain all the thermoreceptors. Still there is no real measurable sensor output. With any human sensing task, the senses are processed by the brain. These cognitive processes use memory and reasoning along with inputs from other senses in order to reach conclusions that can then be reported. Humans are typically completely unaware of the processesing that happens subconciously and will not have access to the intermediate results.

The only remaining logical choice is to make the human as a whole the sensor, incorporating both the senses and any congnitive functions that occur in the process of reaching the sensor result. This led to the development of the Human Sensor Ontology which extends the Semantic Sensor Network Ontology in order to provide a vocabulary for representing "human sensors". The vocabulary provides classes that can be extended per-task to create task-bounded human sensing devices and task-bounded human sensing processes.

Humans are capable of sensing a wide variety of properties of their environment. The ontology provides a basis, to be extended on a per-project basis, for integrating human sensing with the Semantic Sensor Network Ontology. Instead of declaring the sensing device to be a human, an eye, an ear or a hand, classes are provided for human sensors that are performing a specific sensing task.

In the majority of cases where human sensing is being used, a task will likely have been selected beforehand or, when working with historical data, the participants will likely be selected because they performed the task at the time. In a similar way, sensing processes are defined for each high-level sensing task. The full documentation for this ontology can be found in appendix G.

It was noted that the human sensing devices also fulfilled all the criteria for being a `foaf:Person` allowing annotation using FOAF properties such as name, age, gender, etc.

The next problem encountered was with regard to the values that would be reported by a human. In the case of recieving values from humans, they may be able to estimate a value for a measurement but in many cases would be more comfortable giving a value in a qualitative format. Unless dealing with measurements of a property regularly, most humans would not have great accuracy when estimating. Humans can draw on experience from other areas and give qualitative values instead, possibly with greater consistency.

Unit classes from QUDT were subclassed, where appropriate, and the new classes are all subclasses of `qunits:QualitativeUnit`. For any value, `qudt:unit` should point to an instance of one of the subclasses of `qunits:QualitativeUnit` and `qunits:qualitativeValue` should point to a string literal containing an objective (as possible) qualitative description of the property.

In the case of returning values to humans, the values may be more meaningful if they are given in a qualitative form as opposed to the quantitative form as they become easier to imagine. Performing reasoning on the quantitative values to produce qualitative values may be a useful step for the process of the generation of natural language reports on data. An example of the use of this ontology is shown in figure 4.2 and the full documentation for this ontology can be found in appendix H.

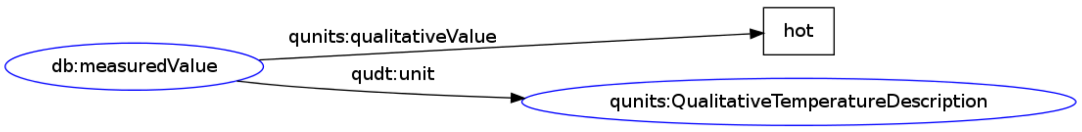

Figure 4.2.: An example of the use of the Qualitative Units Ontology

Finally, a simple ontology was developed for the application itself, defining classes for the Twitter based sensor, the Weather Underground based sensor and postcode districts as a feature of interest. The full documentation for this ontology can be found in appendix I.

4.4. Data Acquisition Modules Design

The data acquisition modules will implement source specific methods for the acquisition of data, parsing the data and then building a SPARQL/Update query to add the data to the triplestore. Whilst in the cases of the two chosen data acquisition methods the first step is performed by calling a web-based API, this is not the only way in which data could be accquired. Any data sources available to Java could be used in the first step of this process, such as keyboard input or pre-collected data in the filesystem.

4.4.1. Twitter

Twitter provides an advanced streaming API that allows for tweets matching the specified criteria to be pushed to a listening application. A Java implementation of a client for this API, twitter4j[1], was used to communicate with the API. The format for tweets was based on the popular UK Snow Map[2] which uses user subitted tweets to map snowfall across the UK although not using any machine produced data or semantic web technologies.

Tweets directed towards the uktemp.net application would have the hashtag #uktemp and this would be the search criteria for the Twitter stream. The important information that must be available is the observation itself and the location for which the observation is about. As tweets are free-form text and users may send any string they wish, it was important to seperate the observation from the rest of the tweet.

Initially, it was considered that simply searching for a list of set words, such as hot, cold or mild, would suffice but it was quickly realised that without context, the meaning of these words can be easily misunderstood. Simply being prefixed with a negation would make the parser's result incorrect and regional variances in dialect may produce words that are not on the set word list. For this reason, it was decided that the observation itself should be surrounded by quote marks allowing the user to produce a meaningful tweet that is also useful to the application.

For the location information, it was requested that users would enter their postcode district in the tweet, which could be easily identified by a regular expression. In the event that the location could not be identified from the tweet, a fallback mechanism could be used to extract geolocation information from the tweet, which is provided when a tweet is pushed to the application if it is available, and then using the Geonames[3] web service translate the geolocation information into a postcode district.

It was allowed that the location information and the observation could be in any order. Due to the implementation, if there were multiple locations or observations matched the first mentioned would be used and the others ignored. A typical tweet would look like:

It's "cold" in AB24 #uktemp

[1]http://twitter4j.org/
[2]http://uksnowmap.com/
[3]http://www.geonames.org/

At this point, the application has enough information to push the data to the triplestore. Problems such as the trustworthyness of the data are not the concern of this module as all observations should be placed into the triplestore and reasoning should be performed by either a seperate reasoning module creating new resources in the triplestore or as part of a presentation module.

4.4.2. Weather Underground

Whilst it would have been nice to deploy the wireless sensing architecture across the UK for sensing temperature and other weather characteristics, it was beyond the budget for this project. As a source of machine generated data, Weather Underground was chosen. Weather Underground aggregates data from weather stations worldwide and published them on their website and via a RESTful API returning XML.

The initial problem with the API was that the search feature required a place name and would not work simply on a postcode district. Once again, the Geonames web service was used to translate postcode districts into a nearby place name. Weather Underground does not have a resolution capable of giving per-postcode district observations and so one weather station would typically be authoritative for multiple postcode districts.

The second problem with the API was the free usage limits. It was certainly not going to be possible to poll for every postcode district in the UK and so the module would only poll for locations that were already present in the triplestore. This allowed for fewer requests to the API whilst still accquiring a useful level of data that can be integrated with the human observations. The number of API requests was then futher reduced by polling each location name only once per cycle instead of every postcode district where the same station was authoritative for multiple postcode districts.

4.5. Web Frontend Design

The web frontend was an important part of the system. It did not contribute to data acquisition directly, but did provide instant feedback to humans that submitted observations so they did not feel that they were blindly tweeting at a system and getting nothing in return. The goal for the web frontend was to demonstrate to users in a simple and easy to understand way why the data was being collected and how it was being used.

As the data collected had a geospatial element it made sense to represent the data on a map. The map contained markers for each location for which there had been observations in the last 6 hours. When a marker was selected, the observation data would be displayed. When creating a frontend to a semantic web application it is easy to lose the semantics and hide them in the background. It was important that this was avoided.

When a marker was selected it would display a list of human observations, who tweeted them, when they were tweeted and the closest machine observation. This way it was easily apparent to a user when a tweet may no longer be accurate due to being too old or if a user is consistently disagreeing with the other users, they can be identified as fraudulent instead of counting all tweets equally and simply choosing the majority opinion.

4.6. Implementation Detail

4.6.1. Base Architecture

The whole system was to be self-contained within a Java web application to be deployed in a servlet container. Whilst the entire Java language, standard libraries and 3rd party libraries were still available for use there were certain factors that had to be considered in this environment where it differed from a standard application.

There is no main method called when a webserver is started and so the data acquisition modules must be initalised in an alternative way. The data acquisition modules are initialised by a servlet that contains the load-on-startup tag in the web.xml file. The load-on-startup tells the servlet container to load the specified resource and call the `init()` method at server startup. The `init()` method is blocking and so threads are used in order to run the main data acquisition module code.

Had the ability to launch threads using such a method not been available it would have been necessary to seperate the system into at least two seperate applications which would have fractured the codebase and introduced more complexity in the deployment process.

4.6.2. Data Acquisition Modules

Twitter

The twitter4j streaming API contains a listener thread implementation and so this was used. The alternative would have been to poll Twitter's search API for tweets which can result in tweets being missed or being delayed. It was also not necessary to check if a tweet had already been processed as the streaming API pushed each tweet to the client only once.

Weather Underground

For the Weather Underground module it was necessary to launch a timer that would call the polling function at a set interval as no streaming API was available. This meant a careful calculation to ensure that the API limits were not exceeded by polling for too many locations too often. The caching mechanisms discussed in the design were implemented but this proved to not be enough and so the length of time between polls was extended to half an hour from the original five minutes.

In order to fetch the maximum possible data, the number of locations to poll for could have dynamically changed the time between polling but for ambient temperature, readings every half hour are sufficient to identify any changes and so this was not implemented.

4.6.3. Web Frontend

The web frontend required a dynamic map and so the gmap3 plugin to jQuery was used to provide that functionality. There is no fuzzy logic available in SPARQL and so selecting the closest machine observation to the human observation for display on the map required the use of a programatically built second query for each row of the query results with the human observations. This increased the page load time considerably. The results of all queries were combined into HTML snippets and attached to markers with the location given as the postcode district. This was all then encoded as JSON and then embedded into the JavaScript of the page.

The queries were all performed against the SPARQL endpoint which is public and is available to clients on the Web. Faster page load times could probably have been experienced if the markers were loaded by the page itself using JavaScript once the core of the page had rendered. This would also have had the advantage of allowing the entire module to be contained within a single static HTML file. This approach was not taken as it was not expected that the page load times would have been affected to the extent that they were and knowledge of the Java APIs for SPARQL were readily available where knowledge for the JavaScript APIs were not.

4.7. Testing

As the system was to run accepting any input that was given to it from Twitter, the tweet parser recieved heavy testing. Tweets containing nonsense, only location, only observation, observation and geolocation data and observation and location in multiple orders and with and without special characters were tried. The parser accepted any tweet with enough information and ignored all others without producing any errors that would have prevented the system from continuing to process new observations.

The results of the Weather Undeground module were verified via the Pubby linked data interface and proved to be working correctly. In the event of the data returned from Weather Underground not being XML or not having the expected schema, it failed gracefully without throwing an exception allowing the application to continue to run.

The system was deployed on an Internet facing server for an extended time and advertised to students of the University and via Reddit[4] and did not require any human intervention in its operation for the time it was online.

[4]http://www.reddit.com/

4.8. Results

Having run the application facing the public Internet over the period of 10 days, over 71000 triples were generated and stored in the database. This included 5548 observations with 30 from Twitter users and 5518 observations from Weather Underground. Almost all the human observations were for the postcode district of AB24 with only 7 observations for other areas or unrecognised areas.

Simple analysis of the results could be performed using R, a statistics programming language. The rrdf library allows SPARQL queries to be performed on a triplestore with the results presented as native R datatypes. The first analysis to be performed was simply to plot the machine observations of temperature in AB24 over the 10 day period.

This was done by performing a SPARQL query against the uktemp.net triplestore for all observations made by the Weather Underground "sensors" observing AB24 and fetching their value and sampling time. This query is shown in listing 4.1.

R treated both the temperature values and the sampling times as strings and so some operations needed to be performed on the data before any plotting could be performed. The temperature values were forced to be numerical data and the sampling times were converted to the native R time format. The plot of these results can be found in figure 4.3.

```
PREFIX qudt: <http://qudt.org/1.1/schema/qudt#>
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ssn:  <http://purl.oclc.org/NET/ssnx/ssn#>
PREFIX unit: <http://qudt.org/vocab/unit#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT ?temperature ?time WHERE {
    ?sensor rdf:type <http://uktemp.net/ont/#
       WundergroundTemperatureSensingDevice> .
    ?observation ssn:observedProperty <http://uktemp.net/dataset/
       temperature/AB24> .
    ?observation ssn:hasValue ?value .
    ?observation ssn:observationSamplingTime ?time .
    ?value qudt:numericValue ?temperature .
    ?value qudt:unit unit:DegreeCelcius .
}
```

Listing 4.1: SPARQL query used to select temperatures and sampling times of observations of outdoor temperature in AB24 by Weather Underground

From the plot, the rise and fall of temperatures with the day/night cycle is clearly seen. There are also parts of the chart after the first few days where it is clear there are data points missing. This is due to the API limits for Weather Underground as more locations were known about the longer the trial ran and this increased the number of API calls.

With the relatively small number of observations from Twitter, it was not possible to gain any statistically significant results. Analysis of the integrated data was still attempted

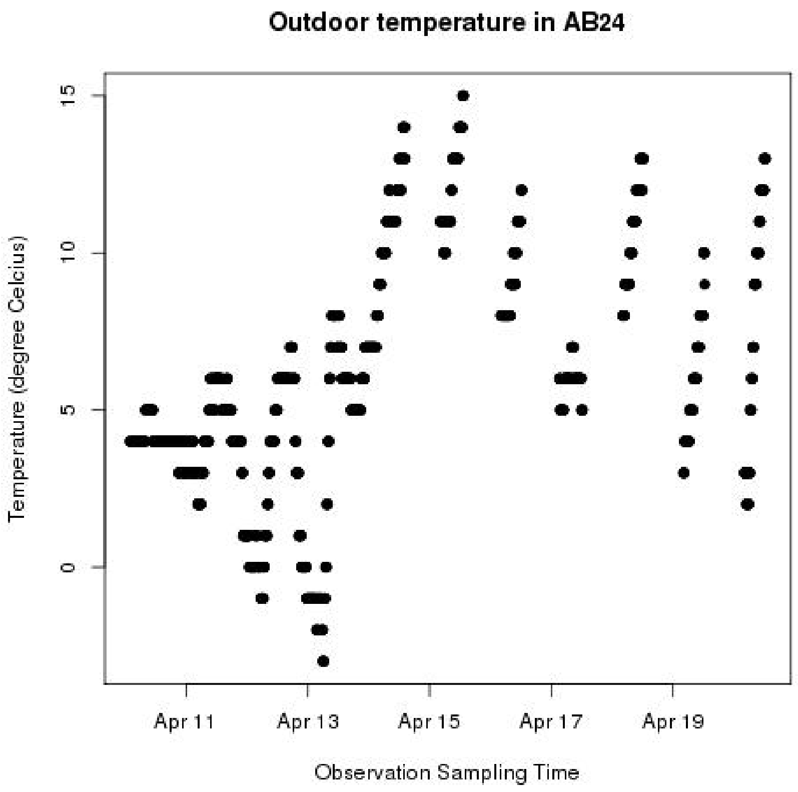

Figure 4.3.: The temperature as reported by Weather Underground in AB24 over the 10 day trial of the uktemp.net application

however to see what results could potentially have been drawn had the response been better.

The first analysis attempted with the integrated data was to see how widespread the vocabulary was for a single region, AB24. If the vocabulary was small, with words often being reused, it would indicate that futher analysis would be more worthwhile as more data for each word would be available. A frequency bar chart was plotted, shown in figure 4.4, showing the frequency of use of each word. It can be seen that the vocabulary only consisted of 5 words with words used on average 2.4 times.

Given the small vocabulary and the fact that words were being reused, it was possible to plot boxplots for the machine temperatures recorded close to observations containing each word. These boxplots are shown in figure 4.5. From the boxplots, it seems that chilly and cold are being used for the same range of temperatures, but freezing, mild and hot all are used to describe distinct ranges of their own. If similar results were achieved from a larger set of observations, it would be possible to create a mapping of qualitative to quantitative values from such analysis although these results may have occured by chance due to the low response rate.

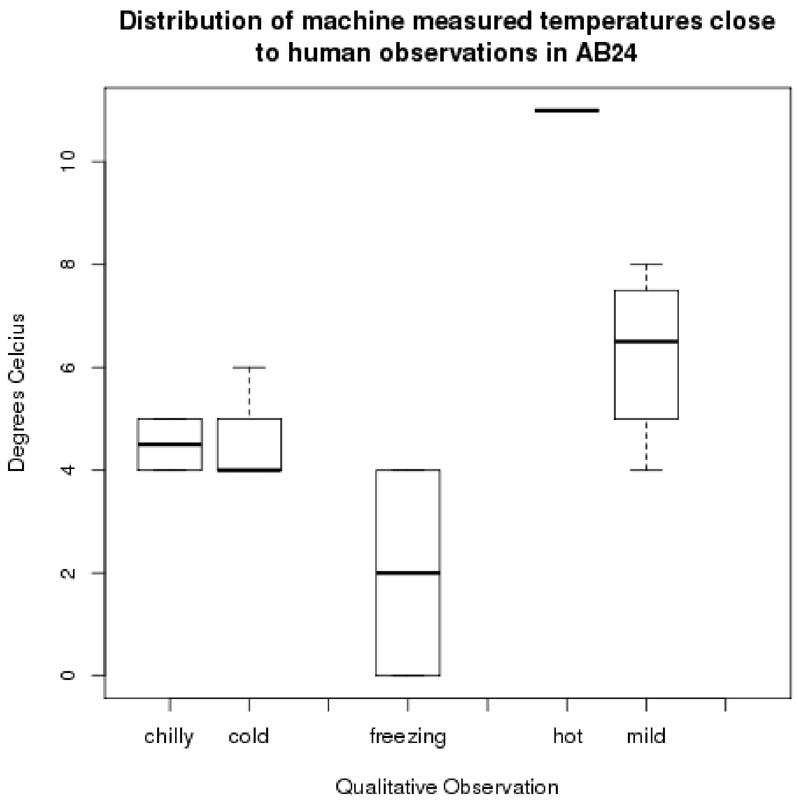

Figure 4.4.: Bar chart showing word frequencies for human qualitative observations of temperature in AB24

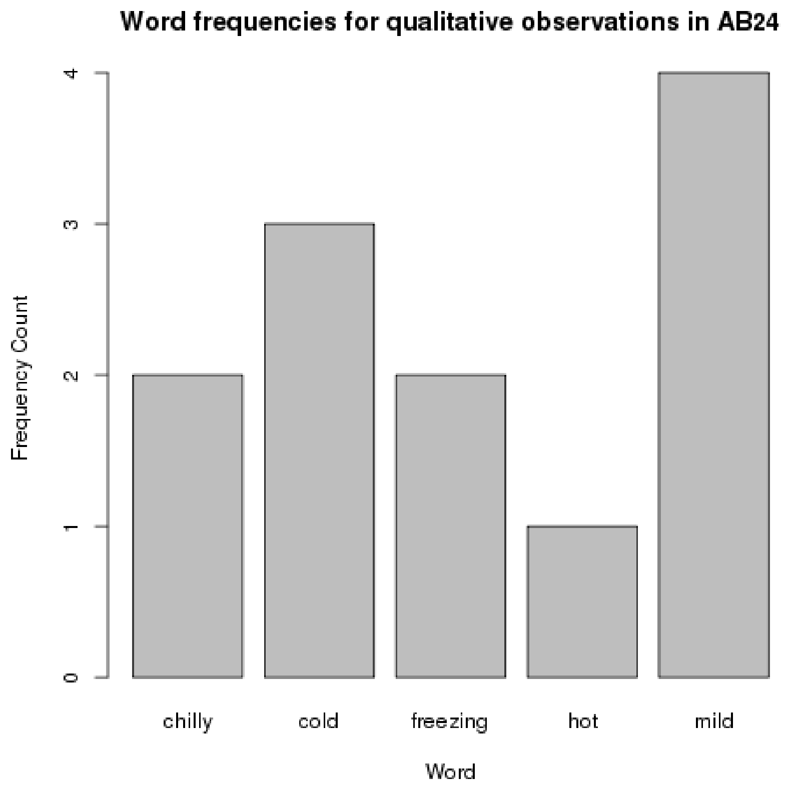

Figure 4.5.: Box plots showing the distribution of machine measured temperatures close to qualitative observations

5. Evaluation

Throughout the project, technologies were chosen for use based on the benefits that they were seen to bring. In this chapter, those choices were evaluated for their suitability in the wireless sensor network environment, for human sensing and for the ease with which results could be integrated.

5.1. HTTP and CoAP for Wireless Sensor Platforms

Whether using a Zigduino or another sensor node platform there are always going to be a series of constraints placed upon the device. The devices typically do not contain a large amount of memory, they have low-bitrate radios and a limited amount of battery power that must be conserved by only using the radio when necessary. The choice of communication stack must take into consideration these constraints.

The Internet Protocol is ubiquitous across Internet connected networks with the next generation of Internet networks using version 6 of this protocol. The ability to allow for communication across networks using existing infrastructure clearly brings many benefits, such as the ability to have the logging server located anywhere on the Internet and not necessarily on site where there may not be available power for anything more than a router and a backhaul link. The problems involved in carrying IPv6[6] over low-power low-bitrate networks has already been approached by the 6lowpan working group of the IETF in the Internet Area[16]. They produced the 6lowpan protocol[19], encapsulating IPv6, which provides header compression and the necessary work to allow it to be carried over the IEEE 802.15.4 MAC layer[14] used by most modern low-power low-bitrate radios.

On top of 6lowpan, at layer 4 of the OSI stack, both UDP[20] and TCP[21] can be easily used by sensor nodes and an API for this is already provided by Contiki. HTTP must use TCP but CoAP is capable of using both TCP and UDP. Only a UDP implementation of CoAP was attempted as the only advantage bought by TCP is offloading the reliability features to the TCP implementation whilst introducing the need to track state and perform a 3-way handshake to establish a connection for every submission. The reliability features of CoAP were deemed to be unnecessary and so a more complicated reliability system would also be unnecessary and just introduce avoidable overhead.

From the previous analysis of HTTP[8] minimum packet sizes as HTTP has evolved in section 3.4.1, it is clear that it is becoming less and less suitable for use in the constrained environments that are found in wireless sensor nodes. It was possible to implement an older version of the protocol in order to submit the observations with smaller payloads on the network packets but this is not a long-term solution as it is likely that at some point, support for the older version of the protocol would be removed from HTTP servers. By using a protocol that is no longer being supported would introduce the same problems as the use of properietary standards: lack of available libraries, lack of documentation and support and lack of interoperability.

CoAP[23] is still in active development by the CoRE working group of the IETF in the Applications Area[15] although there have only been slight changes between the last two drafts (13 and 14). CoAP is designed to emulate the REST features of HTTP but in a way that is more constrained environment friendly. The advantages it brings over HTTP are binary headers thus reducing the packet size and using UDP instead of TCP replacing the 3-way handshake and reliability features designed for use over faster networks that may not work as well over the constrained wireless sensor networks.

As CoAP is using UDP, it can be used for both unicast and multicast requests so that a sensor may update multiple endpoints with a single message. This, again, could reduce the amount of time the radio is on. Whilst CoAP servers are not currently widely deployed, a stateless HTTP mapping exists allowing proxies to be built providing access to CoAP resources via HTTP in a uniform way and allowing for HTTP simple interfaces to be realized alternatively over CoAP.

The problem that exists with both HTTP and CoAP are that they are based on the request-response model. The wireless sensor node doesn't require a response and in the implementations, completely ignores them. With HTTP, the wireless sensor must receive the packet containing the response before the TCP connection can be closed and its associated state discarded. This is actually a problem associated with TCP, but as HTTP requires TCP to function it also inherits it.

With CoAP, over UDP, it is possible to disable the reliability features and not store any state about connections. This means that any packets received can simply be discarded and nothing is lost if the radio is disabled immediately after sending the request. Still, it would be considered bad practice to send data across a network when there was never any intention for it to be received by the node it was addressed to.

Given that CoAP allows for these non-confirmable messages, a server could be configured to not send a response to the request. This would not break the CoAP specification as, for a non-confirmable request, no response is guaranteed. This would however require the non-standard configuration of a server which could break its interoperability with other platforms also depending on it.

The HTTP implementation used version 0.9 of the HTTP specification which only contained the single method: `GET`. This did not have any semantic meaning to it at all. For the CoAP implementation, the `POST` method was used but the semantics of this method are not clearly defined beyond the fact that it is submitting data to the server. This does not foster the creation of standards or best practices for the task of submitting semantic web resources to a server.

When submitting a sensor observation, resources are created within the servers namespace and populated with data either explicitly given by the sensor or inferred from the data submitted. When dealing with semantic web resources, methods such as `CREATE`, `READ`, `UPDATE` and `DELETE` would carry a lot more meaning than just combining all these operations into the single method of `POST`. The use of the methods `GET`, `POST`, `PUT` and `DELETE` are commonly used to manipulate filesystem resources over HTTP with the WebDAV extension[11] but no effort has been made to extend this to semantic web resources. The most important difference that would need consideration is that the `PUT` function when dealing with filesystem resources allows the user to specify the new URI for the resource but when submitting semantic web resources, it may be necessary for the server to determine the new URI to avoid pollution of the namespace.

Observations, in the CoAP version, were submitted as JSON[5] payloads to the server. Compared to other encodings, such as XML, JSON is a very compact encoding which again minimises any overhead in packet transmission. The one transmission created multiple resources and the majority of the data for each observation was inferred by the server instead of explicitly given. If the system were to have dealt with the fact that it was creating multiple resources, it is possible that JSON-LD could have been used instead of the simple JSON to submit the observations. Still, without futher semantics being added to CoAP or standards on dealing with semantic web resources over CoAP, it is doubtful that this would ever allow for increased interoperability.

The CoAP stack is clearly a better solution than the HTTP stack for wireless sensor networks but it seems that there are still places where it has shortcomings. These shortcomings are mainly due to the fact that it aims to maintain interoperability with HTTP systems and so inherits the same models. Where power availability is not constrained, CoAP would be useful for tasks such as pushing configuration updates to nodes and recieving confirmations of completion using a multicast request or having a smart lightswitch turn off all the lights in the house but for simply submitting sensor observations there is room for a more lightweight protocol.

5.2. Suitability of Semantic Web Technologies

Semantic Web technologies played a large part in the entirety of this project. The three core Semantic Web elements were RDF, the SSN ontology from W3C and SPARQL. Using the RDF graph data model and the SSN ontology to model the wireless sensors, their observations and their values presented no issues and beyond the use of the QUDT ontology in representing values, the SSN ontology was sufficient to provide all the necessary vocabulary. In a real application, it would have been necessary to extend the SSN ontology to provide classes for specific sensor platforms, but that was beyond the scope of the wireless sensor network part of this project.

SPARQL/Update provided a clean and simple way of submitting new data to the RDF graph without the complicated setup of drivers and connections that is normally necessary when communicating with a RDBMS. All that was necessary was a HTTP POST request to be made with the new triples to be inserted.

When it came to analysis of the data however, it quickly became clear that the use of Semantic Web technologies offered no advantages over a RDBMS with SQL. To extract the data from the graph a SPARQL query was used which returned a table of results, the same result as would have been achieved using an SQL query on a RDBMS. It should be noted that the performance of RDF triplestores and the processing of SPARQL queries is nowhere near in line with the performance of making a similar query using a RDBMS and so with large amounts of data, this could become a serious bottleneck.

The advantage that does come with the use of semantic web technologies is that the data becomes far easier for others to use. The schemas for relational databases storing sensor observations are likely to vary wildly with application requirements and the information available but the vocabulary for storing sensor observations in an RDF graph has been standardised by the SSN ontology. As SPARQL has an HTTP binding, unlike SQL for RDBMSs, it can be made public easily allowing for others to use the data. A library for R, rrdflib[1], exists, a wrapper around Apache Jena, to fetch data from SPARQL endpoints and make them available as R datatypes. This could allow for data analysis of sensor observations from multiple different deployments with ease.

In the second half of the project, when it came to representing human sensors, the SSN ontology vocabulary was considerably lacking. The SSN ontology has each sensor attached to a platform which did not make any sense for humans and the idea that humans could belong to a "deployment" only works if the universe was intelligently designed. The Human Sensing Ontology was created by extending the SSN ontology to introduce a vocabulary for describing abstract human sensors but this workaround mainly just allowed the representation of the bare minimum with regard to sensors, i.e. that it is a sensor observing a property of a feature of interest and produces sensor results through an abstract sensing process, in order to allow the recording of observations.

When it came to representing qualitative values, the QUDT ontology contained no support.The Qualitative Units ontology was created with the concepts of units and values extended from QUDT to build a framework for representing qualitative values of different properties. Integrating the Qualitative Units ontology back into SSN however worked well, allowing for detailed data about observations to be stored at the same level as that which is stored for machine generated observations, i.e. the value and a unit representing the property that the value describes.

RDF and SPARQL/Update presented no issues with the storing of sensor data and observation data, but the SSN ontology is machine sensing focused and not well suited to representing human sensors with its vocabulary. For some of these challenges, the workaround used limits the potential to store more detailed information about the sensors.

In the map generated from the combined machine and human observations, the closest

[1]http://cran.r-project.org/web/packages/rrdf/

machine observation, if there is one within a sensible time period, is given as a reference for each human observation. To fetch a view of the data like this from an SQL database would require only a single query with a nested SELECT, but the date and time functions in the Apache Fuseki SPARQL processor are not as mature as its RDBMS counterparts and so it had to be split into two queries, one to get the list of human observations and then another, run for every row of the human observations, to fetch the closest machine observation with parts of the second query built programatically.

Temporal reasoning would be important when dealing with sensor observations as observation sampling time would often be used in any data analysis for environmental sensing. This represents a serious shortcoming in current SPARQL processors but the situation will likely improve over time. For reasoning with qualitative values, basic string manipulation is available via XPath functions in Apache Fuseki although this was not used and so cannot be evaluated by this project.

5.3. Crowd-sourcing from Twitter

Twitter is the most popular microblogging platform in the UK with Twitter clients integrated into Mac OS X, iOS and Android devices to name a few. For this reason, it was chosen as the method of receiving the observation values from humans. Natural language processing was beyond the scope of the project and so only simple regular expression parsing was used to extract the data which required a fixed format for the tweet. As a way of identifying categories for a tweet, Twitter users have adopted the hashtag [9] which basically consists of a word prefixed with a hash (#). The hashtag of #uktemp was used to identify tweets for the application.

Collecting tweets via the streaming Twitter API worked extremely well with the hashtag with tweets arriving for the application to process almost instantly after posting and with no tweets missed. Unfortunately, due to the lack of response, it is not clear if this will scale and continue to perform as well. It is likely that the hashtags on tweets are indexed to improve performance for the Twitter search APIs where keywords may not be.

If Twitter had not been used, and instead a private platform with separate user accounts had been constructed, this would have presented a barrier to users and the response count would have likely been lower. By using Twitter, users would use their existing accounts and would not even have to authorise the uktemp.net application as it would be able to capture tweets from any client that the user chose to use.

Whilst the system was modelled on the popular "UK Snow Map" which uses similar tweets to build a live map of snow across the UK, the low response rate was likely due to the fact that people simply do not find the temperature outside to be interesting whereas snow is rare in the UK and so something that users may tweet about anyway. Getting a user to modify a tweet to be parseable when the tweet was going to be sent anyway is a lot easier than getting users to tweet something that they wouldn't normally have considered tweeting.

It was considered that instead of having tweets specifically aimed at the application, that all tweets for a geographic area would be considered and then some heuristics used to determine whether or not it was a comment on the temperature and what that comment was. This would have led to a large number of natural language problems however and so was never attempted.

As with most crowd-sourcing systems, if the user isn't interested then the user won't submit any data. Technically, everything was there to collect observations from Twitter but without the users submitting information, the system is worth nothing. There needs to be some form of incentive for the user to participate. As a result of the low response rate, none of the results analysis had any statistical significance.

6. Conclusion

In this chapter, summaries of the achievements and findings will be discussed along with possibilities for future research and experimentation.

6.1. Achievements

As a result of this project, a complete architecture was designed and implemented for a semantic wireless sensor network platform. This platform was able to describe sensors, their observations and their values and make the data available via both a SPARQL endpoint and a linked data interface. It was shown that it is possible to perform basic statistical analysis on this data using the R statistics programming language.

A more complex architecture was then developed to allow the collection of observations from arbitrary sources with the automatic discovery of sensors. The data model was based on the model used for the first half of the project but extended to allow for human sensing. The ontology development necessary to accomodate human sensing raised some interesting issues in both the design and implementation stages that were discussed in the evaluation.

The combination of the two parts of the project provided adequate exposure to the low-level networking technologies and semantic web technologies to form a comprehensive evaluation of their suitability in sensing applications.

6.2. Future Work

Futher work could be done to analyse the SSN ontology to better allow for mobile sensors and oppertunistic sensing. It was shown in the evaluation of the SSN ontology that only an abstract representation of human sensors and sensing processes could be represented and many vocabulary terms in the SSN ontology were incompatible with human sensing.

Futher work could also be done to investigate the possibility of adapting WebDAV extensions to HTTP for dealing with semantic web resources as opposed to just filesystem resources. The semantics of the `POST` request in HTTP is very poorly defined and does not foster the creation of standards in this area.

6.3. Summary

Overall the project was a success in that it has performed well in producing the two platforms allowing for the exploration of the technologies that needed to be explored in order to fulfill the aims set by the motivation of the project. A number of problems with the technologies were highlighted and areas of future work were identified.

CoAP is a step in the right direction fixing a number of problems with the use of REST within constrained environments though it is not a complete solution. The use of semantic web technologies has shown that they provide benifits when sharing data or allowing collaboration between sensor deployments but for the standalone storage and analysis of data, the performance penalties do not outweigh the advantages at present. The SSN ontology is going to require a substaintial amount of work in order to allow for human sensors to be represented in a less abstract way.

This project has laid the groundwork for a number of futher research projects and each of these will require investigation before any improvements can realistically be implemented by new standards or integrated into existing ones.

Bibliography

[1] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext Transfer Protocol – HTTP/1.0. RFC 1945 (Informational), May 1996.

[2] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web (Berners-Lee et. al 2001). May 2001.

[3] N. Bikakis, C. Tsinaraki, N. Gioldasis, I. Stavrakantonakis, and S. Christodoulakis. The XML and Semantic Web Worlds: Technologies, Interoperability and Integration. A survey of the State of the Art. In *Semantic Hyper/Multi-media Adaptation: Schemes and Applications*. Springer, 2012.

[4] Jean-Paul Calbimonte, Hoyoung Jeung, Oscar Corcho, and Karl Aberer. Semantic Sensor Data Search in a Large-Scale Federated Sensor Network. In *Proceedings of the 4th International Workshop on Semantic Sensor Networks 2011 (SSN11)*, 2011.

[5] D. Crockford. The application/json Media Type for JavaScript Object Notation (JSON). RFC 4627 (Informational), July 2006.

[6] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460 (Draft Standard), December 1998. Updated by RFCs 5095, 5722, 5871, 6437, 6564, 6935.

[7] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008. Updated by RFCs 5746, 5878, 6176.

[8] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), June 1999. Updated by RFCs 2817, 5785, 6266, 6585.

[9] Tim Finin, Will Murnane, Anand Karandikar, Nicholas Keller, Justin Martineau, and Mark Dredze. Annotating named entities in twitter data with crowdsourcing. *Organization*, (June):80–88, 2010.

[10] A. Freier, P. Karlton, and P. Kocher. The Secure Sockets Layer (SSL) Protocol Version 3.0. RFC 6101 (Historic), August 2011.

[11] Y. Goland, E. Whitehead, A. Faizi, S. Carter, and D. Jensen. HTTP Extensions for Distributed Authoring – WEBDAV. RFC 2518 (Proposed Standard), February 1999. Obsoleted by RFC 4918.

[12] Google Inc. SPDY/2 Specification. http://dev.chromium.org/spdy/spdy-protocol/spdy-protocol-draft2.

[13] Google Inc. SPDY/3 Specification. http://dev.chromium.org/spdy/spdy-protocol/spdy-protocol-draft3.

[14] IEEE. IEEE 802.15 WPAN Task Group 4. http://www.ieee802.org/15/pub/TG4.html.

[15] IETF. Constrained RESTful Environments (core) Charter. https://datatracker.ietf.org/doc/charter-ietf-core/.

[16] IETF. IPv6 over Low power WPAN (6lowpan) Charter. https://datatracker.ietf.org/doc/charter-ietf-6lowpan/.

[17] Laurent Lefort, Cory Henson, Kerry Taylor, Payam Barnaghi, Michael Compton, Oscar Corcho, Ral Garca Castro, John Graybeal, Arthur Herzog, Krzysztof Janowicz, and et al. Semantic Sensor Network XG Final Report. http://www.w3.org/2005/Incubator/ssn/XGR-ssn-20110628/, 2011. [Online; Retrieved 26th Feb 2013].

[18] Frank Manola and Eric Miller. RDF Primer. http://www.w3.org/TR/rdf-primer/, February 2004. [Online; Retrieved 26th Feb 2013].

[19] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler. Transmission of IPv6 Packets over IEEE 802.15.4 Networks. RFC 4944 (Proposed Standard), September 2007. Updated by RFCs 6282, 6775.

[20] J. Postel. User Datagram Protocol. RFC 768 (INTERNET STANDARD), August 1980.

[21] J. Postel. Transmission Control Protocol. RFC 793 (INTERNET STANDARD), September 1981. Updated by RFCs 1122, 3168, 6093, 6528.

[22] Toby Segaran, Colin Evans, and Jamie Taylor. *Programming the Semantic Web.* O'Reilly Media, 2009.

[23] Zach Shelby, Klaus Hartke, Carsten Bormann, and Brian Frank. Constrained Application Protocol (CoAP). http://www.rfc-editor.org/internet-drafts/draft-ietf-core-coap-13.txt, December 2012.

[24] W3C Semantic Sensor Network Incubator Group. Semantic Sensor Network Ontology. http://www.w3.org/2005/Incubator/ssn/ssnx/ssn. [Online; Retrieved 26th Feb 2013].

A. Wireless Sensor Network Deployment Manual

A.1. Pre-Requisites

- A Linux machine running kernel version >= 2.6
- Apache Fuseki
- Apache HTTP Server
- The AVR gcc toolchain
- The git version control system
- An AVR RAVENUSB flashed with the Contiki Jackdaw software
- One or more Zigduino (>= r1)

A.2. Configuring and Flashing Sensor Nodes

For each sensor node, it is necessary to set the node ID in the source file, build and then flash the client application onto the device.

Begin by cloning the Contiki port to the Zigduino platform by running:

```
git clone https://github.com/quikshot/contiki-avr-zigduino.git
```

This will create a folder containing the Contiki distribution. Create a new folder within platforms/avr-zigduino/tests named http-client and copy the provided http-client.c file into this new folder. Then add a Makefile to this folder with the contents:

```
CONTIKI_PROJECT = http-client
all: $(CONTIKI_PROJECT)

APPS = webbrowser

WITH_UIP6=1
UIP_CONF_IPV6=1
```

```
CONTIKI = ../../../..
include $(CONTIKI)/Makefile.include
```

For each sensor, edit the http-client.c file on line 58 and change the number following sid= to be a unique node ID. With the USB cable attached to the Zigduino, then run:

```
make TARGET=avr-zigduino upload
```

This will flash the Zigduino with the HTTP client software. Once all the sensor nodes have been flashed, the next step is to configure the server.

A.3. Configuring the logging server

The first step in configuring the server is to connect the AVR RAVEN USB stick and confirm that it has installed correctly by running:

```
ifconfig -a
```

A new network interface will have appeared. This interface requires an IPv6 address in order to allow the sensor nodes to communicate with it. Add it by running:

```
ip -6 addr add fdfd::1 dev usb0
```

If the new interface is not named usb0, replace this with the name of the interface.

Now install the logger script into the webroot of the Apache HTTP Server:

```
cp /path/to/update.php /var/www/
```

The PHP script may need modifying to point to the local SPARQL/Update endpoint for the Apache Fuseki server.

The server will now log any observations submitted and the Zigduino(s) will begin submitting observations as soon as they are powered.

B. uktemp.net Application - User Manual

B.1. Using the uktemp.net map frontend

The uktemp.net application provides a web interface with a visualisation of recent observations on an interactive map. To access this frontend, visit `http://uktemp.net/`.

The map will show markers where there have been recent observations made. Where there are many markers close together, it is possible to zoom in to get finer detail by either using the scroll wheel over the map or using the zoom slider on the left of the map. It is also possible to click and drag the map to navigate around.

To view the data in recent observations for a location, hover the mouse over a marker. This will produce a pop up bubble containing:

- The postcode district
- Recent observations from Twitter
- The latest weather station reading

Each observation from Twitter will be shown with details of:

- The username of the submitter
- The observation value they gave
- How long ago the observation was
- The nearest weather station reading (if available)

B.2. Submitting an observation via Twitter

The uktemp.net application is constantly listening for new observations submitted by Twitter. To submit the observation, no interaction with the uktemp.net application is required. There are a large number of ways to post Tweets, such as command line clients, mobile clients and aggregated social media clients, but this guide will only cover the use of the official Twitter web interface. The format of the tweet given may however be used to submit a tweet from any Twitter client capable of doing so.

This guide assumes that you have already created a Twitter account. If you have not, visit `http://www.twitter.com/` and follow the onscreen instructions to create an account. If you do already have an account, begin by logging in to the Twitter website.

On the right hand side of the screen is a white box, indicated in figure B.1, that is used to write new tweets. This is where you should construct a tweet according to the following format:

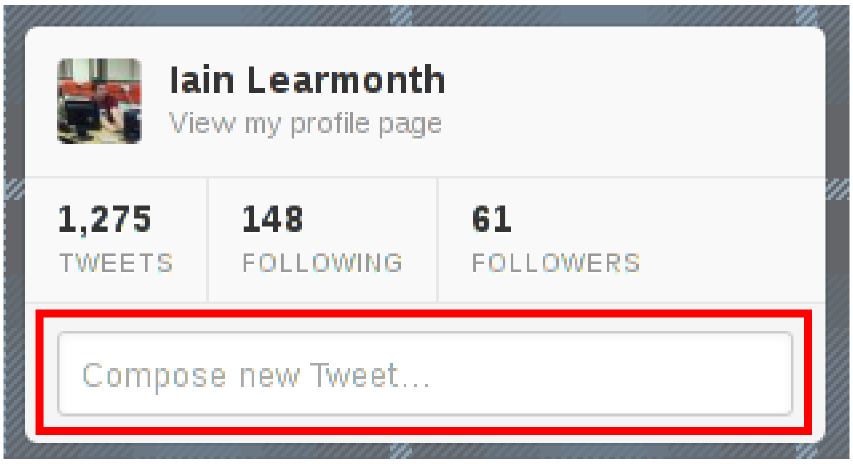

Figure B.1.: Screenshot of Twitter web frontend with the box used to compose new tweets highlighted

- The tweet must contain an observation of the outdoor temperature in quotation marks (")
- The tweet must contain the hashtag uktemp
- The tweet must contain the postcode district that the observation is for if the tweet is not geotagged

You will know if a tweet will be geotagged as underneath the tweet box will be shown details of your current location. If the details shown for your location are incorrect, you can include your postcode district in the tweet and this will be used instead of the geotag information. An example tweet is shown in figure B.2.

Note that only the text within quotation marks is used for the observation value and so you can still form an intelligible tweet that makes sense to both your followers and the uktemp.net application.

B.3. Using the SPARQL endpoint and Linked Data interface

All the data collected by the uktemp.net application is available through both a SPARQL endpoint and a linked data interface.

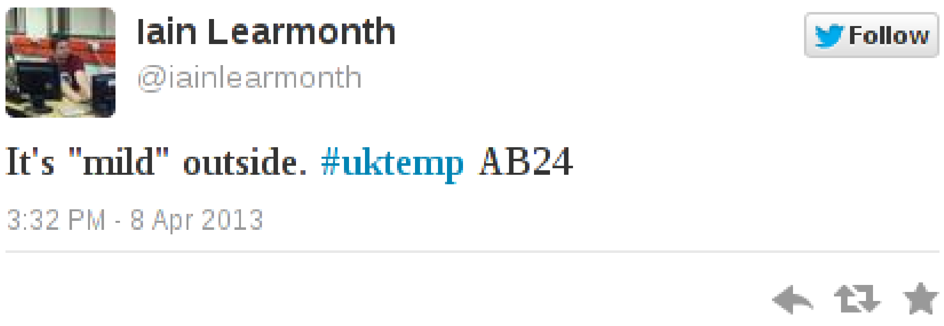
Iain Learmonth
@iainlearmonth
Follow
It's "mild" outside. #uktemp AB24
3:32 PM - 8 Apr 2013

Figure B.2.: An example tweet submitting an observation to the uktemp.net application

To use the SPARQL endpoint, point your SPARQL client to `http://uktemp.net/sparql`.

All the URIs used by the uktemp.net application are in the URI namespace of `http://uktemp.net/dataset/` and are dereferenceable as both human readable HTML and machine readable RDF/XML.

The main ontology in use is the Semantic Sensor Network Ontology by W3C. This is extended by the Human Sensing Ontology and the uktemp.net application ontology. The QUDT ontology is also used and extended by the Qualitative Units Ontology. The prefixes for all ontologies used are given in the table below:

Prefix	Namespace
ssn:	`http://purl.oclc.org/NET/ssnx/ssn#`
foaf:	`http://xmlns.com/foaf/0.1/`
rdf:	`http://www.w3.org/1999/02/22-rdf-syntax-ns#`
hsense:	`http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#`
xsd:	`http://www.w3.org/2001/XMLSchema#`
qunits:	`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#`
qudt:	`http://qudt.org/1.1/schema/qudt#`
rdfs:	`http://www.w3.org/2000/01/rdf-schema#`
uktemp:	`http://uktemp.net/ont/#`
skos:	`http://www.w3.org/2004/02/skos/core#`

Each Twitter user and Weather Underground weather station is represented as a subclass of ssn:SensingDevice and observes a ssn:Property of a ssn:FeatureOfInterest. The URI schemes for these are given in the table below:

Resource	URI Scheme
Location	`http://uktemp.net/dataset/location/[POSTCODE-DISTRICT]`
Property (Temperature)	`http://uktemp.net/dataset/temperature/[POSTCODE-DISTRICT]`

Observations are represented as ssn:Observation instances with their associated values represented as ssn:ObservationValue instances using the QUDT.and Qualitative Units ontologies. All observations are in the `http://uktemp.net/dataset/observation/`

namespace and all observation values have URIs in the namespace of their associated observations.

To browse the dataset using either a web browser or a semantic browser, simply point the browser to a URI in the dataset and content negotiation will return a suitable document describing the resource.

C. uktemp.net Application - Maintainence Manual

This document describes how to deploy the uktemp.net application on a Linux server.

C.1. Pre-requisites

The application is largely self contained within a single Java web application archive (WAR file) although it does have some requirements before it can be deployed. These are:

- A Java Runtime Environment supporting Java SE 7 or above
  Recommended: Oracle JRE 7
- A Java Web Servlet Container supporting Java EE 7 or above
  Recommended: Apache Tomcat 7
- A HTTP server capable of reverse-proxying requests
  Recommended: Apache 2
- Access to the Internet

C.2. Installation Guide

The installation procedure consists of three steps. The first two relate to the RDF triplestore and its interfaces. Whilst an RDF triplestore with a SPARQL endpoint is a requirement, the a Linked Data frontend is not required for the application to function although the functionality will be limited.

C.2.1. RDF Triplestore Setup

If you already have a RDF triplestore available, such as an Apache Fuseki, Virtuoso or IBM DB/2 server, then it is only necessary to create a new service within that server to store a new graph and provide both a SPARQL and SPARQL/Update endpoint. In this case, you can now proceed to the next section.

If you are not already running a RDF triplestore server, the following instructions will show how to set up the Apache Fuseki server. Begin by downloading and unpacking the latest Apache Fuseki distribution:

```
cd /tmp
wget http://www.apache.org/dist/jena/binaries/jena-fuseki-0.2.6-
   distribution.tar.gz
tar xf jena-fuseki-0.2.6-distribution.tar.gz -C /usr/local/
cd /usr/local
mv jena-fuseki-0.2.6 fuseki
```

Next, copy the provided example Fuseki configuration file into the new Fuseki installation:

```
cp /path/to/config.ttl /usr/local/fuseki/
```

Finally, start the Fuseki server with:

```
cd /usr/local/fuseki
./fuseki-server --config ./config.ttl &
```

C.2.2. Optional: Linked Data Frontend

The Linked Data frontend allows for Semantic Browsers to view the data collected by the uktemp.net application. If data is being collected for internal use only and there is no need for a linked data frontend, you may skip to the next section.

The recommended Linked Data frontend for this project is Pubby. Pubby must be deployed in your Java web servlet container, so this must be working at this point.

Start by downloading and unpacking the latest Pubby distribution:

```
cd /tmp
wget http://wifo5-03.informatik.uni-mannheim.de/pubby/download/pubby
   -0.3.3.zip
unzip pubby-0.3.3.zip
mv pubby-0.3.3/webapp /path/to/webapps/pubby
```

Then install the provided configuration file with:

```
mv /path/to/config.ttl /path/to/webapps/pubby/
```

Restart the servlet container to complete the installation of Pubby. It will automatically be deployed.

C.2.3. The uktemp.net Application

Deployment of the uktemp.net application is a matter of simply copying the provided WAR file to the servlet container's webapps directory:

```
cp /path/to/SemanticSensorServer.war /path/to/webapps/
```

Upon restarting the servlet container, the application will automatically be deployed.

C.2.4. Reverse-proxy Configuration

In order to protect access to the servlet container's administration console and the SPARQL/Update endpoint, it is **strongly** recommended that the applications run behind a reverse-proxy. This allows access to the public facing parts of the web application on the standard TCP port 80 whilst allowing a firewall to block access from outside hosts on all other ports. Configuration of firewalls is beyond the scope of this document, try `man iptables` if you do not have a hardware firewall appliance.

This document assumes that you are using Apache 2, although nginx is also suitable for this task and does give the advantage of being more lightweight. The nginx wiki contains information of how to configure reverse proxying if you choose to use nginx.

The provided uktemp.net file contains a VirtualHost entry to be included in your Apache 2 sites configuration. To install it:

```
cp /path/to/uktemp.conf /etc/apache2/sites-available/
a2ensite uktemp.net
apache2ctl graceful
```

The final command restarts the Apache 2 web server to read the new configuration. The system is now deployed.

C.3. Creating a new module

The system has been designed as a series of modules. There are two data acquisition modules to acquire data from Twitter and Weather Underground and a presentation module to display the uktemp.net web frontend with the map. It is possible to easily create new modules.

C.3.1. Data Acquisition Module

There are two approaches that can be taken to creating a data acquisition module. The first is stream-based and the second is polling-based. The included Twitter solution uses the former and the Weather Underground solution uses the latter. If a streaming API is available, this is the recommended option as you can have the data store instantly updated rather than having to wait for a poll to occur to fetch new data.

Either a timer thread or a stream listener thread should be started from the `init()` method of the `ScraperInitialiser` class in the `uk.ac.abdn.cs01il.scrapers` package.

The data objects that must be fetched from the API for each observation are:

- The sensor that produced the observation
- The postcode district the observation was for
- The sampling time for the observation - for the streaming API, use of the local time may be acceptable
- The value, either qualitative or quantiative or both for the observation

The first stage for the submission of the observation to the triplestore is the creation or update of the sensor resource. The URI for the sensor resource should be within the URL space under `http://uktemp.net/dataset/`. The exact details of the URI for the sensor resource can be decided on a per-source basis but should at least be consistent.

The minimum requirements for the sensor resource is that it is labelled as a sensor with an *rdf:type* property linked to an *ssn:SensingDevice* (or a subclass of *ssn:SensingDevice*). If the resource is also could be known by another URI, an *owl:sameAs* property should be added. Other properties either from the SSN ontology or elsewhere can be added if desired. For example, Twitter sensors contain an *rdfs:seeAlso* property linking to the URI of the Twitter user's profile.

Do not be afraid to submit duplicate triples, only one will appear in the dataset and this ensures that no data is missed or not updated when it changes.

The second stage of the submission is to submit the actual observation data. This requires the creation of at least two resources. The Twitter module can create up to three, the observation itself, a qualitative value if present and a quantitative value if present. The Weather Underground module always produces three, the observation, a quantitative value using degrees celcius and a quantiative value in degrees farenheit.

The minimum requirements for the observation resource are that it is labelled as an *ssn:Observation* with an *rdf:type* property, that it has an *ssn:observationSamplingTime* with the time given as an *xsd:dateTime*, that it has a link to the property it observed by an *ssn:observedProperty* and a link to the sensor that observed it by an *ssn:observedBy* and finally links to all values associated with it by *ssn:observationValue*.

Resources for values should be created under the namespace of the observation and should be lablled as *ssn:ObservationValue* with an *rdf:type* property. If the value is quantitative, it should give the numerical value with a *qudt:numericValue* property and the appropriate unit from the unit ontology linked with the *qudt:unit* property. Qualitative values should be linked with the *qunits:qualitativeValue* property and the unit should always be *qunits:QualitativeUnit* or a sub-class of *qunits:QualitativeUnit*.

The following gives an example of how to perform SPARQL/Update queries on the RDF graph:

```
String sparql = "INSERT DATA ......";
UpdateRequest queryObj = UpdateFactory.create(sparql);
UpdateProcessor qexec = UpdateExecutionFactory.createRemote(
  queryObj, "http://localhost:3030/observations/update");
qexec.execute();
```

C.3.2. Presentation Module

New presentation modules should create a new sub-folder in the WebContent folder for their frontends. Any Java classes should be placed into a new package. Beyond this, there are no real constraints placed upon how presentation modules should function as they are not manipulating data from the RDF graph, only reading it.

The following gives an example of how to perform SPARQL queries against the RDF graph:

```
String sparql = "SELECT ......";
Query query = QueryFactory.create(sparql);
QueryExecution qExec = QueryExecutionFactory.createServiceRequest(
  "http://localhost:3030/observations/query", query);
ResultSet results = qExec.execSelect() ;
for ( ; results.hasNext() ; )
{
  QuerySolution soln = results.nextSolution() ;

  String exampleUri = soln.getResource("exampleUri").getURI();
  String exampleLiteral = soln.getLiteral("exampleLiteral").getString()
      ;
}
```

C.4. Notes

C.4.1. Deploying a new version

Simply overwriting the SemanticSensorServer.war file will not trigger an automatic re-deployment. You must also delete the SemanticSensorServer directory in the webapps folder of the servlet container as part of the deployment process is the unpacking of the archive. If the server can see an unpacked version, it will not check to see if the archive is newer. The correct process for deploying a new version is:

1. Stop the servlet container (Note: While the servlet container is down, the reverse proxying agent will return an HTTP 503 Service Unavailable status and an appropriate handler page.)

2. Delete the existing SemanticSensorServer folder and WAR file from the webapps directory
3. Upload the new SemanticSensorServer.war file to the webapps directory
4. Start the servlet container (Note: The reverse proxying agent may continue to return a HTTP 503 status code for up to a minute after the servlet container has restarted. This is normal and is allowing for the application to recover in case it was not responding due to heavy load.)

C.4.2. Running multiple instances

If it is necessary to run multiple instances of the application, note that the Twitter API key used allows only one connection to the API at any given time and so a new API key would need to be placed into the `TwitterScraper` class.

C.5. Code Overview

This section gives an overview of the code at a package level. For specific detail on classes and their methods, see the attached generated JavaDoc.

uk.ac.abdn.csd.cs01il.map This package contains classes used by the uktemp.net web frontend. Note that the web frontend also has code in the WebContent directory.

uk.ac.abdn.csd.cs01il.scrapers This package contains a single class, the scraper initialiser servlet which is loaded when the servlet container starts and is used to start the data acquisition modules background threads.

uk.ac.abdn.csd.cs01il.tweetsense Twitter data acquisition module.

uk.ac.abdn.csd.cs01il.wunderground Weather Underground data acquisition module.

Contents

1 Class Index **1**

1.1 Class List . . . 1

2 Class Documentation **3**

2.1 uk.ac.abdn.csd.cs01il.util.DateTimeUtil Class Reference . . . 3

2.1.1 Detailed Description . . . 3

2.1.2 Member Function Documentation . . . 3

2.1.2.1 getIsoDate . . . 3

2.2 uk.ac.abdn.csd.cs01il.tweetsense.Geolocation Class Reference . . . 3

2.2.1 Detailed Description . . . 4

2.2.2 Constructor & Destructor Documentation . . . 4

2.2.2.1 Geolocation . . . 4

2.2.3 Member Function Documentation . . . 4

2.2.3.1 getPostcode . . . 4

2.3 uk.ac.abdn.csd.cs01il.map.Markers Class Reference . . . 4

2.3.1 Detailed Description . . . 5

2.3.2 Constructor & Destructor Documentation . . . 5

2.3.2.1 Markers . . . 5

2.3.3 Member Function Documentation . . . 5

2.3.3.1 getJSON . . . 5

2.4 uk.ac.abdn.csd.cs01il.wunderground.Observation Class Reference . . . 5

2.4.1 Detailed Description . . . 5

2.4.2 Constructor & Destructor Documentation . . . 6

2.4.2.1 Observation . . . 6

2.4.3 Member Function Documentation . . . 6

2.4.3.1 commit . . . 6

2.5 uk.ac.abdn.csd.cs01il.tweetsense.Observation Class Reference . . . 6

2.5.1 Detailed Description . . . 6

2.5.2 Constructor & Destructor Documentation . . . 6

2.5.2.1 Observation . . . 6

2.5.3 Member Function Documentation . . . 7

2.5.3.1 commit 7
2.6 uk.ac.abdn.csd.cs01il.scrapers.ScraperInitialiser Class Reference 7
2.6.1 Detailed Description 7
2.6.2 Constructor & Destructor Documentation 8
2.6.2.1 ScraperInitialiser 8
2.6.3 Member Function Documentation 8
2.6.3.1 doGet 8
2.6.3.2 doPost 8
2.6.3.3 init 8
2.7 uk.ac.abdn.csd.cs01il.util.SparqlHelper Class Reference 8
2.7.1 Detailed Description 8
2.7.2 Member Data Documentation 9
2.7.2.1 prefixes 9
2.8 uk.ac.abdn.csd.cs01il.util.Temperature Class Reference 9
2.8.1 Detailed Description 9
2.8.2 Constructor & Destructor Documentation 9
2.8.2.1 Temperature 9
2.8.3 Member Function Documentation 9
2.8.3.1 getUnit 9
2.8.3.2 getValue 10
2.8.4 Member Data Documentation 10
2.8.4.1 unit 10
2.8.4.2 value 10
2.9 uk.ac.abdn.csd.cs01il.tweetsense.TwitterScraper Class Reference 10
2.9.1 Detailed Description 10
2.9.2 Member Function Documentation 11
2.9.2.1 run 11
2.10 uk.ac.abdn.csd.cs01il.wunderground.WundergroundScraper Class Reference 11
2.10.1 Detailed Description 11
2.10.2 Constructor & Destructor Documentation 11
2.10.2.1 WundergroundScraper 11
2.10.3 Member Function Documentation 11
2.10.3.1 run 11

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

uk.ac.abdn.csd.cs01il.util.DateTimeUtil 3
uk.ac.abdn.csd.cs01il.tweetsense.Geolocation 3
uk.ac.abdn.csd.cs01il.map.Markers 4
uk.ac.abdn.csd.cs01il.wunderground.Observation 5
uk.ac.abdn.csd.cs01il.tweetsense.Observation 6
uk.ac.abdn.csd.cs01il.scrapers.ScraperInitialiser 7
uk.ac.abdn.csd.cs01il.util.SparqlHelper 8
uk.ac.abdn.csd.cs01il.util.Temperature 9
uk.ac.abdn.csd.cs01il.tweetsense.TwitterScraper 10
uk.ac.abdn.csd.cs01il.wunderground.WundergroundScraper 11

Chapter 2

Class Documentation

2.1 uk.ac.abdn.csd.cs01il.util.DateTimeUtil Class Reference

Static Public Member Functions

- static String **getIsoDate** (Date date)

2.1.1 Detailed Description

The **DateTimeUtil** (p. 3) utility class contains functions for dealing with the xsd:dateTime datatype.

Author

irl

Definition at line 15 of file DateTimeUtil.java.

2.1.2 Member Function Documentation

2.1.2.1 static String uk.ac.abdn.csd.cs01il.util.DateTimeUtil.getIsoDate (Date *date*) `[static]`

Generate a ISO 8601 date

Parameters

date	a Date instance

Returns

a string representing the date in the ISO 8601 format

Definition at line 22 of file DateTimeUtil.java.

The documentation for this class was generated from the following file:

- /home/irl/seriousbusiness/SemanticSensorServer/src/uk/ac/abdn/csd/cs01il/util/DateTimeUtil.java

2.2 uk.ac.abdn.csd.cs01il.tweetsense.Geolocation Class Reference

Public Member Functions

- **Geolocation** (double lat, double lon)
- String **getPostcode** () throws Exception

2.2.1 Detailed Description

The **Geolocation** (p. 3) class provides logic for converting between the latitude and longitude reported for a tweet and the postcode district that the point belongs to.

Author

Iain Learmonth `iain.learmonth.09@aberdeen.ac.uk`

Definition at line 22 of file Geolocation.java.

2.2.2 Constructor & Destructor Documentation

2.2.2.1 uk.ac.abdn.csd.cs01il.tweetsense.Geolocation.Geolocation (double *lat*, double *lon*)

Constructor for **Geolocation** (p. 3) objects initialising them with latitude and longitude.

Parameters

lat	latitude
lon	longitude

Definition at line 36 of file Geolocation.java.

2.2.3 Member Function Documentation

2.2.3.1 String uk.ac.abdn.csd.cs01il.tweetsense.Geolocation.getPostcode ( ) throws Exception

Returns the postcode for the location.

Returns

the postcode for the location

Exceptions

Exception	when postcode cannot be resolved

Definition at line 88 of file Geolocation.java.

The documentation for this class was generated from the following file:

- /home/irl/seriousbusiness/SemanticSensorServer/src/uk/ac/abdn/csd/cs01il/tweetsense/Geolocation.java

2.3 uk.ac.abdn.csd.cs01il.map.Markers Class Reference

Public Member Functions

- **Markers** ()

- String **getJSON** ()

2.3.1 Detailed Description

The **Markers** (p. 4) class provides the logic to produce the markers placed onto the Google Maps map on the web frontend.

Author

Iain Learmonth `iain.learmonth.09@aberdeen.ac.uk`

Definition at line 29 of file Markers.java.

2.3.2 Constructor & Destructor Documentation

2.3.2.1 uk.ac.abdn.csd.cs01il.map.Markers.Markers ( )

Constructor for **Markers** (p. 4). Initialises the class with recent observation data.

Definition at line 37 of file Markers.java.

2.3.3 Member Function Documentation

2.3.3.1 String uk.ac.abdn.csd.cs01il.map.Markers.getJSON ( )

Return the JSON to be included in the Google Maps Javascript

Returns

the JSON to be included in the Google Maps Javascript

Definition at line 121 of file Markers.java.

The documentation for this class was generated from the following file:

- /home/irl/seriousbusiness/SemanticSensorServer/src/uk/ac/abdn/csd/cs01il/map/Markers.java

2.4 uk.ac.abdn.csd.cs01il.wunderground.Observation Class Reference

Public Member Functions

- **Observation** (String stationName, String postcode, String temp, Date created)
- void **commit** ()

2.4.1 Detailed Description

The **Observation** (p. 5) class represents an observation from Weather Underground but should be initialised once the data from the API has been extracted. The **Observation** (p. 5) class can commit observations to the triplestore.

Author

Iain Learmonth `iain.learmonth.09@aberdeen.ac.uk`

Definition at line 29 of file Observation.java.

2.4.2 Constructor & Destructor Documentation

2.4.2.1 uk.ac.abdn.csd.cs01il.wunderground.Observation.Observation (String *stationName,* String *postcode,* String *temp,* Date *created*)

Create a Weather Underground observation

Parameters

stationName	The weather station ID
postcode	The postcode for the observation
temp	The temperature
created	

Definition at line 46 of file Observation.java.

2.4.3 Member Function Documentation

2.4.3.1 void uk.ac.abdn.csd.cs01il.wunderground.Observation.commit ( )

Commit the observation to the RDF graph

Definition at line 65 of file Observation.java.

The documentation for this class was generated from the following file:

- /home/irl/seriousbusiness/SemanticSensorServer/src/uk/ac/abdn/csd/cs01il/wunderground/Observation.java

2.5 uk.ac.abdn.csd.cs01il.tweetsense.Observation Class Reference

Public Member Functions

- **Observation** (long id, String user, String postcode, String temp, String qtemp, Date created)
- void **commit** ()

2.5.1 Detailed Description

The **Observation** (p. 6) class represents an observation from Twitter but should be initialised once the data from the tweet has been extracted. The **Observation** (p. 6) class can commit observations to the triplestore.

Author

Iain Learmonth `iain.learmonth.09@aberdeen.ac.uk`

Definition at line 27 of file Observation.java.

2.5.2 Constructor & Destructor Documentation

2.5.2.1 uk.ac.abdn.csd.cs01il.tweetsense.Observation.Observation (long *id,* String *user,* String *postcode,* String *temp,* String *qtemp,* Date *created*)

Create a new Twitter observation

Parameters

id	The ID of the tweet
user	The username of tweet submitter
postcode	The postcode extracted from the tweet (or result from geolocation)
temp	The qualitative description of the temperature (or null if none given)
qtemp	The quantitative description of the temperature (or null if none given)
created	The observation sampling time (tweet creation time)

Definition at line 46 of file Observation.java.

2.5.3 Member Function Documentation

2.5.3.1 void uk.ac.abdn.csd.cs01il.tweetsense.Observation.commit ( )

Commit the observation to the triplestore

Definition at line 64 of file Observation.java.

The documentation for this class was generated from the following file:

- /home/irl/seriousbusiness/SemanticSensorServer/src/uk/ac/abdn/csd/cs01il/tweetsense/Observation.java

2.6 uk.ac.abdn.csd.cs01il.scrapers.ScraperInitialiser Class Reference

Inherits HttpServlet.

Public Member Functions

- **ScraperInitialiser** ()
- void **init** ()

Protected Member Functions

- void **doGet** (HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
- void **doPost** (HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException

2.6.1 Detailed Description

The Scrapers class, a servlet, contains the logic for launching the scrapers upon deployment of the application in a servlet container. This servlet has a "load-on-startup" value of 1.

Author

Iain Learmonth `iain.learmonth.09@aberdeen.ac.uk`

Definition at line 23 of file ScraperInitialiser.java.

2.6.2 Constructor & Destructor Documentation

2.6.2.1 uk.ac.abdn.csd.cs01il.scrapers.ScraperInitialiser.ScraperInitialiser ( )

Default constructor.

Definition at line 29 of file ScraperInitialiser.java.

2.6.3 Member Function Documentation

2.6.3.1 void uk.ac.abdn.csd.cs01il.scrapers.ScraperInitialiser.doGet (HttpServletRequest *request*, HttpServletResponse *response*) throws ServletException, IOException `[protected]`

See Also

HttpServlet::doGet(HttpServletRequest request, HttpServletResponse response)

Definition at line 36 of file ScraperInitialiser.java.

2.6.3.2 void uk.ac.abdn.csd.cs01il.scrapers.ScraperInitialiser.doPost (HttpServletRequest *request*, HttpServletResponse *response*) throws ServletException, IOException `[protected]`

See Also

HttpServlet::doPost(HttpServletRequest request, HttpServletResponse response)

Definition at line 43 of file ScraperInitialiser.java.

2.6.3.3 void uk.ac.abdn.csd.cs01il.scrapers.ScraperInitialiser.init ( )

Launch the scrapers. Called automatically by the servlet container upon deployment.

Definition at line 51 of file ScraperInitialiser.java.

The documentation for this class was generated from the following file:

- /home/irl/seriousbusiness/SemanticSensorServer/src/uk/ac/abdn/csd/cs01il/scrapers/ScraperInitialiser.java

2.7 uk.ac.abdn.csd.cs01il.util.SparqlHelper Class Reference

Static Public Attributes

- static final String **prefixes** = "PREFIX skos: <http://www.w3.org/2004/02/skos/core#> "

2.7.1 Detailed Description

The **SparqlHelper** (p. 8) utility class contains common functions used by the uktemp.net application for communicating with the triplestore via SPARQL.

Author

Iain Learmonth `iain.learmonth.09@aberdeen.ac.uk`

Definition at line 11 of file SparqlHelper.java.

2.7.2 Member Data Documentation

2.7.2.1 final String uk.ac.abdn.csd.cs01il.util.SparqlHelper.prefixes = "PREFIX skos: <http://www.w3.org/2004/02/skos/core#> " `[static]`

All prefixes used by the uktemp.net application in SPARQL syntax

Definition at line 16 of file SparqlHelper.java.

The documentation for this class was generated from the following file:

- /home/irl/seriousbusiness/SemanticSensorServer/src/uk/ac/abdn/csd/cs01il/util/SparqlHelper.java

2.8 uk.ac.abdn.csd.cs01il.util.Temperature Class Reference

Public Member Functions

- **Temperature** (String input)
- String **toString** ()
- String **getUnit** ()
- double **getValue** ()

Public Attributes

- String **unit**
- double **value**

2.8.1 Detailed Description

The **Temperature** (p. 9) class is provided to assist in dealing with quantitative temperature values and their units.

Author

Iain Learmonth `iain.learmonth.09@aberdeen.ac.uk`

Definition at line 10 of file Temperature.java.

2.8.2 Constructor & Destructor Documentation

2.8.2.1 uk.ac.abdn.csd.cs01il.util.Temperature.Temperature (String *input*)

Create a new temperature instance using the input string

Parameters

input	temperature string

Definition at line 27 of file Temperature.java.

2.8.3 Member Function Documentation

2.8.3.1 String uk.ac.abdn.csd.cs01il.util.Temperature.getUnit ( )

Get the unit

Returns

the unit

Definition at line 43 of file Temperature.java.

2.8.3.2 double uk.ac.abdn.csd.cs01il.util.Temperature.getValue ( )

Get the value

Returns

the value

Definition at line 51 of file Temperature.java.

2.8.4 Member Data Documentation

2.8.4.1 String uk.ac.abdn.csd.cs01il.util.Temperature.unit

A single character representing the unit, either C, F or K

Definition at line 15 of file Temperature.java.

2.8.4.2 double uk.ac.abdn.csd.cs01il.util.Temperature.value

The numerical value of the temperature using the unit given in the unit variable

Definition at line 20 of file Temperature.java.

The documentation for this class was generated from the following file:

- /home/irl/seriousbusiness/SemanticSensorServer/src/uk/ac/abdn/csd/cs01il/util/Temperature.java

2.9 uk.ac.abdn.csd.cs01il.tweetsense.TwitterScraper Class Reference

Public Member Functions

- void **run** ()

2.9.1 Detailed Description

The **TwitterScraper** (p. 10) class is the main class for the Twitter data acquisition module. It is implemented as a thread so as to run in the background.

Author

Iain Learmonth `iain.learmonth.09@aberdeen.ac.uk`

Definition at line 23 of file TwitterScraper.java.

2.9.2 Member Function Documentation

2.9.2.1 void uk.ac.abdn.csd.cs01il.tweetsense.TwitterScraper.run ( )

Start the Twitter data acquisition module.

Definition at line 28 of file TwitterScraper.java.

The documentation for this class was generated from the following file:

- /home/irl/seriousbusiness/SemanticSensorServer/src/uk/ac/abdn/csd/cs01il/tweetsense/TwitterScraper.java

2.10 uk.ac.abdn.csd.cs01il.wunderground.WundergroundScraper Class Reference

Inherits TimerTask.

Public Member Functions

- **WundergroundScraper** ()
- void **run** ()

2.10.1 Detailed Description

The **WundergroundScraper** (p. 11) class is the main class for the Twitter data acquisition module. It is implemented as a thread so as to run in the background.

Author

Iain Learmonth `iain.learmonth.09@aberdeen.ac.uk`

Definition at line 40 of file WundergroundScraper.java.

2.10.2 Constructor & Destructor Documentation

2.10.2.1 uk.ac.abdn.csd.cs01il.wunderground.WundergroundScraper.WundergroundScraper ( )

Create a new Weather Underground scraper

Definition at line 47 of file WundergroundScraper.java.

2.10.3 Member Function Documentation

2.10.3.1 void uk.ac.abdn.csd.cs01il.wunderground.WundergroundScraper.run ( )

Perform polling. Get all postcodes of interest

Get updated temperatures

Definition at line 54 of file WundergroundScraper.java.

The documentation for this class was generated from the following file:

- /home/irl/seriousbusiness/SemanticSensorServer/src/uk/ac/abdn/csd/cs01il/wunderground/Wunderground-Scraper.java

Index

commit
    uk::ac::abdn::csd::cs01il::tweetsense::Observation, 7
    uk::ac::abdn::csd::cs01il::wunderground::Observation, 6

doGet
    uk::ac::abdn::csd::cs01il::scrapers::ScraperInitialiser, 8
doPost
    uk::ac::abdn::csd::cs01il::scrapers::ScraperInitialiser, 8

Geolocation
    uk::ac::abdn::csd::cs01il::tweetsense::Geolocation, 4
getIsoDate
    uk::ac::abdn::csd::cs01il::util::DateTimeUtil, 3
getJSON
    uk::ac::abdn::csd::cs01il::map::Markers, 5
getPostcode
    uk::ac::abdn::csd::cs01il::tweetsense::Geolocation, 4
getUnit
    uk::ac::abdn::csd::cs01il::util::Temperature, 9
getValue
    uk::ac::abdn::csd::cs01il::util::Temperature, 10

init
    uk::ac::abdn::csd::cs01il::scrapers::ScraperInitialiser, 8

Markers
    uk::ac::abdn::csd::cs01il::map::Markers, 5

Observation
    uk::ac::abdn::csd::cs01il::tweetsense::Observation, 6
    uk::ac::abdn::csd::cs01il::wunderground::Observation, 6

prefixes
    uk::ac::abdn::csd::cs01il::util::SparqlHelper, 9

run
    uk::ac::abdn::csd::cs01il::tweetsense::TwitterScraper, 11
    uk::ac::abdn::csd::cs01il::wunderground::WundergroundScraper, 11

ScraperInitialiser
    uk::ac::abdn::csd::cs01il::scrapers::ScraperInitialiser, 8

Temperature
    uk::ac::abdn::csd::cs01il::util::Temperature, 9

uk.ac.abdn.csd.cs01il.map.Markers, 4
uk.ac.abdn.csd.cs01il.scrapers.ScraperInitialiser, 7
uk.ac.abdn.csd.cs01il.tweetsense.Geolocation, 3
uk.ac.abdn.csd.cs01il.tweetsense.Observation, 6
uk.ac.abdn.csd.cs01il.tweetsense.TwitterScraper, 10
uk.ac.abdn.csd.cs01il.util.DateTimeUtil, 3
uk.ac.abdn.csd.cs01il.util.SparqlHelper, 8
uk.ac.abdn.csd.cs01il.util.Temperature, 9
uk.ac.abdn.csd.cs01il.wunderground.Observation, 5
uk.ac.abdn.csd.cs01il.wunderground.WundergroundScraper, 11
uk::ac::abdn::csd::cs01il::map::Markers
    getJSON, 5
    Markers, 5
uk::ac::abdn::csd::cs01il::scrapers::ScraperInitialiser
    doGet, 8
    doPost, 8
    init, 8
    ScraperInitialiser, 8
uk::ac::abdn::csd::cs01il::tweetsense::Geolocation
    Geolocation, 4
    getPostcode, 4
uk::ac::abdn::csd::cs01il::tweetsense::Observation
    commit, 7
    Observation, 6
uk::ac::abdn::csd::cs01il::tweetsense::TwitterScraper
    run, 11
uk::ac::abdn::csd::cs01il::util::DateTimeUtil
    getIsoDate, 3
uk::ac::abdn::csd::cs01il::util::SparqlHelper
    prefixes, 9
uk::ac::abdn::csd::cs01il::util::Temperature
    getUnit, 9
    getValue, 10
    Temperature, 9
    unit, 10
    value, 10
uk::ac::abdn::csd::cs01il::wunderground::Observation
    commit, 6
    Observation, 6
uk::ac::abdn::csd::cs01il::wunderground::WundergroundScraper
    run, 11
    WundergroundScraper, 11

unit
 uk::ac::abdn::csd::cs01il::util::Temperature, 10

value
 uk::ac::abdn::csd::cs01il::util::Temperature, 10

WundergroundScraper
 uk::ac::abdn::csd::cs01il::wunderground::Wunderground-
 Scraper, 11

D. Fuseki Configuration File

E. Pubby Configuration File

```
# Pubby Configuration File
#
# Author: Iain R. Learmonth <iain.learmonth.09@aberdeen.ac.uk>
#

# Prefix declarations to be used in RDF output
@prefix conf: <http://richard.cyganiak.de/2007/pubby/config.rdf#> .
@prefix meta: <http://example.org/metadata#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .
@prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#> .
@prefix dbpedia: <http://localhost:8080/resource/> .
@prefix p: <http://localhost:8080/property/> .
@prefix yago: <http://localhost:8080/class/yago/> .
@prefix units: <http://dbpedia.org/units/> .
@prefix geonames: <http://www.geonames.org/ontology#> .
@prefix prv: <http://purl.org/net/provenance/ns#> .
@prefix prvTypes: <http://purl.org/net/provenance/types#> .
@prefix doap: <http://usefulinc.com/ns/doap#> .
@prefix void: <http://rdfs.org/ns/void#> .
@prefix ir: <http://www.ontologydesignpatterns.org/cp/owl/
    informationrealization.owl#> .
@prefix ssn: <http://purl.oclc.org/NET/ssnx/ssn#> .

# Server configuration section
<> a conf:Configuration;

    # Project name for display in page titles
    conf:projectName "Sensor Observations";

    # Homepage with description of the project for the link in the page
        header
    conf:projectHomepage <http://dev.erg.abdn.ac.uk/projects/iain/>;

    # The Pubby root, where the webapp is running inside the servlet
        container.
    conf:webBase <http://iain-1.erg.abdn.ac.uk/dataset/>;

    # URL of an RDF file whose prefix mapping is to be used by the
    # server; defaults to <>, which is *this* file.
    conf:usePrefixesFrom <>;

    # If labels and descriptions are available in multiple languages,
    # prefer this one.
    conf:defaultLanguage "en";

    # When the homepage of the server is accessed, this resource will
    # be shown.
    conf:indexResource <http://sensorobservations.internet/sensor/1>;

# Dataset configuration section for sensor observations
#
# URIs in the SPARQL endpoint: http://iain-1.erg.abdn.ac.uk/dataset/*
# URIs on the Web:             http://iain-1.erg.abdn.ac.uk/dataset/*

    conf:dataset [

        # SPARQL endpoint URL of the dataset
        conf:sparqlEndpoint <http://localhost:3030/observations/query>;

        # Common URI prefix of all resource URIs in the SPARQL dataset
        conf:datasetBase <http://sensorobservations.internet/>;

        # Will be appended to the conf:webBase to form the public
        # resource URIs; if not present, defaults to ""
        #conf:webResourcePrefix "resource/";

        # Fixes an issue with the server running behind an Apache proxy
          ;
        # can be ignored otherwise
        conf:fixUnescapedCharacters "(),'!$&*+;=@";

    ];

    .
```

Listing E.1: Pubby Configuration File

F. Apache VirtualHost Configuration File

```
<VirtualHost *:80>

  ServerName  uktemp.net
  ServerAlias www.uktemp.net

  RewriteEngine On

  RewriteCond %{HTTP_HOST}  ^www.uktemp.net$
  RewriteRule ^/(.*)$   http://uktemp.net/$1  [R=301,L]

  RewriteRule ^/ont(.*)$  http://homepages.abdn.ac.uk/i.learmonth/pages
     /uktemp/$1 [R,L]

  ProxyPass /dataset  http://localhost:8080/pubby
  ProxyPassReverse  /dataset  http://localhost:8080/pubby

  ProxyPass /sparql   http://localhost:3030/observations/query
  ProxyPassReverse  /sparql   http://localhost:3030/observations/query

  ProxyPass /   http://localhost:8080/SemanticSensorServer/
     HumanTemperature/
  ProxyPassReverse  /   http://localhost:8080/SemanticSensorServer/
     HumanTemperature/

</VirtualHost>
```

Listing F.1: Apache VirtualHost Configuration File

G. Human Sensing Ontology Documentation

Ontology IRI `http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#`

Authors Iain Learmonth

Imported Ontologies `http://purl.oclc.org/NET/ssnx/ssn`
`http://www.loa-cnr.it/ontologies/DUL.owl`

The Human Sensor Ontology extends the Semantic Sensor Network Ontology in order to provide a vocabulary for representing "human sensors". The vocabulary provides classes that can be extended per-task to create task-bounded human sensing devices and task-bounded human sensing processes.

G.1. Introduction

Humans are capable of sensing a wide variety of properties of their environment. This ontology provides a basis, to be extended on a per-project basis, for integrating human sensing with the Semantic Sensor Network Ontology.

Instead of declaring the sensing device to be a human, an eye, an ear or a hand, classes are provided for human sensors that are performing a specific sensing task.

In the majority of cases where human sensing is being used, a task will likely have been selected beforehand or, when working with historical data, the participants will likely be selected because they performed the task at the time. In a similar way, sensing processes are defined for each high-level sensing task.

G.2. Classes

G.2.1. HumanTemperatureSensingDevice

Class IRI

http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanTemperatureSensingDevi

Super-Classes

http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanSensingDevice

G.2.2. HumanPainSensingDevice

Class IRI

http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanPainSensingDevice

Super-Classes

http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanSensingDevice

It is important to note that instances of this class should be using the human's sense of pain, not visually (or otherwise) determining the level of pain being endured by another biological organism.

G.2.3. HumanAccelerationSensing

Class IRI

http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanAccelerationSensing

Super-Classes

http://www.loa-cnr.it/ontologies/DUL.owl#Process

G.2.4. HumanVisualSensingDevice

Class IRI

http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanVisualSensingDevice

Super-Classes

http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanSensingDevice

Sub-Classes

http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanIlluminanceSensingDevi

http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanLuminousIntensitySensi
http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanImageRecognitionSensi

G.2.5. HumanTouchSensingDevice

Class IRI
http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanTouchSensingDevice

Super-Classes
http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanSensingDevice

G.2.6. HumanIlluminanceSensingDevice

Class IRI
http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanIlluminanceSensingDevi

Super-Classes
http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanVisualSensingDevice

G.2.7. HumanTasteSensing

Class IRI
http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanTasteSensing

Super-Classes
http://www.loa-cnr.it/ontologies/DUL.owl#Process

G.2.8. HumanImageRecognitionSensing

Class IRI
http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanImageRecognitionSensi

Super-Classes
http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanVisualSensing

G.2.9. HumanTimeSensingDevice

Class IRI
    http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanTimeSensingDevice

Super-Classes
    http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanSensingDevice

G.2.10. HumanImageRecognitionSensingDevice

Class IRI
    http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanImageRecognitionSensi

Super-Classes
    http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanVisualSensingDevice

G.2.11. HumanVisualSensing

Class IRI
    http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanVisualSensing

Super-Classes
    http://www.loa-cnr.it/ontologies/DUL.owl#Process

Sub-Classes
    http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanLuminousIntensitySensi
    http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanIlluminanceSensing
    http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanImageRecognitionSensi

G.2.12. HumanSoundIntensitySensing

Class IRI
    http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanSoundIntensitySensing

Super-Classes
    http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanAudioSensing

G.2.13. HumanSoundRecognitionSensing

Class IRI
http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanSoundRecognitionSensi

Super-Classes
http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanAudioSensing

G.2.14. HumanTouchSensing

Class IRI
http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanTouchSensing

Super-Classes
http://www.loa-cnr.it/ontologies/DUL.owl#Process

G.2.15. HumanAudioSensingDevice

Class IRI
http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanAudioSensingDevice

Super-Classes
http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanSensingDevice

Sub-Classes
http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanSoundRecognitionSensi
http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanSoundIntensitySensingI

G.2.16. HumanLuminousIntensitySensing

Class IRI
http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanLuminousIntensitySensi

Super-Classes
http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanVisualSensing

G.2.17. HumanSmellSensing

Class IRI

`http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanSmellSensing`

Super-Classes

`http://www.loa-cnr.it/ontologies/DUL.owl#Process`

G.2.18. HumanAudioSensing

Class IRI

`http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanAudioSensing`

Super-Classes

`http://www.loa-cnr.it/ontologies/DUL.owl#Process`

Sub-Classes

`http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanSoundRecognitionSensi`
`http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanSoundIntensitySensing`

G.2.19. HumanTemperatureSensing

Class IRI

`http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanTemperatureSensing`

Super-Classes

`http://www.loa-cnr.it/ontologies/DUL.owl#Process`

G.2.20. HumanIlluminanceSensing

Class IRI

`http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanIlluminanceSensing`

Super-Classes

`http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanVisualSensing`

G.2.21. HumanAccelerationSensingDevice

Class IRI
  `http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanAccelerationSensingDev`

Super-Classes
  `http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanSensingDevice`

It is important to note that instances of this class should be using the human's sense of acceleration, not visually (or otherwise) determining the acceleration of another physical object.

G.2.22. HumanTimeSensing

Class IRI
  `http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanTimeSensing`

Super-Classes
  `http://www.loa-cnr.it/ontologies/DUL.owl#Process`

G.2.23. HumanBalanceSensing

Class IRI
  `http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanBalanceSensing`

Super-Classes
  `http://www.loa-cnr.it/ontologies/DUL.owl#Process`

G.2.24. HumanSensingDevice

Class IRI
  `http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanSensingDevice`

Super-Classes
  `http://www.loa-cnr.it/ontologies/DUL.owl#Organism`
  `http://www.loa-cnr.it/ontologies/DUL.owl#NaturalPerson`
  `http://purl.oclc.org/NET/ssnx/ssn#SensingDevice`

Sub-Classes
  `http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanPainSensingDevice`
  `http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanAudioSensingDevice`

http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanTasteSensingDevice
http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanSmellSensingDevice
http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanVisualSensingDevice
http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanTemperatureSensingDevi
http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanAccelerationSensingDev
http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanBalanceSensingDevice
http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanTouchSensingDevice
http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanTimeSensingDevice

G.2.25. HumanSmellSensingDevice

Class IRI
http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanSmellSensingDevice

Super-Classes
http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanSensingDevice

G.2.26. HumanTasteSensingDevice

Class IRI
http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanTasteSensingDevice

Super-Classes
http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanSensingDevice

G.2.27. HumanPainSensing

Class IRI
http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanPainSensing

Super-Classes
http://www.loa-cnr.it/ontologies/DUL.owl#Process

G.2.28. HumanSoundRecognitionSensingDevice

Class IRI
http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanSoundRecognitionSensin

Super-Classes

`http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanAudioSensingDevice`

G.2.29. HumanSoundIntensitySensingDevice

Class IRI

`http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanSoundIntensitySensingD`

Super-Classes

`http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanAudioSensingDevice`

G.2.30. HumanBalanceSensingDevice

Class IRI

`http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanBalanceSensingDevice`

Super-Classes

`http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanSensingDevice`

It is important to note that instances of this class should be using the human's sense of balance, not visually (or otherwise) determining whether or not another physical object is balanced.

G.2.31. HumanLuminousIntensitySensingDevice

Class IRI

`http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanLuminousIntensitySensi`

Super-Classes

`http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanVisualSensingDevice`

H. Qualitative Units Ontology Documentation

Ontology IRI `http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#`

Authors Iain Learmonth

Imported Ontologies `http://qudt.org/1.1/schema/qudt`

The Qualitative Units Ontology is designed to complement the QUDT ontology by providing the ability to store measurement values in a qualitative format. By extending the QUDT ontology, it allows these qualitative values to be used in a similar way to quantitative values.

H.1. Introduction

http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/qunits-example.png

H.2. Classes

H.2.1. QualitativeInformationEntropyUnit

Class IRI
`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeInformationEntro`

Super-Classes
`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeUnit`
`http://qudt.org/schema/qudt#InformationEntropyUnit`

A qualtative description of information entropy

H.2.2. QualitativeHumanCountUnit

Class IRI

`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeHumanCountUnit`

Super-Classes

`http://qudt.org/schema/qudt#HumanUnit`
`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeUnit`

A qualitative description of a count of humans

H.2.3. QualitativeCurvatureUnit

Class IRI

`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeCurvatureUnit`

Super-Classes

`http://qudt.org/schema/qudt#CurvatureUnit`
`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeUnit`

A qualitative description of a measure of curvature

H.2.4. QualitativeDataRateUnit

Class IRI

`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeDataRateUnit`

Super-Classes

`http://qudt.org/schema/qudt#DataRateUnit`
`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeUnit`

A qualitative description of a data rate

H.2.5. QualitativeAngularMassUnit

Class IRI

`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeAngularMassUnit`

Super-Classes

`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeUnit`
`http://qudt.org/schema/qudt#AngularMassUnit`

A qualitative description of angular mass

H.2.6. QualitativeTimeUnit

Class IRI

`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeTimeUnit`

Super-Classes

`http://qudt.org/schema/qudt#TimeUnit`
`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeUnit`

A qualitative description of a measure of time

H.2.7. QualitativeUnit

Class IRI

`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeUnit`

Super-Classes

`http://qudt.org/schema/qudt#Unit`

Sub-Classes

`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeForceUnit`
`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeIlluminanceUnit`
`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeLengthUnit`
`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeLuminousIntensit`
`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeHeartRateUnit`
`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeConcentrationUni`
`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeAccelerationUnit`
`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeAngularMassUnit`
`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeMomementumUnit`
`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeWorkUnit`
`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeCurvatureUnit`
`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeVolumeUnit`
`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeRespiratoryRateU`
`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeVideoFrameRateUn`
`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeMassUnit`
`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeVelocityUnit`
`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeTemperatureUnit`
`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeThermalConductiv`
`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeHumanCountUnit`
`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeConductanceUnit`
`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeTurbidityUnit`

http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeInformationEntro
http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativePlaneAngleUnit
http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeAreaUnit
http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativePermeabilityUnit
http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeTimeUnit
http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeDataRateUnit
http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeResistanceUnit
http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeDynamicViscosity
http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeCountUnit
http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeFrequencyUnit
http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeBendingMomentUni

Subclasses of this class are qualitative descriptions of measurements

H.2.8. QualitativeIlluminanceUnit

Class IRI

http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeIlluminanceUnit

Super-Classes

http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeUnit
http://qudt.org/schema/qudt#IlluminanceUnit

A qualitative description of illuminance of a surface

H.2.9. QualitativeMomementumUnit

Class IRI

http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeMomementumUnit

Super-Classes

http://qudt.org/schema/qudt#MomentumUnit
http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeUnit

A qualitative description of the momentum of an object

H.2.10. QualitativeMassUnit

Class IRI

http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeMassUnit

Super-Classes

`http://qudt.org/schema/qudt#MassUnit`
`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeUnit`

A qualitative description of the mass of an object

H.2.11. QualitativeCountUnit

Class IRI

`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeCountUnit`

Super-Classes

`http://qudt.org/schema/qudt#CountingUnit`
`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeUnit`

A qualitative description of a count

H.2.12. QualitativePermeabilityUnit

Class IRI

`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativePermeabilityUni`

Super-Classes

`http://qudt.org/schema/qudt#PermeabilityUnit`
`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeUnit`

A qualitative description of permeability

H.2.13. QualitativeForceUnit

Class IRI

`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeForceUnit`

Super-Classes

`http://qudt.org/schema/qudt#ForceUnit`
`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeUnit`

A qualitative description of a force

H.2.14. QualitativeDynamicViscosityUnit

Class IRI

`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeDynamicViscosity`

Super-Classes

`http://qudt.org/schema/qudt#DynamicViscosityUnit`
`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeUnit`

A qualitative description of dynamic viscosity

H.2.15. QualitativeVelocityUnit

Class IRI

`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeVelocityUnit`

Super-Classes

`http://qudt.org/schema/qudt#VelocityUnit`
`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeUnit`

A qualitative description of a measure of velocity

H.2.16. QualitativeRespiratoryRateUnit

Class IRI

`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeRespiratoryRate`

Super-Classes

`http://qudt.org/schema/qudt#RespiratoryRateUnit`
`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeUnit`

A qualitative description of respiratory rate

H.2.17. QualitativeTurbidityUnit

Class IRI

`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeTurbidityUnit`

Super-Classes

`http://qudt.org/schema/qudt#TurbidityUnit`
`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeUnit`

A qualitative description of turbidity

H.2.18. QualitativeConcentrationUnit

Class IRI

`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeConcentrationUni`

Super-Classes

`http://qudt.org/schema/qudt#ConcentrationUnit`
`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeUnit`

A qualitative description of the concentration of a solute in a solution

H.2.19. QualitativePlaneAngleUnit

Class IRI

`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativePlaneAngleUnit`

Super-Classes

`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeUnit`
`http://qudt.org/schema/qudt#PlaneAngleUnit`

A qualitative description of a plane angle

H.2.20. QualitativeConductanceUnit

Class IRI

`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeConductanceUnit`

Super-Classes

`http://qudt.org/schema/qudt#ConductanceUnit`
`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeUnit`

A qualitative description of the conductivity of a material or component

H.2.21. QualitativeAreaUnit

Class IRI

`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeAreaUnit`

Super-Classes
`http://qudt.org/schema/qudt#AreaUnit`
`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeUnit`

A qualitative description of the size of an area

H.2.22. QualitativeTemperatureUnit

Class IRI
`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeTemperatureUnit`

Super-Classes
`http://qudt.org/schema/qudt#TemperatureUnit`
`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeUnit`

A qualitative description of a measure of temperature

H.2.23. QualitativeWorkUnit

Class IRI
`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeWorkUnit`

Super-Classes
`http://qudt.org/schema/qudt#EnergyAndWorkUnit`
`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeUnit`

A qualitative description of an amount of energy or work

H.2.24. QualitativeResistanceUnit

Class IRI
`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeResistanceUnit`

Super-Classes
`http://qudt.org/schema/qudt#ResistanceUnit`
`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeUnit`

A qualitative description of the resitivity of a material or component

H.2.25. QualitativeLengthUnit

Class IRI

`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeLengthUnit`

Super-Classes

`http://qudt.org/schema/qudt#LengthUnit`
`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeUnit`

A qualitative description of a measure of length

H.2.26. QualitativeFrequencyUnit

Class IRI

`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeFrequencyUnit`

Super-Classes

`http://qudt.org/schema/qudt#FrequencyUnit`
`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeUnit`

A qualitative description of a measure of frequency

H.2.27. QualitativeBendingMomentUnit

Class IRI

`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeBendingMomentUni`

Super-Classes

`http://qudt.org/schema/qudt#BendingMomentOrTorqueUnit`
`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeUnit`

A qualitative description of a bending moment

H.2.28. QualitativeHeartRateUnit

Class IRI

`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeHeartRateUnit`

Super-Classes

`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeUnit`
`http://qudt.org/schema/qudt#HeartRateUnit`

A qualitative description of heart rate

H.2.29. QualitativeAccelerationUnit

Class IRI

`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeAccelerationUnit`

Super-Classes

`http://qudt.org/schema/qudt#AccelerationUnit`
`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeUnit`

A qualitative description of an acceleration

H.2.30. QualitativeLuminousIntensityUnit

Class IRI

`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeLuminousIntensit`

Super-Classes

`http://qudt.org/schema/qudt#LuminousIntensityUnit`
`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeUnit`

A qualitative description of luminous intensity of a light source

H.2.31. QualitativeVolumeUnit

Class IRI

`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeVolumeUnit`

Super-Classes

`http://qudt.org/schema/qudt#VolumeUnit`
`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeUnit`

A qualitative description of a measure of volume

H.2.32. QualitativeVideoFrameRateUnit

Class IRI

`http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeVideoFrameRateUn`

Super-Classes

    http://qudt.org/schema/qudt#VideoFrameRateUnit
    http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeUnit

A qualitative description of video frame rate

H.2.33. QualitativeThermalConductivityUnit

Class IRI

    http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeThermalConductiv

Super-Classes

    http://qudt.org/schema/qudt#ThermalConductivityUnit
    http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#QualitativeUnit

A qualitative description of a measure of thermal conductivity

H.3. Data Properties

H.3.1. qualitativeValue

Data Property IRI

    http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#qualitativeValue

The qualitative value

I. UKTemp.net Units Ontology Documentation

Ontology IRI `http://uktemp.net/ont/#`

Authors Iain Learmonth

Imported Ontologies `http://homepages.abdn.ac.uk/i.learmonth/pages/qunits/#`
`http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#`

I.1. Introduction

The UKTemp.net ontology extends well-known ontologies including the Semantic Sensor Network Ontology and the QUDT Ontology as well as two ontologies created specifically to assist with semantic web projects in the domain of integrating human and machine sensing, the Human Sensing Ontology and the Qualitative Units Ontology.

I.2. Classes

I.2.1. UKPostcodeDistrict

Class IRI
`http://uktemp.net/ont/#UKPostcodeDistrict`

Super-Classes
`http://purl.oclc.org/NET/ssnx/ssn#Platform`
`http://purl.oclc.org/NET/ssnx/ssn#FeatureOfInterest`

I.2.2. WundergroundTemperatureSensingDevice

Class IRI

`http://uktemp.net/ont/#WundergroundTemperatureSensingDevice`

Super-Classes

`http://purl.oclc.org/NET/ssnx/ssn#SensingDevice`

I.2.3. TwitterHumanTemperatureSensingDevice

Class IRI

`http://uktemp.net/ont/#TwitterHumanTemperatureSensingDevice`

Super-Classes

`http://homepages.abdn.ac.uk/i.learmonth/pages/hsense/#HumanTemperatureSensingDevi`