# Semantic Sensor Networks

Proceedings of the 5th International Workshop on Semantic Sensor Networks 2012 (SSN2012)

A workshop of the 11th International Semantic Web Conference (ISWC 2012)

Boston, Massachusetts, USA, 12 November 2012

Editors: Cory Henson, Kerry Taylor, and Oscar Corcho

**Sponsored by**

SPITFIRE EU Project Spitfire, contract no. 258885, `http://spitfire-project.eu/`

Published in CEUR-WS proceedings `http://CEUR-WS.org`

# Preface to the Proceedings of the 5th International Workshop on Semantic Sensor Networks (SSN2012)

Cory Henson[1], Kerry Taylor[2], and Oscar Corcho[3]

[1] Kno.e.sis, Wright State University, Dayton, OH USA
coryhenson@gmail.com
[2] CSIRO ICT Centre & Australian National University, Canberra, Australia
Kerry.Taylor@csiro.au
[3] Ontology Engineering Group, Universidad Politécnica de Madrid, Spain
ocorcho@fi.upm.es

Welcome to the Fifth International Workshop on Semantic Sensor Networks 2012, held in conjunction with the 11th International Semantic Web Conference, Boston, USA, 11-15 November 2012.

It is estimated that today there are 4 billion mobile devices that can act as sensors, including active and passive RFID tags. This is complemented by an even larger number of fixed sensors recording observations of a wide variety of modalities. Geographically distributed sensor nodes are capable of forming ad hoc networking topologies, with nodes expected to be dynamically inserted and removed from a network. The sensors are increasingly being connected with Web infrastructure, and the Sensor Web Enablement (SWE) standard developed by the Open Geospatial Consortium is widely being adopted in industry, government and academia alike.

While such frameworks provide some interoperability, semantics is increasingly seen as key enabler for integration of sensor data and broader Web information systems. Analytical and reasoning capabilities afforded by Semantic Web standards and technologies are considered important for developing advanced applications that go from capturing observations to recognition of events and ultimately developing comprehensive situational awareness. Defense, transportation, global enterprise, and natural resource management industries are leading the rapid emergence of applications in commercial, civic, and scientific operations that involve sensors, web, services and semantics. Semantic technologies are often proposed as important components of complex, cross-jurisdictional, heterogeneous, dynamic information systems. The needs and opportunities arising from the rapidly growing capabilities of networked sensing devices are a challenging case.

This workshop aims to provide an inter-disciplinary forum to explore and promote the technologies related to a combination of semantic web and sensor networking. Specifically, to develop an understanding of the ways semantic web technologies can contribute to the growth, application and deployment of large-scale sensor networks on the one hand, and the ways that sensor networks can contribute to the emerging semantic web, on the other.

The workshop sought paper submissions on topics including:

– Semantic support for Sensor Web Enablement
– Semantic integration in heterogeneous sensor networks
– Citizen sensors, participatory sensing and social sensing

– Semantic web services architectures for sensor networks
– Semantic algorithms for data fusion and situation awareness
– Rule-based sensor systems
– Semantic policy management in shared networks
– Semantic discovery of sensors, sensor data and services
– Semantic approaches to status monitoring and configuration of sensor systems
– Semantic sensor context management and provenance
– Semantic web in sensor data mashups
– Spatio-temporal reasoning in sensor networks
– Reasoning with incomplete or uncertain information in sensor networks
– Semantic middleware for active and passive sensor networks
– Experience in sensor network applications of semantic technologies
– Semantic reasoning for network topology management
– Ontologies for sensor and RFID networks
– Semantic feedback and control
– Emergent semantics and ambient intelligence in sensor systems
– Scalability, security, trust and privacy in semantic sensor networks
– Sensors and observations for symbol grounding

The First International Semantic Sensor Network Workshop was held with ISWC in 2006, six years ago. Since that time there has been a considerable growth in interest in the use of modern semantic technologies to address long-standing issues that seem to inhibit the widespread deployment and application of sensor technologies. In particular, the Open Geospatial Consortium has begun to consider the contribution of semantic technologies to the SWE standards. In June 2011, the W3C Semantic Sensor Networks incubator group (SSN-XG) published its final report, including a proposal for the semantic annotation of SWE standards and an ontology to describe sensor networks and facilitate annotation, which has attracted a lot of interest and is being used in semantic sensor network initiatives worldwide. Finally, a W3C Community Group[1] has been recently established to continue channelling the work that is being done in this area.

We received a total of 16 papers, all of which were carefully reviewed by at least three members of our international program committee. Only six were accepted for presentation as full papers, indicating an increasing pressure for quality in the workshop, while we also accepted other five papers as short papers and one as a demonstration paper. Four papers were rejected. Therefore, there will be twelve presentations, covering aspects related to Linked Data publication and exploitation of sensor data, several extensions and uses of the Semantic Sensor Network ontology, event processing and SPARQL extensions for dealing with sensor data, and sensor data characterisation. Furthermore, we will feature a keynote from Dave de Roure.

The chairs would like to thank our advisors (Amit Sheth and Manfred Hauswirth) and program committee. We are also grateful to our sponsor, the Spitfire project[2] funded by EU under contract 258885, which supported our best paper prize for the workshop this year.

[1] http://www.w3.org/community/ssn-cg/
[2] http://spitfire-project.eu/

We hope that you enjoy the workshop, and learn from the papers here. We appreciate your feedback on the workshop this year and hope that you can find a way to contribute in 2013.

## Program Committee

### Chairs:

| | |
|---|---|
| Cory Henson | Kno.e.sis Center, Wright State University, USA |
| Kerry Taylor | CSIRO ICT Centre, Canberra, Australia |
| Oscar Corcho | Ontology Engineering Group, Universidad Politécnica de Madrid, Spain |

### Advisors:

| | |
|---|---|
| Amit Sheth | Kno.e.sis Center, Wright State University, USA |
| Manfred Hauswirth | Digital Enterprise Research Institute, National University of Ireland, Ireland |

### Technical Program:

| | |
|---|---|
| Arun Ayyagari | The Boeing Company |
| Franz Baader | TU Dresden, Germany |
| Payam Barnaghi | University of Surrey, UK |
| Boyan Brodaric | Geological Survey of Canada |
| Michael Compton | CSIRO ICT Centre, Canberra, Australia |
| Oscar Corcho | Universidad Politécnica de Madrid, Spain |
| Peter Edwards | University of Aberdeen, UK |
| Peter Fox | RPI/Tetherless World Constellation, USA |
| Alasdair J. G. Gray | University of Manchester, UK |
| Armin Haller | CSIRO ICT Centre, Canberra, Australia |
| Krzysztof Janowicz | University of California, Santa Barbara, USA |
| Laurent Lefort | CSIRO, Australia |
| Yong Liu | NCSA/UIUC, USA |
| Thomas Meyer | Centre for Artificial Intelligence Research, UKZN and CSIR Meraka |
| Deshendran Moodley | University of KwaZulu-Natal |
| Andriy Nikolov | Knowledge Media Institute, The Open University, UK |
| Heather Packer | University of Southampton, UK |
| Kevin Page | Oxford e-Research Centre, University of Oxford, UK |
| Josiane Xavier Parreira | Digital Enterprise Research Institute, National University of Ireland, Ireland |
| Sascha Schlobinski | Cismet GmbH, Germany |
| Yanfeng Shu | CSIRO ICT Centre, Hobart, Australia |
| Ingo Simonis | International Geospatial Services Institute, Germany |
| Alan Stokes | University of Manchester, UK |

**Website**
http://knoesis.org/ssn2012/

# Table of Contents

# A Linked Sensor Data Cube for a 100 Year Homogenised daily temperature dataset

Laurent Lefort[1], Josh Bobruk[1], Armin Haller[1], Kerry Taylor[1] and Andrew Woolf[2]

[1] CSIRO ICT Centre, GPO Box 664, Canberra, Australia, {firstname.lastname}@csiro.au
[2] Australian Bureau of Meteorology, Canberra, Australia, A.Woolf@bom.gov.au

**Abstract**: The Australian Bureau of Meteorology (BOM) has recently published a homogenised daily temperature dataset, ACORN-SAT, for the monitoring of climate variability and change in Australia. The dataset employs the latest analysis techniques and takes advantage of newly digitised observational data to provide a daily temperature record over the last 100 years. In this paper, we present a case-study to publish the ACORN-SAT as Linked Data. We use the Semantic Sensor Network ontology to deliver the publicly available metadata about the BOM weather stations and their deployment history as linked data. Additionally, for concepts that are not covered by existing vocabularies, we have developed domain ontologies to define the adjusted aggregate variables and associated parameters for the ACORN-SAT homogenised observation data, the BOM weather stations and the BOM Rainfall districts. We use the RDF Data Cube Vocabulary to publish the originally released tabular time series data and structure it into slices to support multiple views and query endpoints. We further describe how these linked open vocabularies have been used and combined in the context of this project to make this dataset linkable to existing or future linked open data resources. We also discuss the versatility of the new service for the consumers of the ACORN-SAT dataset and uncover some issues which are specific to such long term climate data time series. The resulting Linked Sensor Data Cube is now accessible online via a pilot government linked data service built on the Linked Data API at lab.environment.data.gov.au.

Keywords. Ontology, Semantic Sensor Network, Data Cube, Time series.

## 1 Introduction

The Australian Climate Observations Reference Network - Surface Air Temperature (ACORN-SAT) dataset [1-3], [10], [16-17], a flagship data product of the Australian Bureau of Meteorology (BoM), has been developed for monitoring climate variability and change in Australia. To produce this dataset, climate data experts have used all the available information about weather station relocations, changes in technology and changes in observational procedures to detect breakpoints in time series and to compute adjustments for each station. The dataset provides a daily temperature record over the last 100 years. Its primary objective is to underpin better understanding of long-term climate change.

The pilot government linked data service presented in this paper provides access to the observation network metadata and to the data via an API which allows users to retrieve subsets of the published data. We have combined and extended a number of available ontologies to develop this capability, in particular the W3C Semantic Sensor Network ontology [4] and the W3C RDF Data Cube vocabulary [6]. The originally released tabular data has been transformed into a Linked Sensor Data Cube and is now accessible online as a linked data service built on top of a Linked Data API.[1]

The rest of this paper is structured as follows. In Section 2, we describe the ACORN-SAT dataset and the observation network metadata. In Section 3, we describe the role of the Semantic Sensor Network (SSN) and RDF Data Cube ontologies in the ACORN-SAT Linked Sensor Data Cube structure and in Section 4, we describe how we have built the ACORN-SAT API. In Section 5, we discuss the opportunities for climate data producers to further improve their production and publication to better match the increased demand for a transparent and reproducible data production process.

## 2 Overview of the ACORN-SAT data and metadata

The ACORN-SAT dataset originally released by the Bureau of Meteorology is available[2] as a set of tab-delimited data files which contain the homogenised minimum and maximum temperature and the raw rainfall data recorded daily at each selected site.

The temperature time series for the 112 ACORN-SAT locations are sourced from a set of single or composite stations selected according to the availability and quality of the data [17]. 10 locations have been added and one removed since the previous selection was made for the High Quality Temperature dataset [8], now superseded by ACORN-SAT. This new dataset utilises improved analysis techniques ([1], [16], [17]) which exploit pairwise comparison between one station and up to 10 comparable stations used as references. The algorithm applies differential adjustments to different parts of the daily temperature frequency distribution to better estimate the deltas with reference stations before and after an inhomogeneity. The ACORN-SAT method uses all the available background information or metadata about station moves, changes in technology and in observational procedures to identify and locate these breakpoints and to evaluate and validate the adjustments. The general knowledge used for this homogenisation process is described in several peer reviewed technical reports ([1], [16], [17]).

The site-specific knowledge is compiled in a separate station catalogue document [3] which contains the description of the 112 ACORN-SAT weather station sites. Each site is described in one page with a map and a photo of the site, the name,

[1] http://code.google.com/p/linked-data-api/
[2] http://www.bom.gov.au/climate/change/acorn-sat/

number, geographical coordinates and the locality of the station currently used plus the list of the nearest ACORN-SAT sites and some text about the site and its history. The information included in the site history is also available through a second document, which explains the numbering system used by the Bureau of Meteorology and the methods used to manage the changes of stations at each sites ([17], section 2.4 and 3.4). During each transition period, one of the sites, generally the old one, is kept as a comparison site [14] for a minimum period of five years of parallel observations. These modifications of the network structure are related to factors such as the urbanisation of the original site, in particular, the construction of new buildings affecting the quality of the observations, and the systematic transfer of bureau-staffed sites from city centres to airports. For example, the Darwin observations have been recorded at the Darwin Post Office (PO) from 1910 to 1942, and at two different sites at the Darwin Airport (AP), from 1941 to 2007 and from 2001 to now, with an overlap period of one year for the first transition and of just under six years for the second transition.

The BoM system allows for the same station code (014015) to be used for two different sites at different periods: from 1941 to 2001, it is used for a first location at the airport, which is later turned into a "comparison" station (014040) and from 2011 to now, it is used for a second location.

The six-letter station codes are "logical" codes used by the Bureau of Meteorology for the publication of observation data and station metadata for single or composite stations. In the other BoM systems, the raw data, from which the ACORN-SAT data is derived, appears as three separate time series with three different codes: 014016 for the Darwin Post Office station, 014015 for the Darwin Airport Station and 014040 for the Darwin Airport Comparison station. The current or last known location of a station with a given BoM code can be retrieved from the BoM Weather Station Directory.[3] For Darwin, we have three different physical sites: the Post Office site and two airport sites (AP1 and AP2) which are one kilometre away from each other.

Table 1 shows the sub-phases and the key relationships between them from three distinct perspectives: the physical weather stations deployed for each phase, and the BoM and respectively ACORN-SAT time series for these stations. The information included in this table is important for users wishing to compare the homogenised ACORN-SAT data with raw or adjusted data sourced from these stations.

There is not enough useable information prior to 1997 [16] about the additional "minor" moves and the changes in the instrumentation which have occurred at each site to reconstruct the full sequence of changes.

[3] http://www.bom.gov.au/climate/cdo/about/sitedata.shtml

| | PHYSICAL STATIONS | | | COMPOSITE OR SINGLE STATIONS (BOM) | | | HOMOGENEISED STATIONS (ACORN-SAT) | | |
|---|---|---|---|---|---|---|---|---|---|
| Period | Darwin PO | Darwin AP1 | Darwin AP2 | 014016 (PO) | 014015 (AP) | 014040 (AP comp) | Darwin PO | Darwin AP1 | Darwin AP2 |
| 1910-1941 | 014016 | | | PO | | | 014016 | | |
| 1941-1942 | 014016 | 014015 | | PO | AP1 | | 014016 + 014015 overlap | | |
| 1942-2001 | | 014015 | | | AP1 | | | 014015 | |
| 2001-2007 | | 014040 | 014015 | | AP2 | AP1 | | 014040 + 014015 overlap | |
| 2007-now | | | 014015 | | AP2 | | | | 014015 |

**Table 1.** History for the Darwin ACORN-SAT site

## 3 The ontologies used in the Linked Sensor Data Cube

### 3.1 Representation of the station history with the SSN ontology

In this paper we present OWL ontologies in the visual form produced by the Cmap Ontology Editor tool [9].

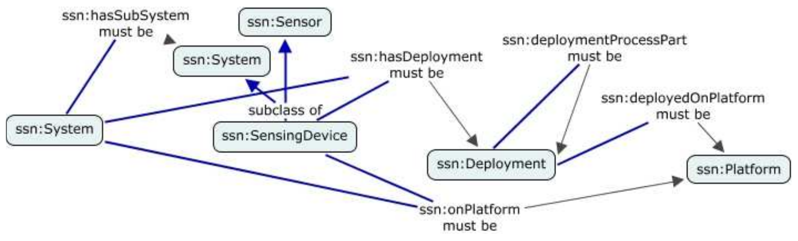


**Fig. 1.** Extract of the SSN Ontology

The Semantic Sensor Network Ontology [4] defines four classes and several relationships (Fig. 1) to model a system, its sensors, the deployment phases and deployment site (or platform).

To capture the historical sequence of changes at each ACORN-SAT site and manage the three perspectives discussed above, we define the ACORN-SAT network (Fig. 3) as a system of composite weather stations (`bom-station:System`) deployed on multiple sites (`bom-station:Station`).

The deployment sequence for each site is divided into three sub-phases with a pre-deployment at the beginning and a post-deployment at the end, to specify when two stations are used in parallel and a standalone phase for the middle period (Fig. 2).

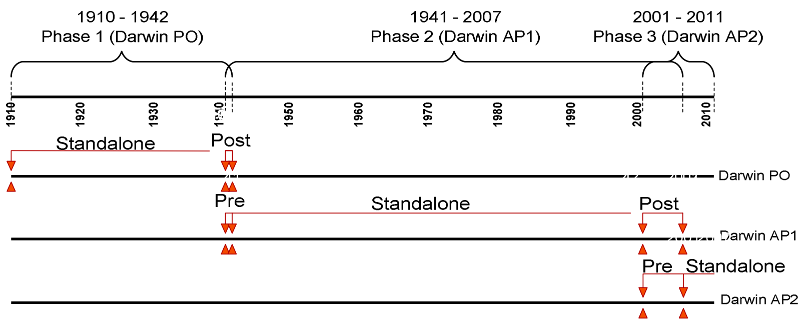


**Fig. 2.** Deployment phases and sub-phases for Darwin

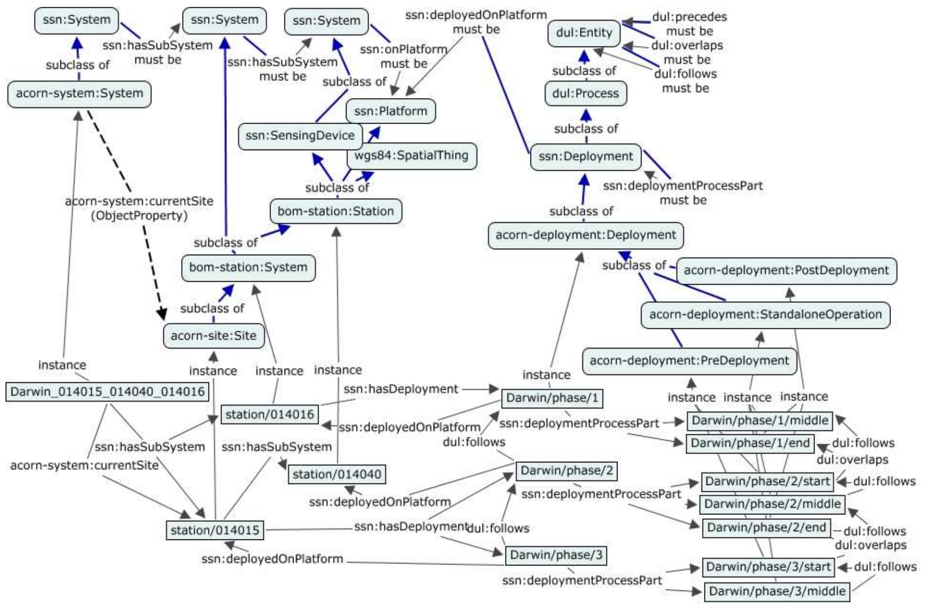


**Fig. 3.** ACORN-SAT system, sub-systems, deployment phases and sub-phases for Darwin

Using the SSN and the DOLCE Ultra Lite[4] (DUL) ontologies, we can define the ACORN-SAT deployment class and its three sub-classes for the pre-deployment,

[4] http://www.loa-cnr.it/ontologies/DUL.owl

standalone operation and post-deployment sub-phases. We reuse the `dul:follows` and `dul:overlaps` properties to specify the temporal relationships between phases and sub-phases (Fig. 3). The nature of the sub-phase is also documented through the URI scheme used for the instances of the deployment and sub-deployment classes shown in this figure.

## 3.2 Structure of the ACORN-SAT data cube

The RDF Data Cube vocabulary [6] is a vocabulary for the publication of statistical data in RDF [5], published by the W3C Government Linked Data working group.[5] Its design has been influenced by earlier efforts like SCOVO[6] and by the Statistical Data and Metadata Exchange (SDMX[7]) data model. The result is a versatile specification which can be applied to a large range of application domains.[8]

The design of the ACORN-SAT Linked Sensor Data Cube is based on the Bathing Water Linked Data pilot[9] developed by the UK Environment Agency to meet its obligations under the EU Bathing Water Directive to report weekly on the water quality measured at more than 500 sampling sites. The URI[10] and API schemes[11] developed for this project apply the URI Sets design principles defined for the UK government [7].

The ACORN-SAT data cube has four *dimensions*, one for the ACORN-SAT site and three for the date of the observation. Each *observation* contains three daily *measures*: the minimum and maximum temperature, the rainfall amount and two additional boolean *attributes* to indicate if there are missing values. Each ACORN-SAT observation refers to a date, or more precisely, to a 24 hour interval during which the maximum and minimum temperature and the amount of rainfall have been measured. For this purpose, we use the `interval:CalendarInterval` class from the UK interval ontology[12] and the URI pattern[13] for a 24 hour period starting and ending at 09:00AM.[14]

The data cube itself is divided into *slices* using the site id first, and then the year and month of the observation. All the slices are compound observations enriched with some extra statistical attributes. For the temperature measures, we pre-compute the minimum, maximum, mean, and standard deviation indicators and the count of available measures for the time series period. For the rainfall measure, we have the maximum, the sum and the count of available measures. The slice size is provided to support the estimation of data quality factors e.g. the percentage of missing data

5 http://www.w3.org/2011/gld/
6 http://vocab.deri.ie/scovo/
7 http://www.sdmx.org/
8 http://wiki.planet-data.eu/web/Datasets
9 http://environment.data.gov.uk/lab/
10 www.epimorphics.com/web/wiki/bathing-water-quality-structure-published-linked-data
11 http://environment.data.gov.uk/lab/doc/api-bwq-reference-v0.2.html
12 http://reference.data.gov.uk/def/intervals
13 http://www.epimorphics.com/web/wiki/using-interval-set-uris-statistical-data
14 Example: http://reference.data.gov.uk/id/gregorian-interval/2005-06-05T09:00:00/PT1D

points for each measure. The start and end dates of the period are encoded as `interval:CalendarInstant` instances. Fig. 4 shows the relationships between the dataset, the slices and the observations and illustrates how the URI scheme mirrors the data cube structure. Inserting keywords like station, year and month into the URI makes it easier to interpret and reduces the risk of errors by end users. In the figure we use the notation "{...}" within a URI to generically represent many URIs of the indicated pattern.

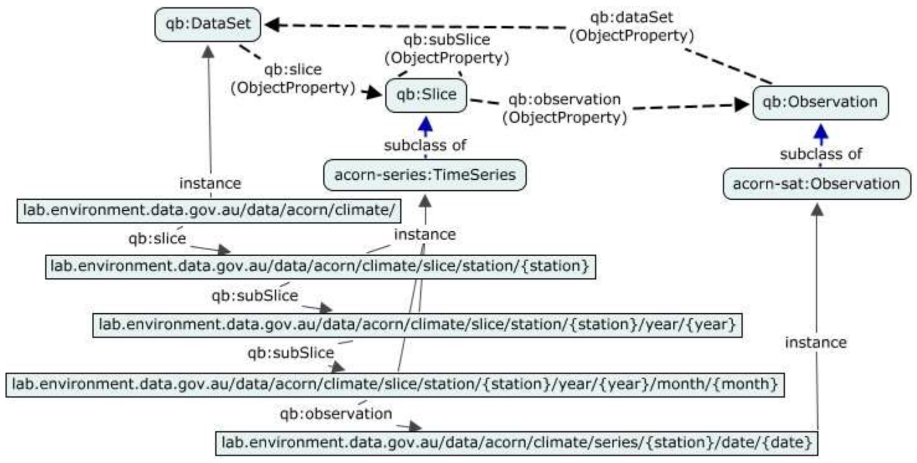

qb:DataSet
qb:dataSet (ObjectProperty)
qb:subSlice (ObjectProperty)
qb:slice (ObjectProperty)
qb:Slice
qb:observation (ObjectProperty)
qb:Observation
subclass of
acorn-series:TimeSeries
subclass of
acorn-sat:Observation
instance
lab.environment.data.gov.au/data/acorn/climate/
qb:slice
instance
lab.environment.data.gov.au/data/acorn/climate/slice/station/{station}
qb:subSlice
lab.environment.data.gov.au/data/acorn/climate/slice/station/{station}/year/{year}
qb:subSlice
instance
lab.environment.data.gov.au/data/acorn/climate/slice/station/{station}/year/{year}/month/{month}
qb:observation
lab.environment.data.gov.au/data/acorn/climate/series/{station}/date/{date}

**Fig. 4.** Linked Sensor Data Cube structure

With such a scheme, it is possible to configure the Linked Data API[15] with two types of endpoints: *item endpoints* to access the data attached to an instance and *list endpoints* to access the sub-objects (Table 2).

| REST API | LDA ENDPOINT |
|---|---|
| http://lab.environment.data.gov.au/data/acorn/climate/slice/station/014015 | Item data for station slice |
| http://lab.environment.data.gov.au/data/acorn/climate/slice/station/014015/year | List of all sub-slices |

**Table 2.** API scheme for ACORN-SAT

## 3.3 Joint use of the RDF Data Cube and the SSN ontology

The SSN ontology and the RDF Data Cube vocabulary have been developed independently of each other. Here we report how they have can be integrated together.

[15] http://elda.googlecode.com/hg/deliver-elda/src/main/docs/index.html

In the Linked Sensor Data Cube, the coupling of the two ontologies is done at two levels (Fig. 5): `ssn:Observation` and `qb:Observation` are tied together via `acorn-sat:Observation`; `acorn-sat:TimeSeries` is also defined as a `qb:Slice` and a `ssn:Observation`.

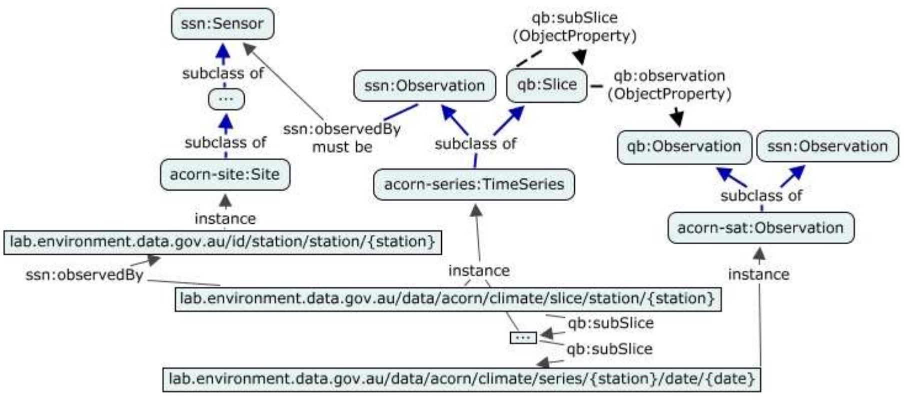

ssn:Sensor
subclass of
...
subclass of
acorn-site:Site
ssn:observedBy must be
ssn:Observation
qb:subSlice (ObjectProperty)
qb:Slice
qb:observation (ObjectProperty)
subclass of
acorn-series:TimeSeries
qb:Observation
ssn:Observation
subclass of
acorn-sat:Observation
instance
lab.environment.data.gov.au/id/station/station/{station}
ssn:observedBy
instance
instance
lab.environment.data.gov.au/data/acorn/climate/slice/station/{station}
qb:subSlice
qb:subSlice
lab.environment.data.gov.au/data/acorn/climate/series/{station}/date/{date}

**Fig. 5.** Coupling of the RDF Data Cube and SSN ontologies

The links between observations and sensors are defined for time series, but not at the level of individual data points. This approach is preferred because of the size of the ACORN-SAT dataset (~ 61 million triples). For the same reason, we have not generated the links between `qb:Observation` and `qb:DataSet` instances.

The datatype properties attached to the `acorn-sat:Observation` and `acorn-series:TimeSeries` classes are imported from two purpose-made ontologies: sat.owl (Fig. 6) and time-series.owl.

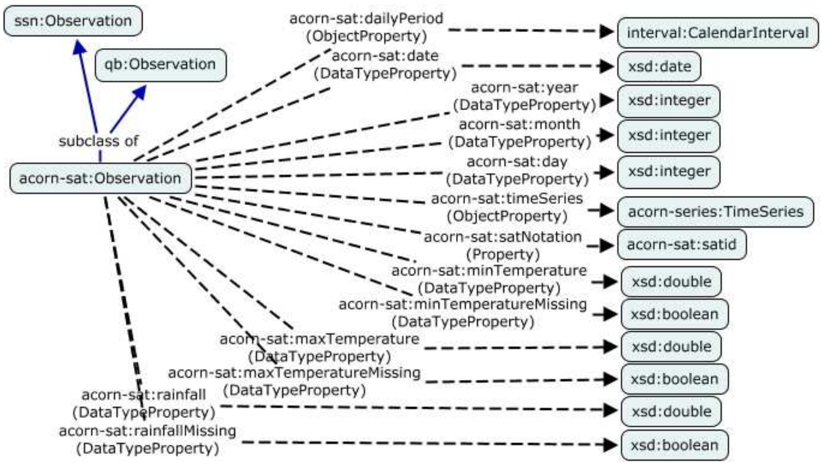
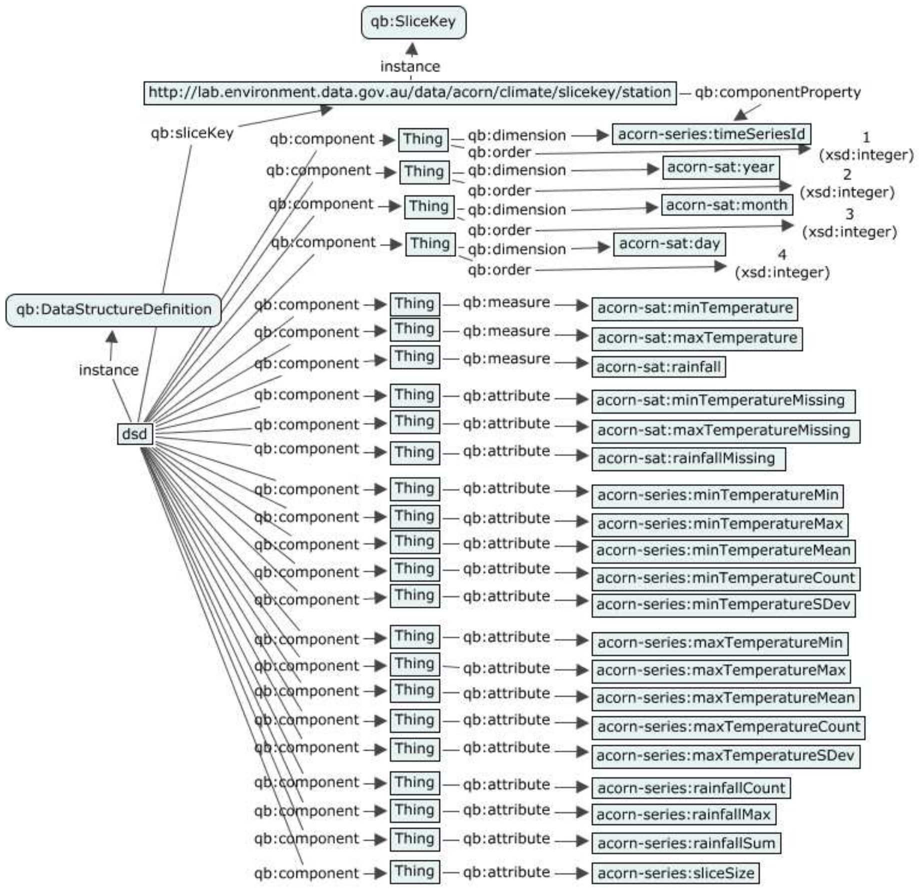

ssn:Observation
qb:Observation
subclass of
acorn-sat:Observation
acorn-sat:dailyPeriod (ObjectProperty)
interval:CalendarInterval
acorn-sat:date (DataTypeProperty)
xsd:date
acorn-sat:year (DataTypeProperty)
xsd:integer
acorn-sat:month (DataTypeProperty)
xsd:integer
acorn-sat:day (DataTypeProperty)
xsd:integer
acorn-sat:timeSeries (ObjectProperty)
acorn-series:TimeSeries
acorn-sat:satNotation (Property)
acorn-sat:satid
acorn-sat:minTemperature (DataTypeProperty)
xsd:double
acorn-sat:minTemperatureMissing (DataTypeProperty)
xsd:boolean
acorn-sat:maxTemperature (DataTypeProperty)
xsd:double
acorn-sat:maxTemperatureMissing (DataTypeProperty)
xsd:boolean
acorn-sat:rainfall (DataTypeProperty)
xsd:double
acorn-sat:rainfallMissing (DataTypeProperty)
xsd:boolean

**Fig. 6.** The Surface-Air-Temperature observation ontology (sat.owl)

Pre-existing definitions for minimum temperature and maximum temperature are not applicable for the ACORN-SAT dataset because they do not specify the boundaries of the time interval during which the daily observations are made and that they are the output of a homogenisation algorithm used to adjust the original measurements. The dotted arrows in Fig. 6 represent domain and range axioms.



qb:SliceKey
instance
http://lab.environment.data.gov.au/data/acorn/climate/slicekey/station
qb:componentProperty
qb:sliceKey
qb:DataStructureDefinition
instance
dsd
qb:component
Thing
qb:dimension
qb:order
acorn-series:timeSeriesId
1 (xsd:integer)
acorn-sat:year
2 (xsd:integer)
acorn-sat:month
3 (xsd:integer)
acorn-sat:day
4 (xsd:integer)
qb:measure
qb:attribute
acorn-sat:minTemperature
acorn-sat:maxTemperature
acorn-sat:rainfall
acorn-sat:minTemperatureMissing
acorn-sat:maxTemperatureMissing
acorn-sat:rainfallMissing
acorn-series:minTemperatureMin
acorn-series:minTemperatureMax
acorn-series:minTemperatureMean
acorn-series:minTemperatureCount
acorn-series:minTemperatureSDev
acorn-series:maxTemperatureMin
acorn-series:maxTemperatureMax
acorn-series:maxTemperatureMean
acorn-series:maxTemperatureCount
acorn-series:maxTemperatureSDev
acorn-series:rainfallCount
acorn-series:rainfallMax
acorn-series:rainfallSum
acorn-series:sliceSize

**Fig. 7.** The Linked Sensor Data Cube DSD (Data Structure Definition) - extract

The Expert Team on Climate Change Detection and Indices [17] has specified the statistical attributes which can be attached to monthly or annual slices. We have transformed these definitions into an ontology[16] and used it to characterise the acorn-series properties when possible.

These properties are (respectively) used in the observations and time series instances and are also registered in the Data Structure Definition (DSD) instance

[16] http://purl.oclc.org/NET/ssnx/etccdi

which documents the dimensions, measures, and attributes of the Linked Sensor Data Cube (Fig. 7).

We have found that the declarations of the observed properties in the SSN ontologies as classes are not directly compatible with their declarations in the RDF Data Cube as properties. Further investigation is needed to define a common ontology design pattern to bridge the two ontologies. The registration of the properties in the DSD is also an aspect of the RDF Data Cube vocabulary which the W3C Government Linked Data working group has identified as an area for further work.

## 4 The ACORN-SAT Linked Sensor Data service

The ACORN-SAT Linked Sensor Data service[17] uses open source software such as the SESAME triple store[18] and the ELDA[19] implementation of the Linked Data API. The architecture of our solution is shown in Fig. 8.

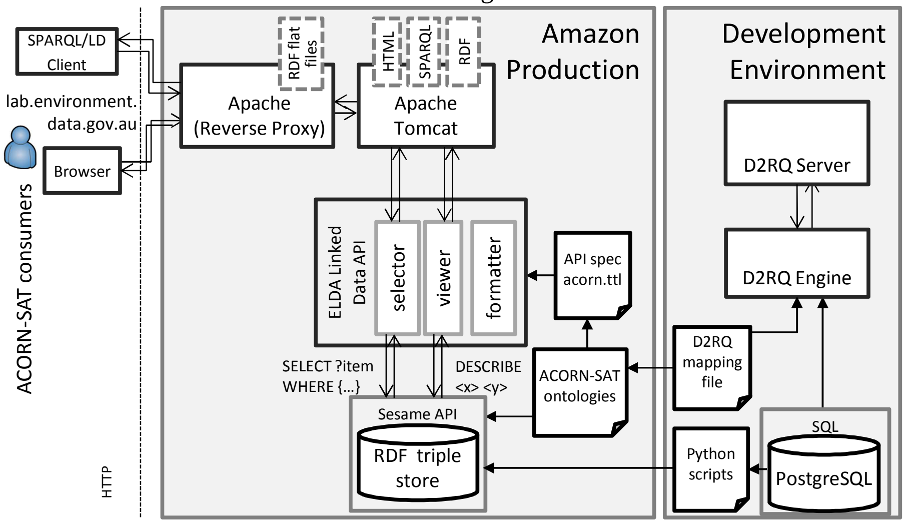


**Fig. 8.** Architecture of lab.environment.data.gov.au

We mapped the tabular time series data of the original ACORN-SAT to RDF using D2RQ and custom-built Python scripts. The mapping file for D2RQ is configured to produce RDF data according to the ACORN-SAT ontologies (see above). We used Jena EyeBall[20] to validate the linked data because of the dependencies on several RDF(S) vocabularies. Since ACORN-SAT is a relatively static data set (we expect no more than one update per year), the generation and validation tools are deployed outside of the production environment.

[17] Accessible via http://lab.environment.data.gov.au/
[18] http://www.openrdf.org/
[19] http://elda.googlecode.com/
[20] http://jena.apache.org/documentation/tools/eyeball-getting-started.html

For the configuration of the Linked Data API we exploited the URI and API scheme introduced in section 3.1 to define all the item and list endpoints which give access to all the individual observations (Fig. 9, Fig. 10) and slices (Fig. 11, Fig. 12) but also to the site metadata for each time series (Fig. 13). The site data is also enriched with additional information from the Weather Station Directory[21] in particular the rainfall district[22] and state.

Due to the size of ACORN-SAT (~61million triples) we have put particular focus on the usability (performance) of the Linked Data API. We defined custom viewers for the different list endpoints in order to avoid expensive DESCRIBE SPARQL queries. Our production environment serving lab.environment.data.gov.au runs on an Amazon cloud with ELDA scaling horizontally at peak demand. Since our data is static, we only replicate ELDA and its cache, but access only a single Sesame triplestore instance.

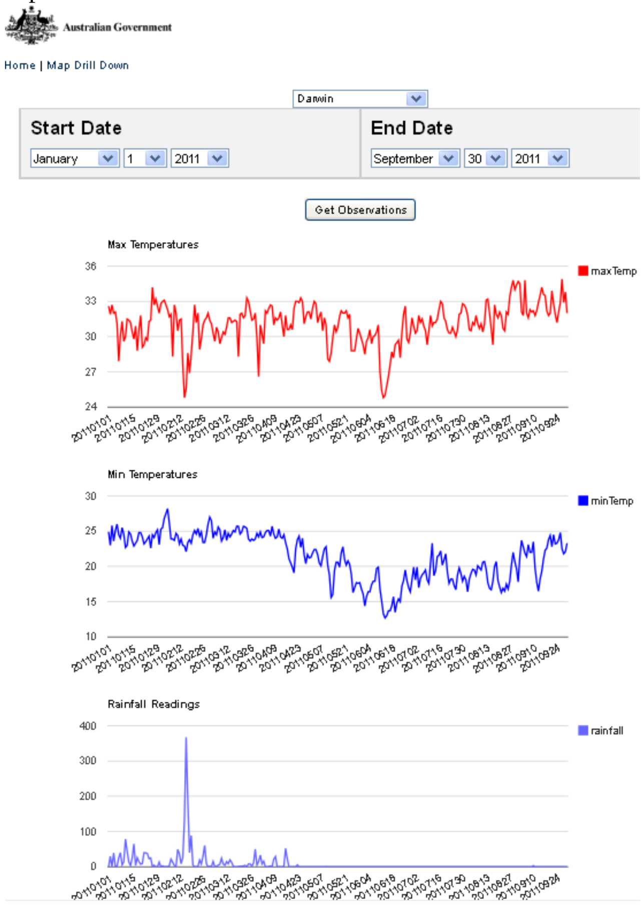
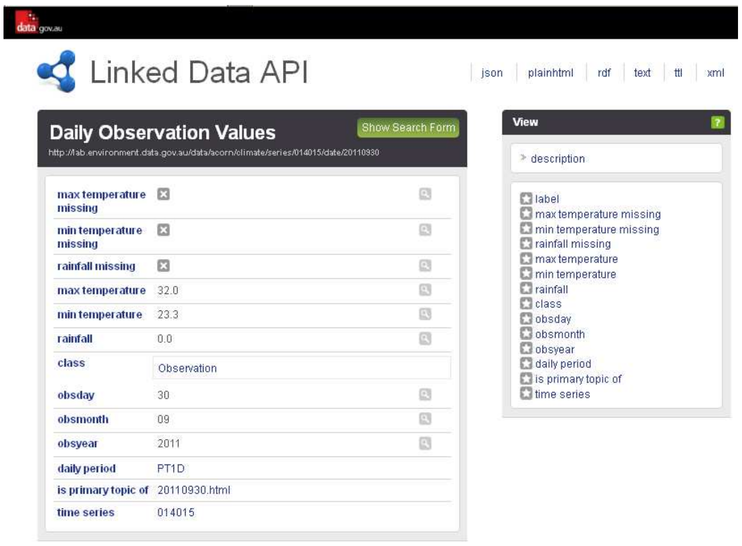


**Fig. 9.** Access to observation data (Darwin, January-September 2011)

[21] ftp://ftp.bom.gov.au/anon2/home/ncc/metadata/sitelists/stations.zip
[22] http://www.bom.gov.au/climate/cdo/about/rain-districts.shtml



Daily Observation Values

http://lab.environment.data.gov.au/data/acorn/climate/series/014015/date/20110930

| | |
|---|---|
| max temperature missing | ☒ |
| min temperature missing | ☒ |
| rainfall missing | ☒ |
| max temperature | 32.0 |
| min temperature | 23.3 |
| rainfall | 0.0 |
| class | Observation |
| obsday | 30 |
| obsmonth | 09 |
| obsyear | 2011 |
| daily period | PT1D |
| is primary topic of | 20110930.html |
| time series | 014015 |

View

- description

- label
- max temperature missing
- min temperature missing
- rainfall missing
- max temperature
- min temperature
- rainfall
- class
- obsday
- obsmonth
- obsyear
- daily period
- is primary topic of
- time series

**Fig. 10.** Access to observation record via the API (Darwin, 30 September 2011)

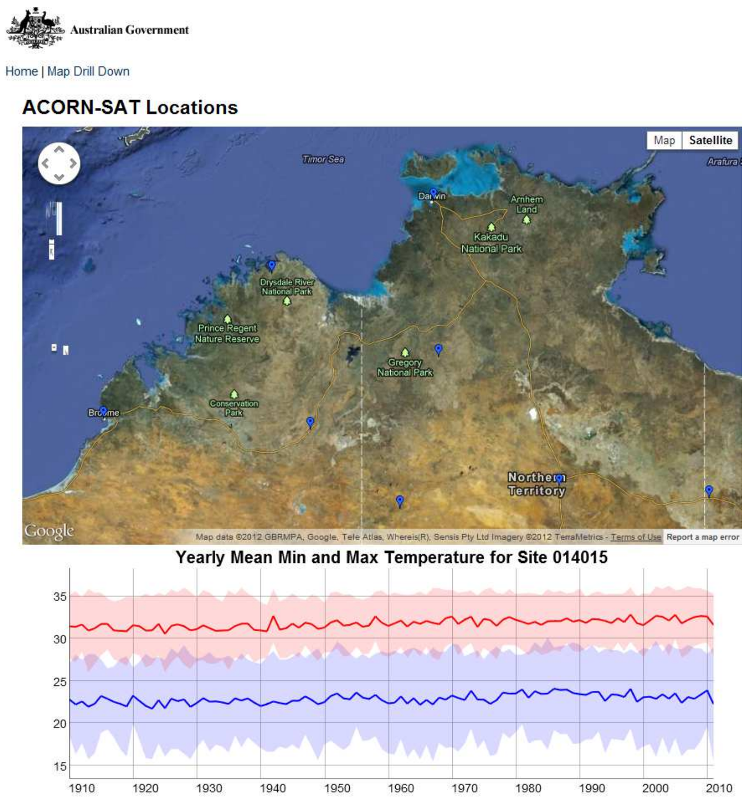
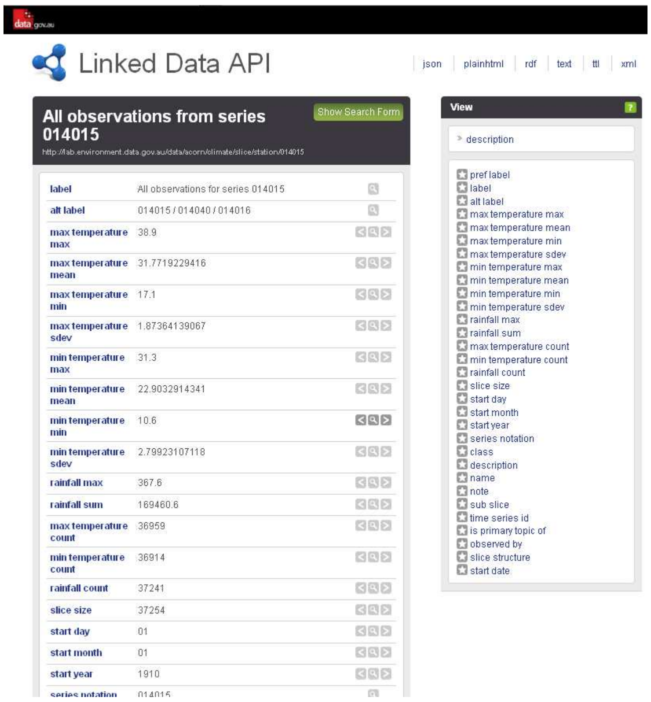


**Fig. 11.** 100 year min-max temperature time series data (Darwin)


Linked Data API

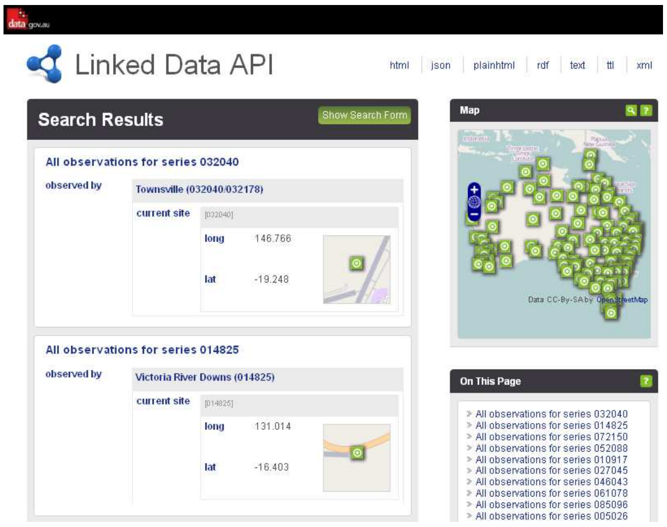
json | plainhtml | rdf | text | ttl | xml

**All observations from series 014015**

http://lab.environment.data.gov.au/data/acorn/climate/slice/station/014015

| | |
|---|---|
| label | All observations for series 014015 |
| alt label | 014015 / 014040 / 014016 |
| max temperature max | 38.9 |
| max temperature mean | 31.7719229416 |
| max temperature min | 17.1 |
| max temperature sdev | 1.87364139067 |
| min temperature max | 31.3 |
| min temperature mean | 22.9032914341 |
| min temperature min | 10.6 |
| min temperature sdev | 2.79923107118 |
| rainfall max | 367.6 |
| rainfall sum | 169460.6 |
| max temperature count | 36959 |
| min temperature count | 36914 |
| rainfall count | 37241 |
| slice size | 37254 |
| start day | 01 |
| start month | 01 |
| start year | 1910 |
| series notation | 014015 |

**Fig. 12.** Access to Darwin 100-year data via the API


Linked Data API

html | json | plainhtml | rdf | text | ttl | xml

**Search Results**

All observations for series 032040

observed by: Townsville (032040/032178)

current site: [032040]

| | |
|---|---|
| long | 146.766 |
| lat | -19.248 |

All observations for series 014825

observed by: Victoria River Downs (014825)

current site: [014825]

| | |
|---|---|
| long | 131.014 |
| lat | -16.403 |

On This Page

- All observations for series 032040
- All observations for series 014825
- All observations for series 072150
- All observations for series 052088
- All observations for series 010917
- All observations for series 027045
- All observations for series 046043
- All observations for series 061078
- All observations for series 085096
- All observations for series 005026

**Fig. 13.** Access to station metadata via the API (list endpoint)

The common foundations of the URI and API schemes (see Fig. 3 and Table 2 above) and the ability to browse the data are important features for new users wanting to rapidly build on the ACORN-SAT data.

## 5 Discussion

We have uncovered multiple challenges during this first attempt to apply the SSN ontology and the RDF Data Cube vocabulary to long term climate data time series. The first challenge is the volume and diversity of metadata which needs to be captured beyond the transitions and overlaps between the deployment phases discussed above. The changes in the sensor locations, technologies and observation procedures ([2], [11]) are not the only categories of events which are recorded in the Station Catalogue document [3]. For example, the construction of buildings and the growth of vegetation close to a weather station are also noted because they can have a significant impact on the quality of the observations. There are also more specific inhomogeneities discussed in other publications [17] e.g. the creation of a substantial artificial lake in Canberra (Lake Burley Griffin) about 4 km west of the observation site. Approximately half of the adjustments done on the ACORN-SAT minimum and maximum temperature values [16] are supported by metadata records of which 80% were linked to station moves. With the help of the SSN ontology, we have captured the major moves which, in Australia, have occurred mainly in the 1940s and the 1990s when the sites of observations were transferred from town centres to airports. For long-term time series of extremes measurements such as ACORN-SAT, the change of observation times are also critical. In Australia, before 1964, about 30% of the weather stations used a 0000-0000 day. In 1964, the BoM switched to the current 0900–0900 standard. We have found that the available MET ontologies define the minimum and maximum temperature but not the limits of the 24 hour period used to record them.

The second challenge is to answer the public demand to have a more transparent homogeneisation process. The ACORN-SAT peer review [10] states that “a list of adjustments made as a result of the process of homogenisation should be assembled, maintained and made publicly available, along with the adjusted temperature series. Such a list will need to include the rationale for each adjustment”. We have not investigated if it would be beneficial to use the W3C PROV-O[23] ontology or to add new slices to our current data cube structure for this purpose. We know that more ontology work is needed for the definition of breakpoints, of the data structures used by percentile-based transfer functions and the parameters used by these algorithms [18].

The third challenge is to help the climate science community to create consistent and comprehensive climate data resources ([13], [15]). We have learned that the dependencies between the datasets published by national and international organisations are hard to document, partly because the numbering scheme can be a source of confusion. One possible approach is to extend the VoID[24] vocabulary to manage these relationships at the level of the slices of a RDF Data Cube rather than at the level of the data cube itself.

[23] PROV-O: The PROV Ontology http://www.w3.org/TR/prov-o/

[24] Vocabulary of Interlinked Datasets (VoID) http://vocab.deri.ie/void/

The fourth challenge is to link the ACORN-SAT data to other datasets. We provide two main ways of how other data sets can link to ACORN-SAT. Like Patni et al [12], we have linked the BoM stations to their associated GeoNames[25] features. We also provide temporal slices for each year of observation and consequently, we have 100 temporal slices that could be linked from other temporal data sets.

## 6 Conclusion

The publication of this data set represents a milestone in e-government in Australia—it is the first linked data published by the Australian Government's open data sharing initiative known as data.gov.au.

We believe that the explicit support for metadata attachment as offered by the linked data approach presented here is of ongoing importance to the publication of climate data, and may help to enrich the public debate about the scientific foundations for climate science.

By coupling the SSN ontology and the RDF Data Cube vocabulary, we are able to capture the station history and to attach it to the data at the right level of temporal and spatial granularity. The well designed URI and API scheme makes the data cube structure easier to understand and navigate by all both producers and consumers of the linked data.

## Acknowledgments

This project was funded by the Australian Bureau of Meteorology under the National Plan for Environmental Information initiative. Part of this research was conducted as part of the CSIRO Water for a Healthy Country National Research Flagship and the Sensor and Sensor Network Transformation Capability Platform.

Thanks to Alexander Coley (UK Environment Agency), Dave Reynolds, Stuart Williams, Ian Dickinson (Epimorphics) for the shared material on the UK Bathing Water project.

Thanks also to Benedikt Kämpgen (AIFB, Karlsruhe Institute of Technology and W3C Government Linked Data working group) for the suggestion to publish sensor data as statistics[26] and to couple the RDF Data Cube and the SSN ontologies.

## 7 References

1. Australian Bureau of Meteorology: Report 3a - ACORN-SAT analysis and results document (2011).
2. Australian Bureau of Meteorology: Report 4 - ACORN-SAT surface air temperature observing methods document (2011).

[25] http://www.geonames.org/
[26] http://www.w3.org/2011/gld/wiki/Data_Cube_Vocabulary/Use_Cases

3. Australian Bureau of Meteorology: Report 5 - ACORN-SAT station catalogue (2011).
4. Compton, M., Barnaghi, P., Bermudez, L., Garcia-Castro, R., Corcho, O., Cox, S., Graybeal, J., Hauswirth, M., Henson, C., Herzog, A., Huang, V., Janowicz, K., Kelsey, W. D., Phuoc, D. L., Lefort, L., Leggieri, M., Neuhaus, H., Nikolov, A., Page, K., Passant, A., Sheth, A., Taylor, K.: The SSN ontology of the W3C semantic sensor network incubator group, Journal of Web Semantics. Elsevier (2012).
5. Cyganiak, R., Hausenblas, M., McCuirc, E.: Official Statistics and the Practice of Data Fidelity. In: Linking Government Data, Wood, D. (Ed.), pp. 135-151. Springer (2011).
6. Cyganiak, R., Reynolds, D., Tennison, J.: The RDF Data Cube Vocabulary, W3C Working Draft 05 April 2012. W3C (2012).
7. Davidson, P.: Designing URI Sets for the UK Public Sector. UK Chief Technology Officer Council (2010).
8. Della-Marta, P., Collins, D., Braganza, K.: Updating Australia's high-quality annual temperature dataset. Aust. Met. Mag. 53(2), pp. 75-93. Australian Bureau of Meteorology (2004).
9. Eskridge, T., Hayes, P., Hoffman, R., Warren, M.: Formalizing the informal: A Confluence of Concept Mapping and the Semantic Web. In: Proc. of the 2nd Int. Conf. on Concept Mapping, Vol. 1, pp. 247-254 (2006).
10. Matthews, K., Peterson, T., Wang, X., Wratt, D.: ACORN-SAT Report of the Independent Peer Review Panel, Australian Bureau of Meteorology (2011).
11. Nicholls, N., Tapp, R., Burrows, K., Richards, D.: Historical Thermometer Exposures in Australia. Int. J. Climatol., 16, pp. 705–710. John Wiley & Sons (1996).
12. Patni, H., Henson, C., Sheth, A.: Linked Sensor Data 2010 In: Proc. Of the Int. Symp. on Collaborative Technologies and Systems (CTS 2010), pp. 362-370. IEEE (2010).
13. Peterson, T. C., Manton, M. J.: Monitoring Changes in Climate Extremes: A Tale of International Collaboration, Bull. Amer. Meteor. Soc. 89, pp. 1266-1271. American Meteorological Society (2008).
14. Street, R. B., Allsopp, D., Durocher, Y.: Guidelines for Managing Changes in Climate Observation Programmes WCDMP-No. 62. World Meteorological Organization (2007).
15. Thorne, P. W., Willett, K. M., Allan, R. J., Bojinski, S., Christy, J. R., Fox, N., Gilbert, S., Jolliffe, I., Kennedy, J. J., Kent, E., Tank, A. K., Lawrimore, J., Parker, D. E., Rayner, N., Simmons, A., Song, L., Stott, P. A., Trewin, B.: Guiding the Creation of A Comprehensive Surface Temperature Resource for Twenty-First-Century. Climate Science, Bull. Amer. Meteor. Soc., 92, ES40-ES47. American Meteorological Society (2011).
16. Trewin, B.: A daily homogenized temperature dataset for Australia. Int. J. Climatol. (2012).
17. Trewin, B.: Techniques involved in developing the Australian Climate Observations Reference Network Surface Air Temperature (ACORN-SAT) dataset. Technical Report No. 049. Centre for Australian Weather and Climate Research (2012).
18. Williams, C. N., M. J. Menne, Thorne, P. W.: Benchmarking the performance of pairwise homogenization of surface temperatures in the United States, J. Geophys. Res., 117, D05116 (2012).
19. Zhang, X., Alexander, L., Hegerl, G. C., Jones, P., Tank, A. K., Peterson, T. C., Trewin, B., Zwiers, F. W.: Indices for monitoring changes in extremes based on daily temperature and precipitation data, WIREs Clim Change, 2, pp. 851–870. John Wiley & Sons (2011).

# A logic-based CoAP extension for resource discovery in semantic sensor networks

Michele Ruta, Floriano Scioscia, Giuseppe Loseto, Filippo Gramegna, Agnese Pinto, Saverio Ieva, Eugenio Di Sciascio

Politecnico di Bari
via Re David 200, I-70125
Bari, ITALY
{m.ruta,f.scioscia,agnese.pinto,disciascio}@poliba.it,
{loseto,gramegna,ieva}@deemail.poliba.it

**Abstract.** Main restraints curbing a deep integration of Semantic Sensor Networks with complex and articulated architectures, basically reside in too elementary allowed discovery capabilities. Several studies agree advanced querying and retrieval mechanisms are needed to truly fulfill the potential of the SSN paradigm. This paper presents a novel SSN framework, supporting a resource discovery grounded on semantic-based matchmaking. Offered contributions are: a backward-compatible extension of Constrained Application Protocol (CoAP) resource discovery; data mining exploitation to detect high-level events from raw data; employment of non-standard inference services for retrieving and ranking resources; adoption of W3C standard SSN-XG ontology to annotate data, events and device features. The effectiveness of the proposed approach is motivated by a case study regarding fire risk prevention and air conditioning control in a university building.

**Keywords:** Semantic Sensor Networks, CoAP, Resource discovery, Matchmaking, Data mining

## 1 Introduction

Typically, Wireless Sensor Networks (WSNs) only include homogeneous sensors and are application-dependent and engineering-oriented [1]. This is a strong limit in terms of interoperability and in sight of the integration in large-scale complex architectures. In the mid-2000s semantic technologies were acknowledged as a mean to overcome these issues: in Semantic Sensor Networks (SSNs), *"semantics refers to the critical meaning of sensory data, sensor nodes and application requirements"*, *"effectively enabling the integration, exchange, and reuse of sensory data across various applications and multiple sensor nets"* [1]. Semantics can be leveraged in several aspects: sensory data [1], device management [2], data dissemination and routing [1, 3], query processing [4], application-level services [5].

In latest years, main research studies have shifted toward the integration of SSNs with the Internet and World Wide Web, following the Internet of Things and (Semantic) Web of Things visions [6]. Semantics is devoted to increase both autonomicity and integrability, in order to truly fulfill the potential of the SSN paradigm through enhanced object/subject/event annotation enabling advanced applications. Recent proposals for standard protocols in the field of object networks are gaining acceptance, such as 6LoWPAN and the Constrained Application Protocol (CoAP). Nevertheless, current solutions only allow coarse data-oriented representation and querying mechanisms and still require a significant human intervention for design, deployment and integration of new applications from elementary building blocks, with very similar issues to those affecting Web mashups [7].

In this paper a novel SSN framework is proposed, enabling semantic-based annotation and discovery, by adapting formalisms and technologies borrowed from Semantic Web and Internet of Things research. Data streams, sensors and actuator devices, objects and subjects, high-level events and services can be connoted by a description having a well defined meaning w.r.t. a shared domain conceptualization (*i.e.*, ontology). The proposal is based on slight backward-compatible extensions to CoAP and CoRE Link Format[1] resource discovery protocol for sensor and actor networks. A simple and computationally efficient data mining component permits to detect and annotate high-level events from raw data collected from sensing devices in the field. The SSN-XG ontology of W3C (World Wide Web Consortium) [8] is adopted as reference vocabulary for resource annotations. Non-standard inference services for semantic-based matchmaking [9] allow to retrieve and rank the best matching resources w.r.t. a given request, supporting not only full matches but also approximate ones.

The remainder of the paper is organized as follows. In Section 2 relevant related work is briefly surveyed. The proposed discovery framework is thoroughly outlined in Section 3, also providing details about CoAP extensions and event mining. Section 4 reports on a case study about fire risk prevention and air conditioning control in a university building, in order to highlight features of the proposed approach; finally Section 5 closes the paper.

## 2 Related work

SSN frameworks are based on reference ontologies to annotate data, devices and services. Many ontology proposals exist, largely varying in aim and scope. OntoSensor [10] and SSN-XG [8] are among the most relevant and widely used ones. Several approaches for publishing sensor data along the Linked Open Data [11] guidelines or via RESTful[2] interfaces are now diffused [12, 7, 13], even as commercial solutions (*e.g.*, Pachube-Cosm, `https://cosm.com/`).

[1] CoRE Link Format, IETF CoRE Working Group Internet-Draft, latest version: 14, 1 June 2012, `http://www.ietf.org/id/draft-ietf-core-link-format-14.txt`

[2] REST: REpresentational State Transfer

From the infrastructure perspective, some interesting protocols are assuming relevance for their lightweight impact on storage and computation. Particularly, CoAP [14] is an alternative to HTTP for interconnected objects, exploiting a binary data representation and a subset of HTTP methods (GET, PUT, POST, DELETE). It follows the REST paradigm for making data and resources accessible. Both 6LoWPAN and CoAP use UDP for transport, as TCP is considered too resource-consuming. 6LoWPAN can be interfaced to IPv6 and CoAP/UDP to HTTP/TCP, so that sensor data can be accessed from the Web.

Recent projects, such as Sense2Web [15] and SPITFIRE [16], combine semantic and networking technologies to build full frameworks, but only allow elementary queries in SPARQL[3] fragments on RDF[4] annotations. More effective techniques such as ontology-based *complex event processing* [17] and *semantic matchmaking* [9] –exploiting logic-based reasoning to support approximated matches, resource ranking and explanation of outcomes– can be used to manage sensory data and events in mobile and pervasive contexts [18–21]. The main issue deriving from the integration of semantics in standard wireless protocols can be inherited by studies in the field of mobile and pervasive computing: *e.g.*, for Bluetooth [18], EPCglobal RFID [21] and ZigBee [20].

Finally, a not negligible related issue is the verbosity of XML-based ontological languages, which calls for devising and implementing efficient compression techniques needed to cope with WSN scenarios, characterized by low throughput of wireless links and strict processing, memory and energy limitations of devices. When evaluating encoding algorithms for such contexts, compression ratio is not enough: processing/memory requirements and efficiency of queries on compressed data become critical parameters, too. High-compression general-purpose approaches such as the PAQ family [22] or XML-specific ones like XMill [23] do not offer the best trade-off. The *EXI* W3C standard (Efficient XML Interchange, http://www.w3.org/XML/EXI/) exploits some basic ideas of XMill, while adding several optimizations. Specific experimental algorithms for the Semantic Web of Things, such as *DIGcompressor* and *COX* [24], aim at either maximum compression ratio or high query efficiency, while keeping the computational costs low.

## 3 Framework and approach

In what follows the approach we present will be thoroughly described, particularly detailing the proposed CoAP enhancements as well as the semantic matchmaking framework adopted for event mining and resource discovery.

[3] SPARQL Query Language for RDF, W3C Recommendation 15 January 2008, http://www.w3.org/TR/rdf-sparql-query/

[4] Resource Description Framework, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/

## 3.1 CoAP basics

Following the REST architectural style, CoAP adopts a loosely coupled client/server model, based on stateless operations on *resource* representations [14]. Every resource is a server-controlled abstraction, unambiguously identified by a URI (Uniform Resource Identifier). Clients access resources via synchronous request/response interactions, using HTTP-derived methods `GET`, `PUT`, `POST`, and `DELETE` (basically mapping the fundamental Read, Create, Update and Delete operations of data management).

CoAP messages are encoded in a simple binary format. A message consists of a 32-bit *header* followed by *option* fields in Type-Length-Value (TLV) format and a *payload*. The header determines the request method (or response status) and the number of options. Possible request methods, option types and response statuses are distinguished by means of binary codes, listed in CoAP specification[5]. Some CoAP options are derived from HTTP header fields (*e.g.*, content type, headers for conditional requests and proxy support), while some other ones have no counterpart in HTTP. In particular, the target resource URI for a CoAP request (which must refer to the `coap` or `coaps` scheme) is split in a `Uri-Host`, a `Uri-Port` (default UDP port is 5683) and a `Uri-Path` option, with one `Uri-Query` option for each field in the query URI portion. If an option value is longer than 270 B, it is split in consecutive option fields of the same type. Moreover, the `Observe` option[6] allows a client to register w.r.t. the server as an *observer* of the resource, so that the server will notify the client of further changes to the resource state automatically, without the need of polling. This capability is lacking in HTTP; it was introduced in CoAP specifically for scenarios like WSNs, where data have to be monitored over a time span.

In CoAP-based WSN scenarios, each sensor is basically seen as a server, exposing both sensor readings and internal information as resources toward clients, which act on behalf of end-user applications. Different data series will be identified by distinct URIs. Further URIs will identify sensor device status and operating parameters; clients will be able to read or modify them as appropriate. For example, a temperature sensor $S$ can expose the latest temperature reading at the URI `coap://[S-address]/temperature`; in order to access it, a client should issue a `GET` request with `Uri-Host=S1-address` and `Uri-Path=/temperature` options. In case of success, it will receive status code `2.05` and the response message payload will contain the value, *e.g.*, `22.5°C` if it is returned as plain text. Furthermore, since CoAP supports proxies, cluster-head or sink nodes can reply on behalf of a set of (possibly more constrained) sensor nodes deployed in an area, exploiting caching and decreasing the load at the edge of the network. This feature allows also the adoption of data fusion and mining techniques at vari-

[5] Constrained Application Protocol, IETF CoRE Working Group Internet-Draft, latest version: 11, 16 July 2012, `http://www.ietf.org/id/draft-ietf-core-coap-11.txt`

[6] Documented in: Observing Resources in CoAP, IETF CoRE Working Group Internet-Draft, latest version: 5, 12 March 2012, `http://tools.ietf.org/id/draft-ietf-core-observe-05.txt`

ous levels along the path from sensors in the field to nodes managing high-level application logic.

Resource discovery is needed to know what resources a given CoAP server is making available. The CoAP discovery protocol is defined in the CoRE Link Format specification. It allows any host to expose its resources, as well as to act as a directory service for other hosts that want to register their resources. A client will access the reserved URI path `/.well-known/core` on the server with `POST` method to register a resource, or with `GET` to discover available ones. `GET` requests can include `URI-query` fields to retrieve only resources with specific attributes. Standardized query attributes include:

– `href` (hypertext reference): a regular expression to filter resources based on their path (*e.g.*, "/temperature" or "/temperature/*");
– `type` (media type): MIME (Multipurpose Internet Mail Extensions) type/-subtype for the resource;
– `rt` (resource type): an opaque string representing an application-specific meaning of a resource (*e.g.*, "outdoor-temperature");
– `if` (interface description): an opaque string used to provide a name or a URI which indicates what operations can be performed on the resource and their meaning; it typically references a machine-readable document.

Further non-reserved attributes can be freely used. Response payload consists of a comma-separated list of resource paths, each having optionally a list of semicolon-separated attributes.

## 3.2 Fundamentals of Semantic-based matchmaking

As evident, CoAP resource discovery protocol only allows a syntactic string-matching of attributes, lacking every explicit and formal characterization of the resources semantics. In [9], framework and algorithms were proposed for a logic-based matchmaking between a request and one or more resource descriptions, both expressed using languages grounded on a well known logic. Also a ranking of resource annotations w.r.t. the original request was made possible according on the meaning of descriptions with reference to a shared conceptualization, *i.e.*, an ontology. Description Logics (DL) [25] was the reference formalism and particularly the $\mathcal{ALN}$ (Attributive Language with Unqualified Number Restrictions) DL subset was used, which has polynomial computational complexity for standard and non-standard inferences described hereafter. Given a request $Q$ and a resource $R$, both consistent w.r.t. a common ontology $\mathcal{T}$ (containing axioms that model knowledge for the reference problem domain), *concept subsumption* [25] standard inference service can be used to identify *full matches*, *i.e.*, resources providing all features requested in $Q$. Unfortunately, such correspondences are infrequent in practical scenarios involving heterogeneous resources and articulate descriptions. Whenever $R$ is not a full match for $Q$, *Concept Abduction Problem* (CAP) [9] non-standard inference service allows to determine what should be hypothesized in $R$ in order to completely satisfy $Q$. The solution $H$ (for *Hypothesis*) to CAP represents "why" the subsumption relation $\mathcal{T} \models R \sqsubseteq Q$ does not hold. $H$ can be interpreted as *what is requested in $Q$ and not specified in $R$*. Previous inference services were implemented via structural algorithms based on *Conjunctive Normal Form* (CNF) normalization of concept expressions [26]. Since a concept CNF is unique, a *semantic distance* can be associated to every $(Q, R)$ pair, based on the "size" of the respective CAP solution $H$. This enables a logic-based relevance ranking of a set of available resources w.r.t. a given request [26].

## 3.3 Data mining techniques for event annotation

Sensor Networks basically produce large amounts of raw data which have to be collected and interpreted to extract application-oriented information. This is particularly relevant in case of event detection where (semi) automatic procedures are strongly required. The proposed framework includes a simple and resource-efficient data mining process, distributed along several steps and aimed at a semantic-based event annotation from low-level data audit. Each sink device in the SSN collects data from sensors in the field and analyzes them. Whenever an event is detected, a dedicated CoAP resource record is updated by adding a semantic description. The record will also contain extra-logical context parameters such as a timestamp and geographic information about the monitored area (coordinates and size). After detection, the sink will serve as a CoAP gateway, waiting for resource discovery requests coming from client applications searching for either: (a) sensors needed to detect events in the area or (b) actuators that can effectively act upon a given detected event.

As said, the procedure for identification of sensory events via surveys comprises several stages.

**1.** Data are read from sensors in the field through standard CoAP GET requests, possibly using `Observe` option to be notified of updates. Then a list of elements is built, consisting of three fields: *ID*, storing the identifier of the sensor and therefore the type of data; *value*, where the detected data is stored; *timestamp*. This list will group measurements in time slots of application-defined period $T$, which are used to compute statistical indexes.

**2.** For each data set, the average, variance and standard deviation values are computed for the current time slot, to assess the variability of collected information within the monitored area.

**3.** Statistical indexes of elapsed periods are then exploited to compute an incremental ratio able to evidence trends and significant event changes inside the monitored area.

**4.** For every data collection, the application defines a binary or multiple classifier, to reveal a situation when given conditions occur. In fact identification is performed by taking into account threshold values for statistical indexes (see Table 2 in the case study section for an extensive explanation).

**5.** The output of each classifier is mapped to a logic-based expression according to knowledge modeled in the reference ontology. The final semantic description is produced by composing the logical conjunction of all expressions.

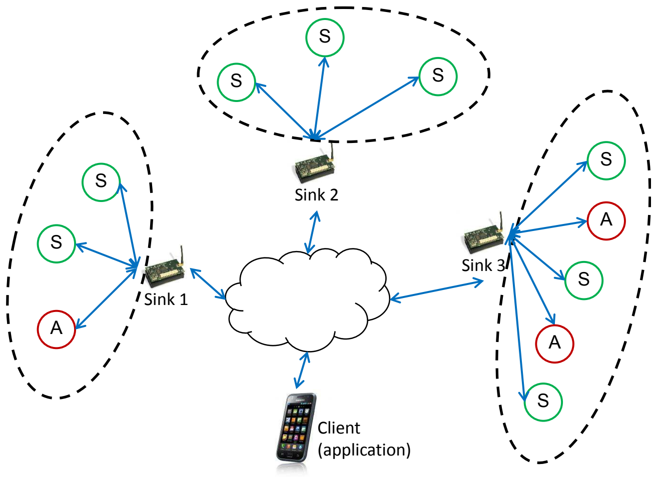


**Fig. 1.** SSN framework architecture

It is important to note that semantic-enhanced CoAP discovery *per se* does not impose restrictions on where data mining happens, whether in clients running application logic, in sink nodes or in sensors having processing capabilities enough. These three main SSN configurations can even be combined according to application or environmental requirements.

### 3.4 Advanced resource discovery in Semantic Sensor Networks

The basic SSN architecture is depicted in Figure 1. Each group of sensors and actuators deployed in a given area will communicate through a local *sink* node, acting as a cluster head for resource-constrained devices in the field. Particularly, sink nodes will: (i) allow sensors/actuators to register their semantic annotation as CoAP resources and (ii) embed a lightweight semantic matchmaker [27]. Local or remote applications are CoAP clients and: (i) use semantic-based discovery to search for sensors or actuators, based on annotated descriptions of their properties and capabilities and produced data; (ii) get raw data and/or event descriptions via standard CoAP primitives. The SSN-XG ontology [8] was used as reference model for the selected domain.

In order to support the novel semantic-based resource retrieval, slight extensions were devised to the CoAP discovery protocol outlined in Section 3.1. They are based only on an innovative usage of standard `URI-query` options and on the addition of new ones. Consequently, the resulting framework is still fully backward compatible: servers which do not support semantics will simply reply to requests returning no resource records.

When receiving a request, a CoAP server will start a matchmaking process comparing it w.r.t. all stored annotations referred to the same ontology. The semantic matchmaking is carried out by executing CAP non-standard inference between request and each resource description, in order to find what elements of the request are lacking in the retrieved resource. A normalized semantic distance in the $[0, 1]$ range is computed for each resource, with lower values for better matches and 0 for full ones. In advanced mobile environments, it is meaningful to involve in a more accurate discovery not only the semantic descriptions, but also data-oriented contextual properties featuring client requests and object/subject/events. In this case an overall *utility function* will be adopted to combine numeric matching with matchmaking results, in order to give a global *score* for resource ranking. Final results will be provided in a normalized ascending $[0, 100]$ % scale, where 100% is the best possible score. Since device and event locations are a key aspect in sensor networks, in the proposed SSN framework the utility function takes into account the geographic distance of each resource from a reference location set in the client request. Hence, a semantic-enhanced CoAP resource is characterized by the following attributes:

– `ro` (reference ontology): it is a new attribute containing the URI of the ontology the resource description refers to. This attribute is mandatory in both resource registration (`POST`) and discovery (`GET`) requests. Its presence allows CoAP servers to discriminate between semantic-enhanced and standard requests.
– `rt`: it is a standard attribute which maintains, instead of an opaque string, the annotated request or the resource description in case of registration or discovery response. It is a concept expression OWL (Web Ontology Language) exploiting the RDF/XML syntax, although the framework does not impose restrictions. In order to cope with the verbosity of XML-based languages, the annotation is compressed and then encoded in base64 for string format compatibility. In the adopted settings *gzip* compression is used, but other schemes can be also employed, such as *EXI*[7] W3C standard or even experimental algorithms for the Semantic Web of Things such as *COX* [24]. Even with compression, an annotation might still be longer than the $270B$ limit of the `URI-query` option field. In such case, multiple `URI-query` fields will be present with `rt` name and the full annotation will come joining such values.
– `at` (annotation type): it is a new attribute indicating ontology language, syntax and encoding scheme of semantic annotation. It is defined in the same way as the standard `type` (media type) attribute. For each language-syntax-encoding, the related MIME type should be added to the CoAP Media Type Registry, which maps MIME type strings to 16-bit codes. The `30004` unassigned code was adopted for the
`application/rdf+zip` MIME type used in this work.
– `st` (semantic task): it indicates which kind of reasoning task is required for resource discovery. Each provided inference is identified by a numeric code. At the moment, `1` is assigned to CAP.
– `sr` (semantic threshold): this new attribute is used in discovery requests to specify a minimum score threshold. Resources having an overall score w.r.t.

[7] Efficient XML Interchange, http://www.w3.org/XML/EXI/

the request lower than this threshold will not be returned to the client. This allows to modulate the granularity of discovery and to limit data transfers when many resources are available. In replies to discovery, this field contains the overall score of a resource w.r.t. the request.

– `lg` (longitude) and `lt` (latitude): they are two novel and optional attributes (expressed in degrees). Within a request, they specify a reference geographical location in order to measure distances of discovered resources and grade matchmaking outcomes; in replies or registrations they simply express the resource location.
– `md` (maximum distance): in a request it indicates the maximum acceptable distance (in meters) from the reference location set with the pair $< lg, lt >$. The adoption of a (center,distance) constraint allows the server to pre-filter resources, so avoiding the relatively expensive semantic matchmaking for resources outside the requested area. In replies, the attribute is used to specify the actual distance of a resource from the reference location.

Some toy examples will clarify both structure and content of registration, request and reply messages in the semantic-enhanced variant of CoAP protocol. Semantic annotations will be voluntarily omitted here for the sake of clarity: several examples will be provided in the case study report in Section 4.

**Registration**. A sensor acting as a CoAP server wants to expose a resource representing its own features and capabilities. The corresponding semantically annotated description is expressed in OWL/RDF language w.r.t. SSN-XG ontology and compressed with gzip. The sensor has (latitude,longitude) coordinates of $(41.109, 16.878)$. The sensor will therefore issue a `POST` CoAP request to:

```
coap://localhost:5683/.well-known/core?rt=xxxxxx
&ro=http://purl.oclc.org/NET/ssnx/ssn&at=30004&lg=16.878&lt=41.109
```

**Discovery request**. An application queries an SSN sink having 193.204.59.75 IP address to find sensors best matching a semantic description expressed in OWL/RDF language w.r.t. Ontosensor ontology and compressed with gzip. The application is interested only in sensors located within 100m from the location at $(41.1566734, 16.884755)$ coordinates. Furthermore, it sets a score threshold of 70% to retrieve only good matches. The application will therefore send a `GET` CoAP request to:

```
coap://193.204.59.75:5683/.well-known/core?
&ro=http://www.memphis.edu/eece/cas/onto_sensor/OntoSensor.txt
&rt=yyyyyy=&at=30004&lg=41.1566734&lt=16.884755&md=100&st=1&sr=70
```

**Discovery reply**. Let us suppose that two sensors match the above request. The CoAP server response payload will be:

```
</HumidSens204>;ct=0;ct=41;at=30004;lg=41.1566735;lt=16.884754;
md=19.4;ro=http://www.memphis.edu/eece/cas/onto_sensor/OntoSensor.txt;
rt=aaaaaaa;sr=76.4;title="Humidity-Sensor-204",
</HumidSens111>;ct=0;ct=30004;lg=41.1566732;lt=16.884756;
md=60.9;ro=http://www.memphis.edu/eece/cas/onto_sensor/OntoSensor.txt;
rt=bbbbbbb;sr=82.1;title="Humidity-Sensor-111"
```

## 4 Case study

In what follows, illustrative examples are presented to better explain flexibility and potentialities our approach provides and to let its novelty emerge. The proposed enhancement has been applied and tested in two different case studies: (i) fire risk prevention; (ii) air conditioning. Both focused on an environment composed of three rooms of the Technical University of Bari equipped with several devices. All of them refer to one or more sink nodes representing the interface of the SSN toward external applications. The CoAP server runs on a sink node based on Android 2.3.3 platform. It is able to accept `GET/POST` messages –for example a sensor discovery request– and send responses to clients. For our experiments the *Californium CoAP framework* [28] was extended with the semantic-based enhancement proposed in Section 3.4. *Copper plugin* [29] for Firefox was used to simulate the requests coming from applications.

Device arrangement is shown in Figure 2: the rooms contain different kinds of sensors –in green– and actuators –in red. Sensor and actuator descriptions are represented by conjunctive concept expressions referring to the same ontology, which extends SSN-XG [8]. In particular, sensors are described by means of their properties and capabilities, whereas actuator descriptions also include features of the event for which they should be activated. Figure 3 shows an example of temperature sensor modeling. We exploited the pattern defined in [8] to describe the measuring features of a sensor, with some differences. In particular, each sensor can "observe" properties modeled as subclasses of `ssn:FeatureOfInterest` and has proper measurement capabilities expressed as subclasses of the `ssn:MeasurementCapability` class. Each specific subclass of `ssn:MeasurementCapability` has a set of measurement properties, represented as subclasses of the `ssn:MeasurementProperty` class. Furthermore, a sensor is related to a subclass of `ssn:EnergyDevice` through the `ssn:hasSubSystem` property to model its energy source.

The first scenario refers to disaster prevention, focusing on discovering possible fire risks in given areas. Thanks to a continuous monitoring of sensed parameters, possible hazards can be quickly detected and recovery procedures rapidly started to take danger under control for both people and structures. The first step is to discover sensors in the environment able to monitor useful data. Application-defined sensor requirements play the role of "request" and the device descriptions the ones of "supplied resources". Let us suppose *a temperature sensor is needed in room C with a large measurement range –able to take both low and high values of temperature–, a medium sampling frequency and low accuracy and resolution –temperature changes quickly during a blaze, so high accuracy and resolution are not required. At the same time, the application requires that sensor has a battery with medium lifetime.* Using concepts defined in the domain ontology the request can described as follows:

$(\mathbf{R_1})\mathbf{FireRisk\_Request} \equiv TemperatureSensor \sqcap \exists\ hasMeasurementCapability \sqcap \forall\ hasMeasurementCapability.(\exists\ hasMeasurementProperty \sqcap \forall\ hasMeasurementProperty.(HighMeasurementRange \sqcap MediumFrequency \sqcap LowAccuracy \sqcap LowResolution)) \sqcap \exists\ hasSubSystem \sqcap \forall\ hasSubSystem.(Battery \sqcap \forall\ hasSurvivalProperty.(MediumBatteryLifetime))$.

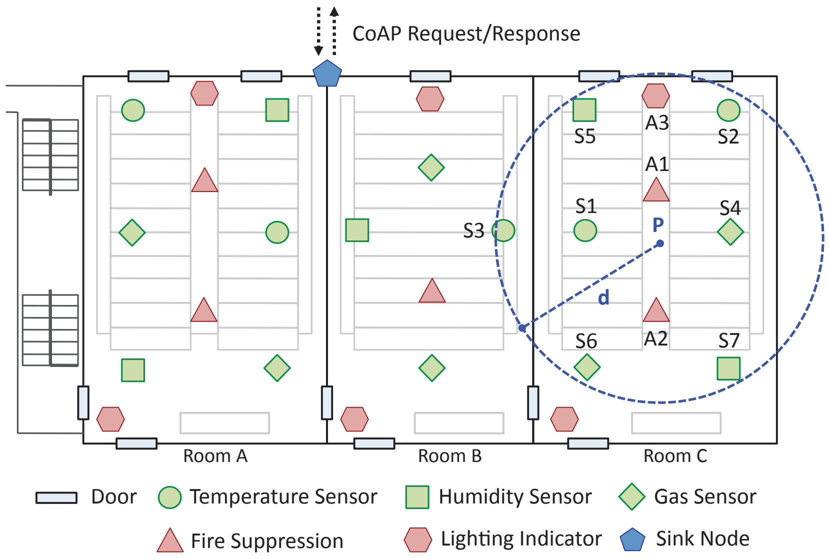


**Fig. 2.** Rooms map

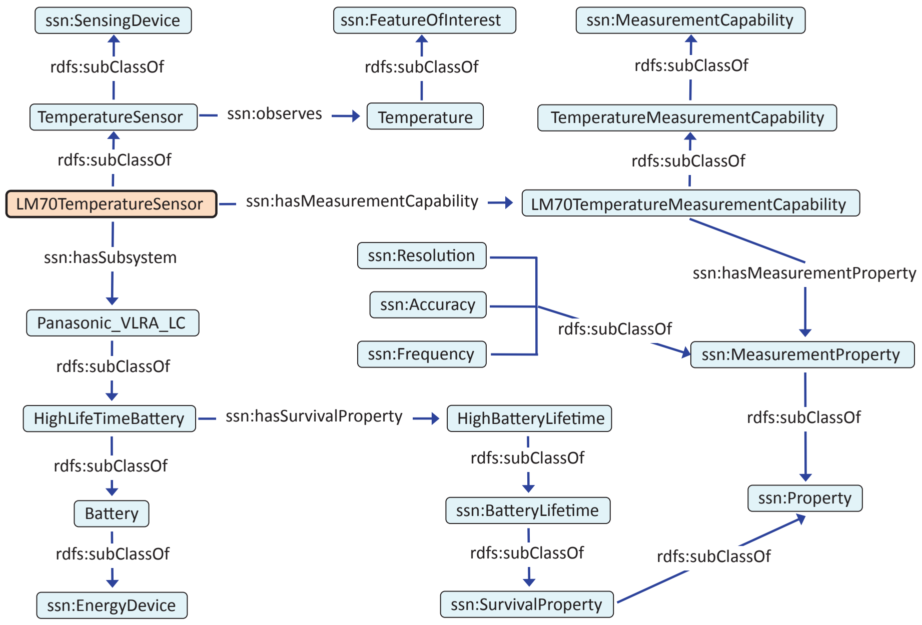


**Fig. 3.** Temperature sensor modeling

Notice that a sensor could also have more subsystems. In that case, related features are conjunctive concept expressions in the filler of the `hasSubSystem` property. With reference to contextual query attributes in the `GET` request, let us suppose to select devices positioned in **Room C** with a maximum distance of **25m** from the reference location $P$ and a threshold discovery score of **65%**. The sink acts as a CoAP gateway w.r.t. deployed sensors. It executes a pre-processing step to exclude components outside the user-specified range. Hereafter the concept expressions for some of the sensor instances inside the measurement area in Figure 2 are summarized.

**($\mathbf{S_1}$)TSic306TemperatureSensor** $\sqsubseteq$ $TemperatureSensor \sqcap \exists\ hasMeasurementCapability \sqcap \forall\ hasMeasurementCapability.(Tsic306TemperatureMeasurementCapability) \sqcap \exists\ hasSubSystem \sqcap \forall\ hasSubSystem.(EnixEnergies\_RS\_689)$.
**Tsic306TemperatureMeasurementCapability** $\equiv$ $\exists\ hasMeasurementProperty \sqcap \forall\ hasMeasurementProperty.(MediumResolution \sqcap HighAccuracy \sqcap MediumFrequency \sqcap LowMeasurementRange \sqcap MediumPrecision)$.
**EnixEnergies_RS_689** $\sqsubseteq$ $Battery \sqcap \forall\ hasSurvivalProperty.(LowBatteryLifetime)$.
**($\mathbf{S_2}$)LM70TemperatureSensor** $\sqsubseteq$ $TemperatureSensor \sqcap \exists\ hasMeasurementCapability \sqcap \forall\ hasMeasurementCapability.(LM70TemperatureMeasurementCapability) \sqcap \exists\ hasSubSystem \sqcap \forall\ hasSubSystem.(Panasonic\_VRLA\_LC)$.
**LM70TemperatureMeasurementCapability** $\equiv$ $\exists\ hasMeasurementProperty \sqcap \forall\ hasMeasurementProperty.(LowResolution \sqcap LowAccuracy \sqcap MediumFrequency \sqcap HighMeasurementRange)$.
**Panasonic_VRLA_LC** $\sqsubseteq$ $Battery \sqcap \forall\ hasSurvivalProperty.(HighBatteryLifetime)$.
**($\mathbf{S_3}$)SE95TemperatureSensor** $\sqsubseteq$ $TemperatureSensor \sqcap \exists\ hasMeasurementCapability \sqcap \forall\ hasMeasurementCapability.(SE95TemperatureMeasurementCapability) \sqcap \exists\ hasSubSystem \sqcap \forall\ hasSubSystem.(Philips\_FR\_6LB)$.
**SE95TemperatureMeasurementCapability** $\equiv$ $\exists\ hasMeasurementProperty \sqcap \forall\ hasMeasurementProperty.(HighResolution \sqcap HighAccuracy \sqcap HighFrequency \sqcap HighMeasurementRange)$.
**Philips_FR_6LB** $\sqsubseteq$ $Battery \sqcap \forall\ hasSurvivalProperty.(MediumBatteryLifetime)$.
**($\mathbf{S_4}$)MX6GasSensor** $\sqsubseteq$ $GasSensor \sqcap \exists\ hasMeasurementCapability \sqcap \forall\ hasMeasurementCapability.(MX6GasMeasurementCapability) \sqcap \forall\ hasSubSystem.(Panasonic\_VRLA\_LC) \sqcap \exists\ hasSubSystem$.
**MX6GasMeasurementCapability** $\equiv$ $\forall\ hasMeasurementProperty.(HighResolution \sqcap MediumAccuracy \sqcap HighFrequency \sqcap MediumPrecision) \sqcap \exists\ hasMeasurementProperty$.

For each device annotation, the sink node apply CAP resolution w.r.t. $R_1$ in order to evidence requested capabilities missing in the device structure. For example, $S_2$ has a different battery lifetime w.r.t. the request, therefore it is a nearly full match. On the contrary, $S_3$ refers to a different type of sensor and does not match the required resolution, accuracy and frequency constraints.

$\mathbf{H_{(R_1,S_2)}} \equiv \forall\ hasSubSystem.(\forall\ hasSurvivalProperty.(MediumBatteryLifetime))$.
$\mathbf{H_{(R_1,S_3)}} \equiv \forall\ hasMeasurementCapability.(\forall\ hasMeasurementProperty.(MediumFrequency \sqcap LowAccuracy \sqcap LowResolution))$.

Afterwards, the sink replies to the client request with the list of suitable sensors in relevance order. The arrangement score is computed via the following utility function:

$$f(R,S) = 100 * \left[1 - \frac{s\_match(R,S)}{s\_match(R,\top)} * \left(1 + \frac{distance(R,S)}{d}\right)\right]$$

where *s_match* measures the CAP-induced semantic distance between a request $R$ and resource description $S$; this value is normalized dividing by the semantic distance between $R$ and the universal concept –*Top* or *Thing*– which depends only on axioms in the ontology. Geographical distance –normalized by user-specified maximum range $d$– is combined as weighting factor. The top results

| ID | URI | Distance | Semantic Rank | Final Rank |
|---|---|---|---|---|
| $S_2$ | `/sink/LM70TemperatureSensor` | 20.53 m | 0.050 | 90.89 % |
| $S_3$ | `/sink/SE95TemperatureSensor` | 23.20 m | 0.075 | 85.53 % |
| $S_1$ | `/sink/TSicTemperatureSensor` | 13.70 m | 0.125 | 80.64 % |
| $S_4$ | `/sink/MX6GasSensor` | 9.60 m | 0.225 | 68.85 % |

**Table 1.** Discovery outcomes in case of fire risk prevention

| **Fire event** | Fire risk | Ignition | Propagation | Flash over | Conflagration |
|---|---|---|---|---|---|
| **temp** ($^\circ C$) | $40 \div 60$ | $60 \div 75$ | $> 75$ | $> 120$ | $> 120$ |
| $\frac{\Delta \mathbf{temp}}{\Delta \mathbf{t}}$ | $> 0$ | $> 1$ | $> 2$ | $5 \div 7$ | $> 7$ |

**Table 2.** Example of fire hazard event detection criteria

of discovery are reported in Table 1. Sensor $S2$ has the best rank because its description is very similar to the request, in fact it has a semantic distance of 0.050. Nevertheless, its final score is about 90% because it is 20m far from the reference location $P$.

Now the application can select sensors to query/observe; subsequently it can apply mining procedures on retrieved data –as explained in Section 3.3– and even detect risk events. For example, let us suppose that temperature sensor $S_2$, having a $f = 0.5Hz$ sampling rate, produces the following data point series in Celsius degrees: $(22.3, 22.5, 25.5, 29.4, 38.7, 49.3, 60.4, 75.5)$. Let us also suppose the fire prevention application adopts an observing window $T = 8s$. Then the following data mining steps are executed:
– Data are divided in $N = fT = 4$-sample blocks: $B_1 = (22.3, 22.5, 25.5, 29.5)$; $B_2 = (38.7, 49.3, 60.4, 75.5)$.
– Average, variance and standard deviation are computed: $\mu_1 = 24.9$; $\sigma_1^2 = 8.3$; $\sigma_1 = 3.3$; $\mu_2 = 56.0$; $\sigma_2^2 = 187.9$; $\sigma_2 = 15.8$.
– Incremental ratio is $\frac{\Delta f}{\Delta t} = \frac{\mu_t - \mu_{t-1}}{T}$ therefore $\frac{\Delta temp}{\Delta t} = \frac{56.0 - 24.9}{8} = 3.9$.
– Based on studies and laws on fire risk prevention, we designed a classifier able to detect one of the five common fire stages, reported in Table 2. In the example, the classifier detects that *fire propagation* is occurring in the environment. It is useful to point out that the example just used the average temperature for the sake of clarity; in the case study, temperature variance, relative humidity, CO and $CO_2$ concentrations are also taken into account.

The event description then becomes a query for the actuator discovery phase. In this way, the application can find all devices able to prevent a given dangerous event or perform recovery procedures. Each actuator can be modeled defining, in addition to operating specification, also the context description that, if verified, should lead to an activation. Different kinds and stages of fire events have been modeled mainly through temperature, extension, propagation speed and toxicity parameters. For example, detection of a propagation event can be characterized by high temperature, moderate extension, low propagation speed and moderate toxicity. In such case, actuator request and fire suppressor devices can be described as:

| ID | URI | Distance | Semantic Rank | Final Rank |
|---|---|---|---|---|
| $A_2$ | `/sink/FireSuppressionTypeB` | 11.01 m | 0.000 | 100.00 % |
| $A_1$ | `/sink/FireSuppressionTypeA` | 5.41 m | 0.150 | 81.75 % |

**Table 3.** Actuator discovery results

| ID | URI | Distance | Semantic Rank | Final Rank |
|---|---|---|---|---|
| $S_3$ | `/sink/SE95TemperatureSensor` | 23.20 m | 0.050 | 90.35 % |
| $S_1$ | `/sink/TSicTemperatureSensor` | 13.70 m | 0.075 | 88.38 % |
| $S_2$ | `/sink/LM70TemperatureSensor` | 20.53 m | 0.125 | 77.23 % |
| $S_4$ | `/sink/MX6GasSensor` | 9.60 m | 0.200 | 72.31 % |

**Table 4.** Sensor discovery results in case of HVAC control

($\mathbf{R_2}$)**Actuator_Request** $\equiv \exists\ hasTemperature \sqcap \forall\ hasTemperature.(HighTemperature) \sqcap \exists\ hasToxicity \sqcap \forall\ hasToxicity.(ModerateToxicity) \sqcap \exists\ hasExtension \sqcap \forall\ hasExtension.(GrowingExtension) \sqcap \exists\ hasSpeed \sqcap \forall\ hasSpeed.(LowSpeed)$.

($\mathbf{A_1}$)**FireSuppressorTypeA** $\sqsubseteq FireSuppression \sqcap \exists\ hasToxicity \sqcap \forall\ hasToxicity.(HighToxicity) \sqcap \exists\ hasExtension \sqcap \forall\ hasExtension.(WideExtension) \sqcap \exists\ hasSpeed \sqcap \forall\ hasSpeed.(HighSpeed) \sqcap \exists\ hasTemperature \sqcap \forall\ hasTemperature(HighTemperature)$.

($\mathbf{A_2}$)**FireSuppressorTypeB** $\sqsubseteq FireSuppression \sqcap \exists\ hasToxicity \sqcap \forall\ hasToxicity.(ModerateToxicity) \sqcap \exists\ hasExtension \sqcap \forall\ hasExtension.(GrowingExtension) \sqcap \exists\ hasSpeed \sqcap \forall\ hasSpeed.(LowSpeed) \sqcap \exists\ hasTemperature \sqcap \forall\ hasTemperature.(HighTemperature)$.

Results of the actuator discovery phase are reported in Table 3. $A_2$ completely satisfies the request and consequently produces an overall score of 100%. Comparing the two fire suppressor actuators, it can be seen $A_1$ presents a lower final rank due to its different specification in terms of detected temperature, toxicity and extension.

The second example we propose aims to show how other applications can query a sink node when searching for devices suitable for specific purposes. The reference scenario refers to an application aiming to control the HVAC system of Room C. Even very small variations of temperature and humidity should be detected in order to improve user comfort: hence devices with finer operating specifications will be required now as evident in the request reported in what follows:

($\mathbf{R_3}$)**AirConditioning_Request** $\equiv TemperatureSensor \sqcap \exists\ hasMeasurementCapability \sqcap \forall\ hasMeasurementCapability.(\exists\ hasMeasurementProperty \sqcap \forall\ hasMeasurementProperty.(LowMeasurementRange \sqcap MediumFrequency \sqcap HighAccuracy \sqcap HighResolution)) \sqcap \exists\ hasSubSystem \sqcap \forall\ hasSubSystem.(Battery \sqcap \forall\ hasSurvivalProperty.(HighBatteryLifetime))$.

Table 4 reports on the top five results for $R_3$. Unlike the first scenario, sensor $S_3$ has the highest score thanks to the lowest semantic distance. Sensor $S_1$ is also a good candidate, with a similar overall score: it is closer to the reference location $P$ but it has a slightly higher semantic rank than $S_3$.

## 5 Conclusion

The paper proposed a novel SSN framework, supporting logic-based matchmaking of meaningful and semantically rich event/device/resource annotations via simple and backward-compatible CoAP extensions. Peculiarities of the proposed solution have been outlined w.r.t. a case study showing its benefits.

Future work includes a thorough experimental campaign on a large testbed, in order to accurately evaluate performance issues, the integration of novel non-standard inferences (such as Concept Contraction and Concept Covering), to make semantic matchmaking even more detailed as well as the extension of underlying logic toward $\mathcal{EL}/\mathcal{EL}^{++}$ for increasing allowed expressiveness of resource and request descriptions.

## Acknowledgments

The authors acknowledge partial support of EU ECTP 2012-14 project "GAIA: Generalized Automatic exchange of port Information Area".

## References

1. Ni, L., Zhu, Y., Ma, J., Li, M., Luo, Q., Liu, Y., Cheung, S., Yang, Q. In: Semantic Sensor Net: An Extensible Framework. Volume 3619 of Lecture Notes in Computer Science. Springer (2005) 1144–1153
2. Avancha, S., Patel, C., Joshi, A.: Ontology-driven adaptive sensor networks. In: First Annual International Conference on Mobile and Ubiquitous Systems, Networking and Services. (2004) 194–202
3. Ke, W., Ayyash, S., Little, T.: Semantic internetworking of sensor systems. In: Mobile Ad-hoc and Sensor Systems, 2004 IEEE International Conference on, IEEE (2004) 484–492
4. Compton, M., Neuhaus, H., Taylor, K., Tran, K.: Reasoning about sensors and compositions. Proc. Semantic Sensor Networks (2009) 33
5. Jiang, G., Chung, W., Cybenko, G.: Semantic agent technologies for tactical sensor networks. In: 2003 SPIE Conference on AeroSense. (2003) 21–25
6. Sheth, A., Henson, C., Sahoo, S.: Semantic Sensor Web. Internet Computing, IEEE **12**(4) (2008) 78–83
7. Guinard, D., Trifa, V.: Towards the Web of Things: Web Mashups for Embedded Devices. In: Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM 2009), in proceedings of WWW (International World Wide Web Conferences), Madrid, Spain. (2009)
8. Compton, M., Barnaghi, P., Bermudez, L., Garcia-Castro, R., Corcho, O., Cox, S., Graybeal, J., Hauswirth, M., Henson, C., Herzog, A., et al.: The SSN ontology of the W3C semantic sensor network incubator group. Web Semantics: Science, Services and Agents on the World Wide Web (2012) in print.
9. Colucci, S., Di Noia, T., Pinto, A., Ragone, A., Ruta, M., Tinelli, E.: A Non-Monotonic Approach to Semantic Matchmaking and Request Refinement in E-Marketplaces. International Journal of Electronic Commerce **12**(2) (2007) 127–154
10. Russomanno, D.J., Kothari, C.R., Thomas, O.A.: Building a Sensor Ontology: A Practical Approach Leveraging ISO and OGC Models. In: The 2005 International Conference on Artificial Intelligence. (2005) 637–643
11. Bizer, C., Heath, T., Berners-Lee, T.: Linked data-the story so far. International Journal on Semantic Web and Information Systems (IJSWIS) **5**(3) (2009) 1–22

12. Patni, H., Henson, C., Sheth, A.: Linked Sensor Data. In: Collaborative Technologies and Systems (CTS), 2010 International Symposium on, IEEE (2010) 362–370
13. Page, K., De Roure, D., Martinez, K., Sadler, J., Kit, O.: Linked Sensor Data: RESTfully serving RDF and GML. Proceedings of the 2nd International Workshop on Semantic Sensor Networks (2009) 49–63
14. Bormann, C., Castellani, A., Shelby, Z.: Coap: An application protocol for billions of tiny internet nodes. Internet Computing, IEEE **16**(2) (2012) 62–67
15. Barnaghi, P., Presser, M., Moessner, K.: Publishing Linked Sensor Data. In: Proceedings of the 3rd International Workshop on Semantic Sensor Networks. (2010)
16. Pfisterer, D., Romer, K., Bimschas, D., Kleine, O., Mietz, R., Truong, C., Hasemann, H., Pagel, M., Hauswirth, M., Karnstedt, M., et al.: SPITFIRE: Toward a Semantic Web of Things. Communications Magazine, IEEE **49**(11) (2011) 40–48
17. Taylor, K., Leidinger, L.: Ontology-driven complex event processing in heterogeneous sensor networks. The Semanic Web: Research and Applications (2011) 285–299
18. Ruta, M., Di Noia, T., Di Sciascio, E., Donini, F.: Semantic-Enhanced Bluetooth Discovery Protocol for M-Commerce Applications. International Journal of Web and Grid Services **2**(4) (2006) 424–452
19. Ruta, M., Scioscia, F., Di Sciascio, E.: Mobile Semantic-based Matchmaking: a fuzzy DL approach. The Semantic Web: Research and Applications (2010) 16–30
20. Ruta, M., Scioscia, F., Di Noia, T., Di Sciascio, E.: A hybrid ZigBee/Bluetooth approach to mobile semantic grids. Computer Systems Science and Engineering **25**(3) (May 2010) 235–249
21. De Virgilio, R., Di Sciascio, E., Ruta, M., Scioscia, F., Torlone, R.: Semantic-based RFID Data Management. In: Unique Radio Innovation for the 21st Century: Building Scalable and Global RFID Networks. Springer (2011) 111–142
22. Mahoney, M.: Adaptive Weighing of Context Models for Lossless Data Compression. Technical report, Florida Tech. Technical Report CS-2005-16, 2005
23. Liefke, H., Suciu, D.: Xmill: an efficient compressor for xml data. SIGMOD Rec. **29**(2) (2000) 153–164
24. Scioscia, F., Ruta, M.: Building a Semantic Web of Things: issues and perspectives in information compression. In: Semantic Web Information Management (SWIM'09). In Proceedings of the 3rd IEEE International Conference on Semantic Computing (ICSC 2009), IEEE Computer Society (2009) 589–594
25. Baader, F., Calvanese, D., Mc Guinness, D., Nardi, D., Patel-Schneider, P.: The Description Logic Handbook. Cambridge University Press (2002)
26. Ruta, M., Di Sciascio, E., Scioscia, F.: Concept Abduction and Contraction in Semantic-based P2P Environments. Web Intelligence and Agent Systems **9**(3) (2011) 179–207
27. Ruta, M., Scioscia, F., Di Sciascio, E., Gramegna, F., Loseto, G.: Mini-ME: the Mini Matchmaking Engine. In Horrocks, I., Yatskevich, M., Jimenez-Ruiz, E., eds.: OWL Reasoner Evaluation Workshop (ORE 2012). Volume 858 of CEUR Workshop Proceedings., CEUR-WS (2012) 52–63
28. Kovatsch, M., Mayer, S., Ostermaier, B.: Moving Application Logic from the Firmware to the Cloud: Towards the Thin Server Architecture for the Internet of Things. In: Proceedings of the 6th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS 2012). (July 2012)
29. Kovatsch, M.: Demo Abstract: Human-CoAP Interaction with Copper. In: Proceedings of the 7th IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS 2011). (June 2011)

# Deriving Semantic Sensor Metadata from Raw Measurements

Jean-Paul Calbimonte[1], Zhixian Yan[2], Hoyoung Jeung[3], Oscar Corcho[1], and Karl Aberer[2]

[1]OEG, Facultad de Informática,Universidad Politécnica de Madrid, Spain
jp.calbimonte@upm.es,ocorcho@fi.upm.es
[2] LSIR, Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland
zhixian.yan@epfl.ch,karl.aberer@epfl.ch
[3] SAP Research, Brisbane, Australia
hoyoung.jeung@sap.com

**Abstract.** Sensor network deployments have become a primary source of big data about the real world that surrounds us, measuring a wide range of physical properties in real time. With such large amounts of heterogeneous data, a key challenge is to describe and annotate sensor data with high-level metadata, using and extending models, for instance with ontologies. However, to automate this task there is a need for enriching the sensor metadata using the actual observed measurements and extracting useful meta-information from them.
This paper proposes a novel approach of characterization and extraction of semantic metadata through the analysis of sensor data raw observations. This approach consists in using approximations to represent the raw sensor measurements, based on distributions of the observation slopes, building a classification scheme to automatically infer sensor metadata like the type of observed property, integrating the semantic analysis results with existing sensor networks metadata.

## 1 Introduction

Ubiquitous sensor networks are a primary source of observations from the physical world, from environmental measuring stations, participatory or citizen sensing, to various sensor applications in traffic, media and health monitoring. Publishing sensor networks data on the web has the potential of increasing public awareness and involvement on these different domains at a massive scale [1]. Cheap sensing devices can be easily configured and deployed, plugged to sensor data platforms such as Cosm[1] for exploitation, storage and querying.

The increasing availability of sensor data in the web introduces higher heterogeneity, which makes it more difficult for potential users to make sense out of these data sources and be able to identify which ones are useful for their applications. An example of this scenario is the Swiss Experiment[2] project, a

[1] Cosm, formerly Pachube `https://cosm.com/`
[2] Swiss Experiment: `http://www.swiss-experiment.ch/`

platform that enables real-time publishing environmental data on the web, from a large-scale federation of sensor networks, mainly in the Swiss Alps. The published data is heterogeneous as it comes from different geographical locations, with different time spans (e.g. observations collected during 1 year, 3 months, etc.), as well as varying sampling rates (e.g. per minute, per 10 minutes). Moreover, the metadata for these sensor types is not always complete and coherent. As an example, to indicate that a sensor measures temperature (i.e. the *observed property*), different sensors use various tag names, like "temperature", "temp", "t", "msptemperature", "tp", etc. Although the data is available for anyone to use, these noisy descriptions are not understandable enough and do not provide semantic information about what this data is about.

In less-controlled scenarios than the Swiss Experiment, the problems of heterogeneity are even more noticeable. For instance in the Cosm web platform, users tag their sensor data as means of metadata, identifying which types of measurements they are publishing. Projects like the Air Quality Egg[3], aiming at promoting air-quality participatory sensing, enable almost any citizen to publish measurements at web-scale. However, the user-provided metadata is often incomplete. In many cases these tags are misleading or they are not provided at all, making it very hard for other users to query or make use of this data.

To overcome this problem, establishing explicit semantics on the metadata has been proposed in previous works, using sensor ontologies [2]. When using these ontologies, sometimes it is needed to manually map the semantic information from the sources to the new metadata model [3], which is a cumbersome and error-prone task. In this paper we propose a novel approach of semantic sensor analysis that infers semantic properties such as the type of observed property, using the raw sensor observations as input. The main contributions of this paper are the following:

- We propose a novel method for representing time series as distributions that represent the slopes of a linear approximation of the initial numeric sensor measurements.
- Based on the statistics of the observation slopes, we infer the type of observed property of the sensor measurements. We use a classification method that exploits the similarity of the slopes distributions.
- We provide a mechanism for enriching the sensor metadata, based on the SSN Ontology [2], with the metadata inferred from the observation slopes.
- We build a self-contained evaluation system linking raw sensor measurements to high-level semantics, and validate our method using two real-life environmental sensor datasets, from the Swiss Experiment and AEMET[4](the Spanish meteorological office).

The remainder of this paper is organized as follows: Section 2 describes the global approach proposed for semantic analysis of sensor data. Section 3 studies

[3] AirQuality Egg `http://airqualityegg.wikispaces.com/`
[4] Agencia Estatal de Metereología: `http://www.aemet.es`

the sensor data representation using slopes, whereas Section 4 focuses on building classification algorithm for inferring observed property types and integrating them to the sensor metadata. In Section 5, we experimentally evaluate our approach. Section 6 summarizes existing related work. Finally, Section 7 includes concluding remarks and points to future works.

## 2 From Raw Measurements to Semantic Metadata

Sensor data is typically represented as time series, describing the evolution over time of a certain observed property. Raw sensor data without any metadata that describes it, has limited use as it is hard to discover, integrate or interpret. While in controlled environments the sensor metadata can be reasonably well managed and controlled by the data owners, in the context of the sensor web, where any citizen is able to produce and publish data, it becomes a more difficult task. While semantic metadata has been shown to be effective for managing large sensor metadata repositories, current proposals require expensive manual curation and tagging (see Section 6). However, these approaches do not look into the data values, from which we can derive some of these metadata properties using analysis and mining techniques.

We describe in Figure 1 our architecture for deriving semantic metadata from sensor data measurements. The approach includes characterizing sensor time series and extracting their observed property types to enrich sensor metadata, and consists of four main layers:

– At the *sensor deployment* layer, sensor nodes provide initial measurements in terms of real-time numerical values, e.g. temperature, humidity, etc.
– In the *semantic sensor analysis* layer, we first represent the sensor data stream using linear approximations and calculate the observation slopes. Based on the sensor slopes, we are able to compute similarity between sensor data series, detecting the observed property types through classification, and performing detection of these types with partial information.
– A semantic representation of the analysis component is integrated into the *semantic metadata*. Using the SSN Ontology as a basis, and combined with domain specific ontologies, this enriched metadata is made available for further processing or querying.
– In the application layer, users can build tools and visualizations to query such sensor data and receive results that include the new metadata computed by the analysis layer.

The deployment layer is usually built using sensor or stream data management systems. These systems centralize the data captured by the devices and provide storage, query interfaces and streaming operators. As for the semantic metadata, we built upon previous work on semantic management of sensor networks [3], centered on the use of the SSN Ontology, coupled with domain ontologies and vocabularies for quantities and units of measurements. For the analysis of the time series, we propose a representation based on the slopes of a linear approximation of the data, as described in Section 3. Then these representations can be used to compare and find similarities among new and existing time series, classifying them according to the detected observed property type, etc. As a result, we are able to complete and query the sensor metadata, as detailed in Section 4.

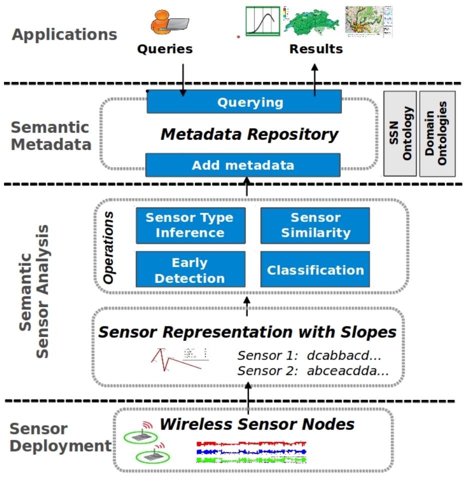
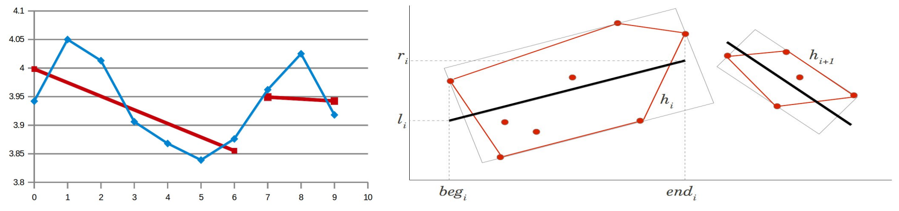


Fig. 1: Semantic Sensor Analysis Architecture

## 3 Sensor Data Representation with Slopes

In environmental time series, similar patterns can be observed periodically over time. These patterns can be characteristic to a type of sensor data, and therefore help to recognize it. If we represent a time series using a linear representation, such as the one in Figure 2(b), the patterns of the data can be associated to the angles of the linear segments or its corresponding slope. For instance, a steep slope indicates a sudden increase of the measured property. The intuition is that if these slopes are repetitive over time, we can build slope distributions that can be representative of a type of time series. Using slopes makes it possible to find similarities between time series that not necessarily have the same value ranges but similar behavior, e.g such as the *air temperature* in two different locations.



(a) Linear approximation (b) Constructing the convex hulls and segments

Fig. 2: Piecewise linear representations

## 3.1 Piecewise Linear Representation

We can use linear segments to approximate a time series (Piecewise Linear Representation, PLR), and analyze the trends by observing the angles that the segments form. For instance in Figure 2(a), we use 2 segments to represent the original 10 data points. Notice that the number of points for a segment can be variable (adaptive approximations). We used the algorithm of [4] for the construction of piecewise linear histograms.

Consider we have a time series of $n$ data points $X = x_1, x_2, ..., x_n$, and we want to fit it in $m << n$ segments. The algorithm maintains a set $B$ of buckets $b_i = h_i, beg_i, end_i, l_i, r_i, h_i$, where $h_i$ is a convex hull of data points, and $(beg_i, l_i), (end_i, r_i)$ are the coordinates of the segment that best fits the convex hull (the segment that bisects the thinnest bounding rectangle of $h_i$ [4]). The slope of $b_i$ can be calculated as $slope(b_i) = \frac{r_i - l_i}{end_i - beg_i}$. The algorithm adds elements to $B$ from $X$, until there are no buckets available, and then it starts to merge those adjacent buckets $b_i$ and $b_{i+1}$ that combined produce the smallest increase in total error. Merging is reduced to a convex hull merge of $h_i$ and $h_{i+1}$. The algorithm iterates until all elements of $X$ have been placed in a bucket. The resulting set of segments of each bucket $b_i$ is the linear approximation of $X$.

For instance in Figure 2(b), the convex hull $h_i$ encloses 8 data points and its minimum rectangle is bisected by the thick black segment defined by the points $(beg_i, l_i), (end_i, r_i)$. This is the linear representation for these 8 points. During the computation of the linear representation, if merging $h_i$ with the next hull $h_{i+1}$ reduces the approximation error, they will form a new single hull with its own bisecting segment. Once we apply this PLR algorithm we have the time series represented as line segments, each with a distinctive slope.

## 3.2 Slope Distributions

To build the slope distributions, we first compute a linear approximation of the time series, using the algorithm described in Section 3.1. It is possible to create linear approximations of different accuracy, depending on the number of segments per unit of time. For instance for a time series of 30 days, if we use 4 segments per day, their slopes will reflect coarse-grained changes in the data during each day. Time series of originally different sampling times, can be represented using the same segment/day rate, in order to be comparable. Obviously, if the original sampling interval is greater than the number of segments/day, the representation with that rate is not possible.

Once the linear representation is built, we can compute the slopes and analyze them. The slope or gradient space, bounded in the $[\infty, -\infty]$ interval for the possible angles $[\frac{\pi}{2}, -\frac{\pi}{2}]$, can be divided in sectors, each represented with a symbol $\alpha_j$ from an alphabet $A$ and we can assign each segment to its corresponding symbol. We propose using the segment representation discussed in the previous section, to compute *slope symbolizations*, which characterize a time series as a sequence $S$ of symbols $s_i$ from an alphabet $A$ that correspond to a type of slope.

In this way, we characterize a time series by the type of variations present in the sensor data, regardless of the data values. For example if we divide the angle space in 4 sectors (labeled $a, b, c, d$), at intervals of $\frac{\pi}{4}$, we can match each segment slope with one symbol. For instance in Figure 3 we have 4 segments, whose symbolic representation is *adac*, by matching each slope with a symbol.

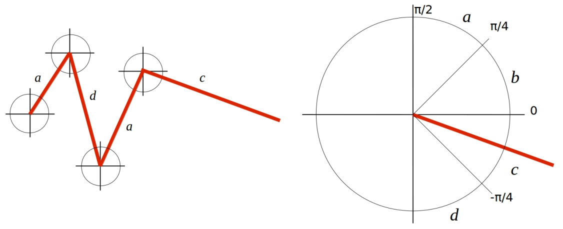


Fig. 3: Slopes symbolization. The angle space in this example is divided in 4 sectors, each of $\frac{\pi}{4}$. According to which division the segments falls in, it is assigned a symbol.

Having this symbolic representation of the slopes, it is possible to compare them to check if two series have similar slope patterns. One simple way to do so, is to generate *symbol distributions*, or histograms that count how many symbols of each type exist in a time series. So a distribution of a sequence $S$ can be defined as a set $D_S$ of elements $d_{\alpha_j} = |\{s_i \in S, s_i = alpha_j\}|$, for all symbols in $A$. For the previous example, it would be a vector $2, 0, 1, 1$, which can be normalized by the total elapsed time, so that we can compare series encompassing different time spans. A simple distance measure is the euclidean distance, defined for two distributions $D_{S_1}, D_{S_2}$ of length $n$ as: $d_{eucl}(D_{S_1}, D_{S_2}) = \sqrt{\sum_i^n (d_{S_1 i} - d_{S_2 i})^2}$

### 3.3 Choosing the angle divisions

Although we can arbitrarily choose how to divide the angle space (e.g. 4 sectors of $\frac{\pi}{4}$ as in the previous example), the actual angles may be more concentrated in some intervals than others. For instance time series with highly changing angles such as *wind speed*, may have steeper gradients than a more stable series. Taking into account this fact, we propose to analyze the training data sets to determine an angle division that better represents the actual distribution of angles in the training set. Using this distribution information, we can divide the angle space in divisions that hold the same number of angles of the training data.

## 4 Deriving Semantic Metadata

After establishing how the data is segmented and symbolized, we can use the symbol distributions for data analysis tasks to help understanding the semantics of the data. Given a time series, if it does not contain appropriate metadata, the potential user of this data can use already analyzed time series and compare the new one with them. We show how this can be done using our symbolization and a simple classification scheme, even with a partial subset of a time series.

### 4.1 Semantic Descriptions

A semantic description of an observation is a collection of statements that includes the observed property (e.g. humidity, pressure), feature of interest (e.g. the air at some location), unit of measurement, among others. For instance, using the vocabulary of the SSN Ontology [2], we describe a *wind speed* observation in Listing 1. The observation, identified as `swissex:WindSpeedObservation1`, has been observed by sensor `swissex:SensorWind1` and reported a value of `6.245`. The sensor observed property type `cf-property: wind_speed` (speed of the wind feature) is defined in a domain specific vocabulary (in this case the Climate and Forecast vocabulary defined by the W3C SSN-XG group[5]). Additional metadata about this observation are omitted for brevity.

```
swissex:WindSpeedObservation1 rdf:type ssn:Observation;
  ssn:featureOfInterest cf-feature:wind;
  ssn:observedProperty  cf-property:wind_speed;
  ssn:observationResult
   [rdf:type ssn:SensorOutput;
    ssn:hasValue [qudt:numericValue "6.245"^^xsd:double]];
    ssn:observedBy swissex:SensorWind1;
```

Listing 1: Wind Speed observation in RDF according to the SSN ontology

Concretely, the `cf-property:wind_speed` property indicates that this is an observation of wind speed, and it has further semantic information in the Climate & Forecast ontology, as seen in Listing 2. It states that it is a property of the wind (`cf-feature:wind`) and is a property of the more general *speed* quantity (`qu:speed`). In order to extract this information, the type of observed property from an unannotated dataset, we propose the classification scheme in the next subsection. The goal is basically to identify the `ssn:observedProperty` for a time series.

```
cf-property:wind_speed rdf:type dim:VelocityOrSpeed;
  rdfs:label "wind speed";
  ssn:isPropertyOf cf-feature:wind;
  qu:propertyType qu:scalar;
  qu:generalQuantityKind qu:speed.
```

Listing 2: Wind Speed property according to the Climate and Forecast vocabulary

### 4.2 Data Classification

Given two sets of time series, a *training set* already annotated according to the type of data that is captured, and an unannotated *test set*, we are interested in finding the observed property for the second set. Assume we have a collection $\mathcal{D}$ of symbol distributions $D_1, ..., D_i, ..., D_n$ as a training set, each of them corresponding to a time series $ts_i$, already classified with a type observed property (e.g. “wind speed”). The classification task consists in finding the best property for time series $ts_{test}$ in the test set.

We can use a simple k-nearest neighbor scheme, which has been successfully used for time series classification [5,6]. First, the time series $ts_{test}$ is segmented

[5] C&F vocabulary: `http://purl.oclc.org/NET/ssnx/cf/cf-property`

and symbolized. Then, we generate a symbol distribution $D_{test}$, as described in Section 3.2, which can be compared iteratively with each of the distributions $D_i$ in $\mathcal{D}$. From the $k$ distributions closer to $D_{test}$, we select the observed property of the majority.

### 4.3 Using Partial Data Subsets

This classification technique may use all the complete time series for computing the symbolization and the slope distribution. However, for types of data with recurring patterns such as the ones present in environmental and meteorological data, using a smaller subset of data can be enough to extract the feature that help detecting the type of observed property. In that case for the construction of the linear representation of the data, we simply choose a subset of the original data: $X = x_1, x_2, ..., x_n$, with a different $n'$ such that $n' < n$.

### 4.4 Querying using the Analysis Results

After executing the classification, we can use the extracted information to complete the sensor metadata, that is then available for querying. In Listing 4 we show a simple SPARQL query that asks for sensors that measure air temperature.

```
SELECT ?sensor
WHERE {
 ?sensor a ssn:Sensor;
         ssn:observes cf-property:air_temperature.}
```

Listing 3: Query all sensors that measure air temperature

The streams produced by sensors can be seen as streaming datasets, whose metadata can also be queried. The stream, identified by a URI, can be seen as an unbounded dataset of observations, some of which are actually used to compute the slope symbolizations and classification described above. The observed properties obtained for the sensor (e.g. `cf-property:air_temperature`) are therefore the observed properties of the stream observations. We can also query for more general types of data, for instance, the generic *temperature* property. In Listing 4 we ask for all stream URIs of sensors that measure some type of temperature.

```
SELECT ?stream ?observedProperty
WHERE {
 ?sensor a ssn:Sensor;
         ssn:observes ?observedProperty.
 ?stream ssn:isProducedBy ?sensor.
 ?observedProperty qu:generalQuantityKind qu:temperature.}
```

Listing 4: Query all streams of sensors that measure air temperature

Furthermore, we can expose the similarity measurements computed between the time series, so that users can also query this information. As an example, in Listing 5 we use the Similarity Ontology[6](`sim`) to represent the computed distance between two series, using our slope representation. Then we can query, for instance the top 5 series similar to a given time series.

[6] The Similarity Ontology: `http://purl.org/ontology/similarity/`

```
swissex:slopeSim1_2 a sim:Similarity;
  sim:subject swissex:timeseries1;
  sim:object  swissex:timeseries2;
  sim:weight  0.32;
  sim:method  swissex:SlopeDistributionDistance.
```

Listing 5: Slope distribution similarity between two time series

This type of queries allows users not only to use the final results of a classification task, but also to query more detailed information including the precision of the computations. This information can be used to validate this metadata or provide insight about the analysis process and the relationship of a sensor stream with other streams. In the case of the early detection of the observed property of a time series, the user may be interested in knowing, for example, how many days of data are typically used for classifying those sensors that measure wind speed 6.

```
SELECT ?sensor ?dur
WHERE {
  ?sensor a ssn:Sensor;
          ssn:observes cf-property:wind_speed.
  ?timeseries ssn:isProducedBy ?sensor.
  ?timeseries swissex:duration [qu:numericalValue ?dur].}
```

Listing 6: Query the number of data days used for classifying wind speed sensors

## 5 Experimentation

The main goal of these experiments is to show that the proposed sensor data representation using slopes can be used to characterize sensor data and extract sensor metadata corresponding to the types of observed properties. First we show how the classification behaves with two real life data sets, in terms of precision. Next, we are interested in experimenting with smaller subsets of data samples, and observing how the classification behaves with less data, as we know there are repeating data patterns. Finally, we compare our approach with a classification using the widely used SAX symbolic representation of the data [5].

To validate the classification approach presented in Section 4.2, we implemented and applied it to two different datasets in the environmental domain: one from the Swiss Experiment[7] and another form AEMET. The data is heterogeneous as it comes from different geographical locations, some have different time spans (e.g. observations collected during 1 year, 3 months, etc), others have different sampling rates.Also the number of sensors per observation type varies (e.g. 78 for temperature, only 4 for snow height). Due to the conditions of the deployments, some of them experimental and others deployed in harsh environments, this dataset contains a considerable amount of noise in the data.

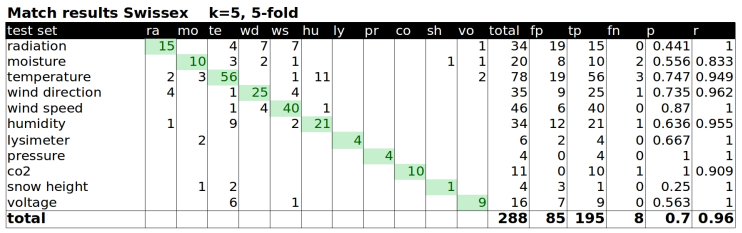
The AEMET dataset consists of sensor data from 100 weather stations managed by the Spanish meteorological office. The data is heterogeneous, coming from stations all over Spain, and was originally collected in intervals of 10 minutes. It contains, in general, less noise and anomalies than the Swiss Experiment dataset, as it comes from stations daily used for meteorological forecasts.

[7] The dataset is available at: `http://lsirpeople.epfl.ch/qvhnguye/benchmark/`

## 5.1 Classification in Swiss Experiment and AEMET

The goal of our first experiment consists in evaluating the effectiveness of the classification in terms of precision and recall. The classifier is expected to assign the correct label (the type of observed property, e.g. "humidity") to time series from a test set. The classifier uses a training set of time series and the evaluation criteria is computed in terms on the number of true positives ($tp$), false positives ($fp$) and false negatives ($fn$): precision ($p = \frac{tp}{tp+fp}$), and recall ($r = \frac{tp}{tp+tn}$).

***Swiss Experiment*** The heterogeneity of the Swiss Experiment dataset required applying different parameters for the linear approximation step. Some time series had very short sampling time intervals (e.g. every 2 seconds for pressure, for at most two days), while others had very long ones (e.g. every half-an-hour for several months). Hence, the approximations were very different in these cases (hundreds of segments per day for short intervals, and only a few per day for long ones).We applied a 5-fold cross validation scheme to divide our dataset in training and test set, and then apply the nearest neighbor algorithm. We present the confusion matrix in Table 4, for $k = 5$.



**Match results Swissex k=5, 5-fold**

| test set | ra | mo | te | wd | ws | hu | ly | pr | co | sh | vo | total | fp | tp | fn | p | r |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| radiation | 15 | | 4 | 7 | 7 | | | | | | 1 | 34 | 19 | 15 | 0 | 0.441 | 1 |
| moisture | | 10 | 3 | 2 | 1 | | | | | 1 | 1 | 20 | 8 | 10 | 2 | 0.556 | 0.833 |
| temperature | 2 | 3 | 56 | | 1 | 11 | | | | | 2 | 78 | 19 | 56 | 3 | 0.747 | 0.949 |
| wind direction | 4 | | 1 | 25 | 4 | | | | | | | 35 | 9 | 25 | 1 | 0.735 | 0.962 |
| wind speed | | | 1 | 4 | 40 | 1 | | | | | | 46 | 6 | 40 | 0 | 0.87 | 1 |
| humidity | 1 | | 9 | | 2 | 21 | | | | | | 34 | 12 | 21 | 1 | 0.636 | 0.955 |
| lysimeter | | 2 | | | | | 4 | | | | | 6 | 2 | 4 | 0 | 0.667 | 1 |
| pressure | | | | | | | | 4 | | | | 4 | 0 | 4 | 0 | 1 | 1 |
| co2 | | | | | | | | | 10 | | | 11 | 0 | 10 | 1 | 1 | 0.909 |
| snow height | | 1 | 2 | | | | | | | 1 | | 4 | 3 | 1 | 0 | 0.25 | 1 |
| voltage | | | 6 | | 1 | | | | | | 9 | 16 | 7 | 9 | 0 | 0.563 | 1 |
| **total** | | | | | | | | | | | | **288** | **85** | **195** | **8** | **0.7** | **0.96** |

Fig. 4: Swiss Experiment confusion matrix, k=5. Column header abbreviations: ra:radiation, mo:moisture, te:temperature, wd:wind direction, ws:wind speed, hu:humidity, ly:lysimeter, pr:pressure, co:$CO_2$, sh: snow height, vo:voltage

We can observe that the effectiveness of the classification varies among the different types of data. The nearest neighbor scheme is also biased as the dataset is highly unbalanced. Since we have comparatively much more samples of temperature or wind speed, than for pressure or snow height, these last are less likely find nearest neighbors of the same class. For instance for *lysimeter* and *snow height*, almost no series are correctly identified, as we have a very small number of series. Nevertheless, in the cases of *pressure* or $CO_2$ the precision is good regardless of the low number of series. This is a special case, since these series have very different slope distributions, and also, have very short sampling interval. Since their resolution is much smaller (e.g. every 2 seconds) than most of the other series in the dataset, their comparison throws very large distances that are quickly discarded.

In cases where the total number of time series was very small (e.g. only 4 for *snow height*), the approach is clearly not effective. It requires a larger training set to have an acceptable precision. Also, when the series are very irregular (sometimes due to noise and false non-curated data in the original dataset), they logically fail to be correctly classified.

***AEMET*** For the AEMET dataset, we followed the same approach as with the Swiss-Experiment. However, for the AEMET data, we had a larger number of time series for every type of data, thus avoiding the problem of lack of training data encountered in the previous tests. Moreover, the dataset sampling interval is the same, making it easier to compare their slope distributions. We applied the classification scheme with a 10-fold cross validation for this dataset. We provide the confusion matrix for $k = 5$ in Table 5.

**Match results AEMET k=5, 10-fold**

| test set | st | ba | te | wd | ws | hu | wsx | pr | wdx | pre | total | fp | tp | fn | p | r |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| soiltemp | 48 |  | 23 | 1 | 1 | 2 |  | 6 |  |  | 81 | 33 | 48 | 0 | 0.59 | 1 |
| battery | 2 | 66 |  | 2 |  | 1 |  |  |  | 10 | 81 | 15 | 66 | 0 | 0.81 | 1 |
| temperature | 15 | 1 | 84 |  |  |  |  |  |  |  | 100 | 16 | 84 | 0 | 0.84 | 1 |
| winddirection |  | 1 |  | 43 | 3 |  |  |  | 53 |  | 100 | 57 | 43 | 0 | 0.43 | 1 |
| windspeed |  | 1 | 1 | 2 | 54 | 4 | 37 |  | 1 |  | 100 | 46 | 54 | 0 | 0.54 | 1 |
| humidity | 1 |  | 1 |  | 2 | 92 | 2 |  | 2 |  | 100 | 8 | 92 | 0 | 0.92 | 1 |
| windspeedmax |  | 1 | 1 | 1 | 54 | 3 | 39 |  | 1 |  | 100 | 61 | 39 | 0 | 0.39 | 1 |
| pressure |  |  | 2 |  |  |  |  | 97 |  |  | 99 | 2 | 97 | 0 | 0.98 | 1 |
| winddirmax |  | 1 |  | 43 | 3 |  |  |  | 53 |  | 100 | 47 | 53 | 0 | 0.53 | 1 |
| precipitation |  | 2 |  |  |  |  | 1 |  |  | 97 | 100 | 3 | 97 | 0 | 0.97 | 1 |
| **total** |  |  |  |  |  |  |  |  |  |  | **961** | **288** | **673** | **0** | **0.7** | **1** |

Fig. 5: AEMET confusion matrix, k=5. Column header abbreviations: st:soil temperature, ba:battery, te:air temperature, wd:wind direction, ws:wind speed, hu:humidity, wsx: wind speed (max), pr:pressure, wdx: wind direction (max), pre:precipitation.

We can notice that in this case the approach achieves better precision, as expected, since we avoided the problems of sampling times and unbalanced types (the number of series per each type is similar or the same). However, it can be observed that there are important false positives at some specific spots. For instance the number of *soil temperature* series falsely identified as *air temperature* is very high. This is in fact an expected result, since both are specializations of the more general type *temperature*. Hence, both share patterns in the time series, that are reflected in the slope distributions that are compared during the classification process. The same situation can be seen between *wind speed* and *wind speed (max)*, and for *wind direction* and *wind direction (max)*.

It is also interesting to see that if we consider the "unification" of similar types of data (e.g. *wind speed* and *maximum wind speed*), the precision is much higher (Figure 6). This suggests that the slope distributions are useful for identifying similar data, because they have very similar slope distributions. This is an expected behavior, for instance for *wind speed* and *wind speed (max)*, which are measurements of the same type of data. In order to discern between

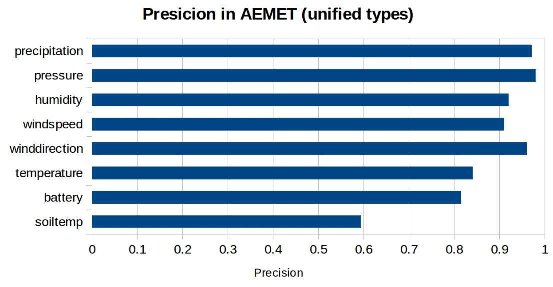
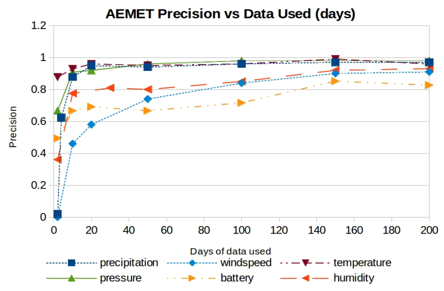


Fig. 6: Precision in AEMET, not differencing the specific types wind speed (max) and wind direction (max).

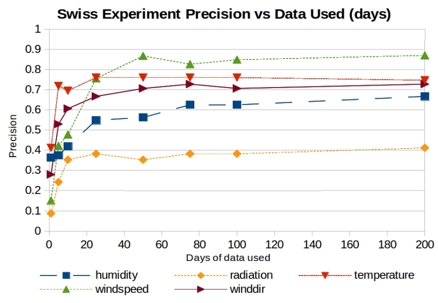
small differences like these, other characteristics of the data have to be taken into account. In these cases where two types of observations are similar, we can use a higher level definition of observed property. For instance, in the Climate and Forecast vocabulary, the specific properties `cf-property:air_temperature` and `cf-property:soil_temperature` both have `qu:temperature` as its general quantity kind.

### 5.2 Classification with Partial Information

In this experiment we aim at showing how the classification precision varies when using smaller subsets of the test data. As we discussed in Section 4.3, for our environmental and meteorological datasets, recurrent slope patterns in the data can be representative enough to compute the slope distribution, and make it possible to classify the data. We have tested the classification reducing the number of days-of-data used for computation. In Figure 7(a) and Figure 7(b) we plot the precision for the AEMET and Swiss Experiment dataset series, for different subsets of the data (expressed in terms of the number of days of measured data). In total we have around 200 days of observations, but we can see that for some types of data we require much less and obtain similar precision in the classification. This is the case especially with series that include very repetitive patterns on a daily basis, but not for others that have a more unpredictable behavior such as *wind speed*. In this case we see that it needs more days-of-data than other types to increase the precision.



(a) AEMET

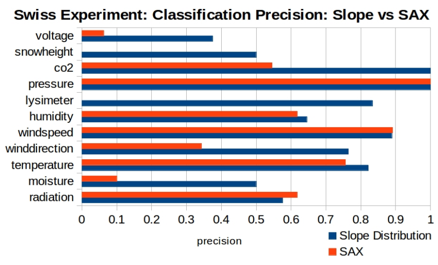


(b) Swiss Experiment

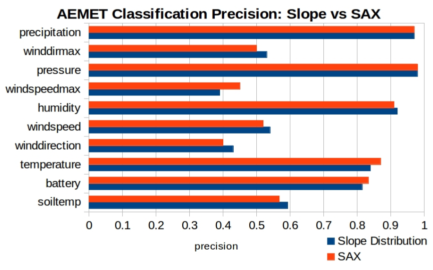
Fig. 7: Classification precision, for different partial datasets, in terms of the days of data used.

### 5.3 Comparison with SAX Classification

The goal of this experiment is to compare our approach with a classification based on the widely used SAX representation of time series [5]. The comparison is based on the precision using both approaches. By classifying with SAX we can verify how well our method behaves in comparison to a well established technique. The SAX approach also produces a symbolization of the time series, although the angles and slopes are not taken into account, as it uses a PAA approximation. We applied the same classification method used for our slope-based representation. We show the classification precision for the Swiss Experiment and AEMET datasets in Figure 8(a) and Figure 8(b) respectively.



(a) Swiss Experiment



(b) AEMET

Fig. 8: Classification precision with SAX and the Slope representation.

As it can be seen, the classification throws similar results for both methods, with small differences in AEMET, and slightly better for the slope-based approach in the Swiss experiment dataset. Using the slopes distributions shows to be helpful at differencing time series with similar values but very different angles. In the case of AEMET, the measured values are already enough to discern between two different types of observation, and hence the results are not improved by the slope distribution. While the SAX representation has been exploited in other ways, for example by considering substrings of a fixed size, instead of only one symbol, this experiment shows that our approach is also able to extract features that help characterizing a type of time series, and enabling its semantic identification. A classification technique throws different results depending on the type of data. Further amendments could be plugged to the classification scheme, but they risk to be too specific to the characteristics of certain datatypes, and such methods are outside of the scope of this work.

## 6 Related Work

Previous works on time series classification and mining, have studied different approaches for summarizing and exploiting sensor raw data, and have been complemented with semantic representations for sensor data management.

***Data Approximations*** High level representations reduce the dimensionality of time series data, in order to reduce the complexity of indexing and comparison algorithms, using different techniques. These include piecewise constant and linear approximations (e.g. PAA[7], APCA[8], PLR[9]) that use constant and linear segments respectively, to represent the original time series. Generally simple to compute, either in batch mode and online using sliding window algorithms, these methods offer accurate approximations of the original data. These representations have been widely used for tasks including similarity search, fuzzy queries, dynamic time warping, clustering and classification [9].

While these approximations reduce dimensionality, some approaches introduce a further step that consists in the symbolization of the time series. These techniques, such as SAX [5], have shown to be space and time efficient for indexing, classification and clustering, and also for additional tasks such as motif discovery and visualization [10]. These symbolizations can be used to compute distance measures that help in classification and clustering tasks [5]. Other works have considered also the slopes of linear approximations such as the STS distance [11] for clustering time series.

SAX symbolization has also been used for sensor events detection [12] and for creating high-level perception abstractions from the raw sensor data, by matching SAX patterns with low-level thematic abstractions [13].

***Time Series Classification*** Particularly, for the task of classification, different techniques such as decision trees, neural networks and bayesian classifiers have been used [6]. Classification approaches usually fall into the following three categories: distance-based, feature-based and model-based[6]. Simple distance measures such as euclidean, are very limited because they only consider one-to-one matches in the time axis. Distance measures with more elastic matching for the time axis, such as Dynamic Time Warping (DTW), have been proved effective for similarity matching [14]. These have been coupled with k-nearest neighbor (k-NN) classifiers, proving an effective combination for a number of time series classification problems [15,16]. These techniques have space and time computation limitations in some scenarios, and offer little explanation on why a series belongs to a particular class [17]. Feature-based approaches try to find properties that are representative of a type of series, in order to classify them. Most of these approaches use a high level representation e.g. symbolization or discretization methods, before extracting the features[6] while others work extracting representative subsequences (e.g. shapelets [17]).

***Semantic Sensor Representations*** The task of modeling sensor data and metadata with ontologies has been addressed by the semantic web research community in recent years. Early ontology proposals for describing wireless sensors have been reviewed in [18]. However, the focus of most of these approaches was on sensor meta information, while the description of observations was generally overlooked. Besides some of these approaches lack ontology design best practices of reuse and alignment with standards an reference ontologies. Others, including the OntoSensor ontology [19], use the concepts defined in the OGC SensorML[8] standard as a basis. More recent proposals like [20] and [21], also consider the OGC Observations and Measurements (O&M) standard[9] to represent observations captured by sensor networks.

Recently, through the W3C SSN-XG group, the semantic web and sensor network communities have made an effort to provide a domain independent ontology, generic enough to adapt to different use-cases, and compatible with the OGC standards at the sensor and observation levels. The result, the SSN ontology [2], is based on the stimulus-sensor-observation design pattern [22] and the OGC standards.

[8] SensorML. `http://www.opengeospatial.org/standards/sensorml`

[9] OGC O&M: `http://www.opengeospatial.org/standards/om`

## 7 Conclusions and Future Work

We have described an approach for identifying the type of data from sensor data sources, using a symbolic representation of the time series slopes. We have shown how this representation can be used for enriching semantic sensor metadata. We have shown specific use cases of time series data classification, providing similarity measures, and metadata aggregation that can be queried in terms of high-level standard ontologies. Finally, we evaluated our approach with real-life datasets of the Swiss-Experiment project and AEMET.

We have shown through experimentation that this representation can be useful for balanced datasets, as the classification gets biased when there are small numbers of samples in the training set, for a particular type of data. Moreover, our results show that this representation can help grouping data of the same type, despite geographical locations, since it is based on the distribution of slopes of a linear approximation. Therefore, it can identify similarities of related types of data: e.g. *air temperature* and *soil temperature*. We have compared our characterization of sensor data with a competitive approach, and showed that for the chosen environmental datasets it effectively enables the extraction of semantic metadata.

The proposed approach, however, was evaluated within the same dataset, and in the future we will study its applicability in an inter-dataset classification. This framework could be used in the future for other tasks such as clustering, or for identifying simple patterns in streams of sensor data. Moreover, complex symbolizations consisting of sequences of slopes could be considered, which would represent more complete patterns that can be exploited. Also, we can consider building a more complex representation that includes not only the slopes information but also the value ranges, and even tags and labels provided the data publishers. This may enable a more complete and accurate extraction of metadata that enriches the growing Semantic Sensor Web. As a final future path, we may consider applying online execution of these techniques for real-time analysis.

## References

1. Sheth, A., Henson, C., Sahoo, S.: Semantic sensor web. IEEE Internet Computing **12**(4) (2008) 78–83
2. Compton, M., Barnaghi, P., Bermudez, L., García-Castro, R., Corcho, O., Cox, S., Graybeal, J., Hauswirth, M., Henson, C., Herzog, A., Huang, V., Janowicz, K., Kelsey, W.D., Phuoc, D.L., Lefort, L., Leggieri, M., Neuhaus, H., Nikolov, A., Page, K., Passant, A., Sheth, A., Taylor, K.: The SSN ontology of the W3C semantic sensor network incubator group. Journal of Web Semantics **(In press)** (2012)
3. Calbimonte, J.P., Jeung, H., Corcho, O., Aberer, K.: Semantic sensor data search in a large-scale federated sensor network. In: Proc. 4th International Workshop on Semantic Sensor Networks. (2011) 14–29
4. Buragohain, C., Shrivastava, N., Suri, S.: Space efficient streaming algorithms for the maximum error histogram. In: Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on, Ieee (2007) 1026–1035
5. Lin, J., Keogh, E.J., Wei, L., Lonardi, S.: Experiencing sax: a novel symbolic representation of time series. Data Min. Knowl. Discov. **15**(2) (2007) 107–144

6. Xing, Z., Pei, J., Keogh, E.J.: A brief survey on sequence classification. SIGKDD Explorations **12**(1) (2010) 40–48
7. Keogh, E., Chakrabarti, K., Pazzani, M., Mehrotra, S.: Dimensionality reduction for fast similarity search in large time series databases. Knowledge and information Systems **3**(3) (2001) 263–286
8. Chakrabarti, K., Keogh, E., Mehrotra, S., Pazzani, M.: Locally adaptive dimensionality reduction for indexing large time series databases. ACM Transactions on Database Systems (TODS) **27**(2) (2002) 188–228
9. Keogh, E., Chu, S., Hart, D., Pazzani, M.: Segmenting time series: A survey and novel approach. Data mining in time series databases **57** (2004)
10. Kasetty, S., Stafford, C., Walker, G., Wang, X., Keogh, E.: Real-time classification of streaming sensor data. In: Tools with Artificial Intelligence, 2008. ICTAI'08. 20th IEEE International Conference on. Volume 1., IEEE (2008) 149–156
11. Möller-Levet, C., Klawonn, F., Cho, K., Wolkenhauer, O.: Fuzzy clustering of short time-series and unevenly distributed sampling points. Advances in Intelligent Data Analysis V (2003) 330–340
12. Zoumboulakis, M., Roussos, G.: Escalation: Complex event detection in wireless sensor networks. In: Proceedings of the 2nd European conference on Smart sensing and context, Springer-Verlag (2007) 270 – 285
13. Payam Barnaghi, Frieder Ganz, C.H., Sheth, A.: Computing perception from sensor data. In: Proceedings of the 2012 IEEE Sensors Conference (to appear). (2012)
14. Ding, H., Trajcevski, G., Scheuermann, P., Wang, X., Keogh, E.J.: Querying and mining of time series data: experimental comparison of representations and distance measures. PVLDB **1**(2) (2008) 1542–1552
15. Xi, X., Keogh, E., Shelton, C., Wei, L., Ratanamahatana, C.: Fast time series classification using numerosity reduction. In: Proceedings of the 23rd international conference on Machine learning ICML 06. Volume 150., ACM Press (2006) 1033–1040
16. Geurts, P.: Pattern extraction for time series classification. Principles of Data Mining and Knowledge Discovery (2001) 115–127
17. Ye, L., Keogh, E.: Time series shapelets: a new primitive for data mining. In: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM (2009) 947–956
18. Compton, M., Henson, C., Lefort, L., Neuhaus, H., Sheth, A.: A survey of the semantic specification of sensors. In: Proc. 2nd International Workshop on Semantic Sensor Networks. (2009) 17
19. Russomanno, D., Kothari, C., Thomas, O.: Sensor ontologies: from shallow to deep models. In: Proc. 37th Southeastern Symposium on System Theory. (2005) 107–112
20. Barnaghi, P., Meissner, S., Presser, M., Moessner, K.: Sense and sensability: Semantic data modelling for sensor networks. In: Proceedings of the ICT Mobile Summit. (2009)
21. Compton, M., Neuhaus, H., Taylor, K., Tran, K.: Reasoning about sensors and compositions. In: SSN. (2009)
22. Janowicz, K., Compton, M.: The Stimulus-Sensor-Observation Ontology Design Pattern and its Integration into the Semantic Sensor Network Ontology. In: SSN. (2010) 7–11

# Extension of the Semantic Sensor Network Ontology for Wireless Sensor Networks: The Stimulus-WSNnode-Communication Pattern

Rimel Bendadouche[1], Catherine Roussey[1], Gil De Sousa[1], Jean-Pierre Chanet[1], and Kun Mean Hou[2]

[1] Irstea, National Research Institute of Science and Technology for Environment and Agriculture
{rimel.bendadouche,catherine.roussey,gil.de-sousa, jean-pierre.chanet}@irstea.fr
[2] LIMOS, Laboratoire Informatique, Modélisation et Optimisation des Systèmes
kun-mean.hou@isima.fr

**Abstract.** Wireless Sensor Networks (WSN) are designed to collect large amounts of heterogeneous data to monitor environmental phenomenon. Our aim is to adapt WSN nodes communication to their context, in order to optimize the lifetime of the network. Our description of context and WSN characteristics are based on ontologies. Based upon a critical analysis of existing ontologies which formalize the WSN domain, we determine that the Semantic Sensor Network (SSN) ontology is the most suitable to represent the WSN issues. However, as the communication data policy is not characterized either by SSN or by other ontologies, we propose to enrich the SSN ontology with a new pattern describing communication. In this paper, we will first integrate the different concepts related to WSN in the SSN ontology and then we will use the resulting ontology, called Wireless Semantic Sensor Network ontology, in an agri-environmental scenario to illustrate the interest of our approach.

## 1 Introduction

In recent decades, thanks to the advance of embedded systems and wireless technologies, Wireless Sensor Network (WSN) becomes widely used. WSN usually consists of a set of wireless sensors nodes (from a few tens to a few hundreds), which acquire, store, transform and communicate data using wireless technologies [1]. The development of WSN is motivated by several systems such as military monitoring system, smart home system, etc.

Our work focuses particularly on agri-environmental applications. A WSN can improve the knowledge of the environment and the effectiveness of management methods for instance in our case to prevent flooding. In this application, the WSN collects large amounts of heterogeneous data (rainfalls, temperatures, water levels, etc.). This work aims to integrate into an information system these data in an optimal way.

However, by its nature, the WSN nodes are characterized by limited resources: energy, computing power and storage capacity. In this kind of network, energy is the main constraint that limits the rendered services by each node. Knowing that communication is the most energy-consuming service, the WSN nodes should limit the communication of their data as much as possible to increase their lifetime. That is why the communication data policy may be very different from the acquisition one. Even when two nodes are parameterized with the same acquisition policy, their communication policy can be very different depending on their resources available. WSN nodes with enough energy could communicate all their acquired data with the same frequency as the acquisition one. On the contrary, WSN nodes that have a low energy level would communicate, for example, just an aggregate value (max, min, etc.) with a lower frequency than the acquisition one. Thus, the communicated data of these nodes are different even if the same kind of data is acquired.

Therefore, WSN nodes must be able to adapt to their context (for example, their energy level). The context is a set of information that the node may have on its environment. A node takes into account its context to improve its lifetime and consequently the overall functioning of the network. Ontologies are a solution to describe the sensors, their data and their context. They also define metadata vocabularies. Through ontologies, it will be possible to assign to the data a description of its acquisition and communication policies in order to enhance the integration of the sensor data. Moreover, by using semantic rules, ontologies are also components of reasoning systems. So we can also use them to develop an intelligent system able to modify the nodes' behaviour to optimize their lifetime and, by extension, WSN lifetime. In our case, ontologies have two objectives: defining metadata about WSN or observation of phenomenon and defining the knowledge needed for reasoning purposes.

In the literature, several sensor ontologies are developed in different domains. In this paper, we will illustrate how we can use these ontologies to describe WSN concepts. The remainder of this paper is organized as follows. Section 2 details our needs and defines several concepts related to acquisition and communication policies. Section 3 presents a state of the art about sensor ontologies. Our critical review highlights that the SSN ontology is the most suitable to describe WSN topics. However, this ontology ignores the communication process of the sensor data. After a brief presentation of some parts of SSN ontology, section 4 gives our proposition of a extension of this ontology called Wireless Semantic Sensor Network ontology (WSSN). Section 5 illustrates the use of WSSN ontology in agri-environmental scenario with JADE platform. This experiments evaluate the improvement of the lifetime of WSN nodes using our approach.

## 2 Description of our Needs

WSN are largely used to collect data of various domains (agri-environmental, military, etc). In our work we are interesting in agri-environmental applications using Decision Support System (DSS) to anticipate environmental risks. Before presenting our needs, we will present our use case: the monitoring of flooding in a watershed using a WSN that provides data to a dedicated DSS. A watershed is an extent or an area of land where surface water from rain, melting snow or ice converges to a single point, usually the outlet of the basin and joins another waterbody, such as a river, lake, reservoir, estuary, wetland, sea, or ocean. In case of heavy rain, the DSS evaluates and predicts the flooding risks in the outlet of the basin. The system needs WSN to monitor the amount of rainfall (precipitation) during a given time interval at the top of mountains and hills in the watershed. This precipitation measure is carried out through a pluviometer. When it rains, the WSN performs the precipitation measurement every hour. If the rain persists, the WSN changes its communication policy and sends measurements every minute for a better monitoring of the phenomenon. Then, if the precipitation quantity exceeds a threshold, the DSS requests water level measures of the outlet and the different waterbodies that feed the basin. Then, the WSN sends these measurements. The flood phenomena can be modelled by the succession of four states:

– "Normal", when it does not rain;
– "Waiting for rise in water levels ", when it rains;
– "Rise in water levels", when the rain persists and a certain amounts of precipitation fall;
– "Flood warning", when the water level of the outlet and waterbodies exceed a threshold.

The current state of the observed phenomenon can be deduced from WSN measurements, the acquired data. For example, if the total rainfall on the watershed exceeds a certain threshold, then the flood phenomenon changes from "normal" to the second state "Rise in water levels". Thus, acquisition and communication data policies of a WSN node reflect both the current node state (for example, the node energy level) and the phenomenon state. All of these state changes can be performed by a rule-based system which is a type of expert systems. The reasoning can be modelled using decision rules applied to a set of facts. Indeed, some types of ontologies define entities used in rule-based engine. In this scenario, WSN node should:

1. interoperates with other nodes;
2. reasons to be able to adapt to its context: the wireless sensor node state and the observed phenomenon state;
3. changes intelligently its acquisition and communication data policies under the supervision of both the network and the DSS.

In the following, we assume the definition of the "context" notion based on "state" one:

***State***: "The state is a qualitative data which changes over time, summarizing a set of information."

***Context***: "The context is a set of entities states or information describing an environment where an event occurs".

Ontologies as metadata vocabularies will allow to describe the sensor data, define the node state and the phenomena state. In our work, we use ontologies to meet several needs:

– Normalize the WSN vocabulary. This vocabulary will define concepts relating to:
    • Composition of WSN and their settings: node, sensor, energy device, etc.

  - Acquisition of the measure: stimulus, measure, observed phenomena, acquisition policy, etc.
  - Data communication: data types, aggregation data type, data stream, communication policy, etc.
- Define message formats using the previous vocabulary for building acquisition and communication data stream. The messages will contain some information about node state.
- Model the knowledge used in rules engines. The ontology will help to formalize the facts used by the rules. Two rules engines are needed. The first one is dedicated to deduce the state changes of the observed phenomenon from the data provided by WSN. The second engine adapts the WSN node acquisition and communication policies based on the state of the observed phenomenon and the state of the node. It also deduces the state of the node based on its energy level.

In summary, our ontology should allow us to normalize a metadata vocabulary used to describe the WSN and its components. These metadata will describe sensors data and the measuring process of their measures. The ontology will also define the data exchange format for describing data stream. Finally, it will also support the intelligence of the WSN node, the WSN and the DSS. It should be used in a rule-based system to infer new data (according to the state of the WSN nodes and the observed phenomenon) an control WSN behaviour by optimizing data communications. In the literature, several sensor ontologies fulfil part of these different needs. In order to find the most suitable ontology to our purposes, we present in the next section an overview of these ontologies.

## 3 State of the Art

In our scenario, our ontology should describe 3 topics: “1: Sensor topic”, the WSN and its components; “2 Observation topic”, the measurement process; “3 Data topic”, processes using data like aggregation or communication processes. In the literature, several state-of-art address the sensor ontologies topics issue such as in [4] and [7].

Inspired by this latest review [7], which has analyzed in details sensor ontologies with the two first topics previously mentioned: ’Sensor’ and ’Observation’, our work analysed these ontologies focusing on the third topic – the ’Data topic’, as reviewed in [4].

**Data**: a third topic that ontology can describe is the sensor data. The data is the result obtained by an observation which may be transformed, stored and communicated to the DSS. In this topic, we are interested in all the processes using this data, as transforming and communicating, and generated data stream (set of data). Ontologies that integrate this topic tend to describe the data generated by the sensor network. They should describe the acquired and communicated data.

In Table 1, the presence/lack of the symbol (*) identifies the ability/inability of the ontology to describe the related topic. However, for the two first topics we attribute a percentage of the facets which are described by the ontology. For example SSN ontology has 8/8 sensor facets, it means that SSN ontology describe all the facets of the sensor topic.

|  | Sensor | Observation | Data |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Ontologies | 8 facets | 5 facets | Data | Data Stream | Sensing process | Communicating process | Tranforming process | States | Data Quality | Acquisition policy | Communication policy |
| SSN [3] ontology | 8/8 | 4/5 | * |  | * |  |  |  |  | * |  |
| CESN otology [6] | 2/8 | 1/4 | * |  | * |  |  |  |  |  |  |
| CSIRO ontology [9] | 8/8 | 4/5 | * |  | * |  |  |  |  | * |  |
| Sensei O&M ontology [3] | N/V | N/V(not available) | * |  | * |  |  |  |  |  |  |
| OOSTETHYS ontology [5] | 2/8 | 2/5 | * |  | * |  |  |  |  |  |  |
| MMI [4] ontology | 5/8 | N/V |  |  | * |  |  |  |  | * |  |
| SWAMO [5] ontology | 3/8 | 2/5 |  |  | * |  |  |  |  |  |  |
| SEEK ontology [14] | N/V | N/V | * |  | * |  |  |  |  |  |  |
| SDO ontology [12] | 2/8 | 2/5 | * |  | * |  |  |  |  | * |  |
| SeReS O&M ontology [15] | N/V | N/V | * |  | * |  |  |  |  |  |  |
| OntoSensor ontology [16] | 5/8 | 5/5 | * |  | * |  |  |  |  | * |  |

Table 1: Review on sensor, observation and data ontologies

According to the analysis made by the authors in [7] and Table 1, it can be noted that in one hand, CSIRO and SSN are the best ontologies to represent the first topic 'Sensor', and on the other hand, CSIRO, SSN, SDO and Ontosensor ontologies are best suited to represent Observation topic.

**Data**: SDO ontology is an example of such ontologies, it gives just a simple classification of sensor data. In SSN ontology, the data generated by the sensor are associated to *SensorOutPut* class and their values to the *observationValue* one. However, none of the ontologies previously mentioned, describes data stream and their characteristics, especially the communication of data acquired by sensors. For processes using data, only the sensing process is described by the 11 sensor ontologies. However none of these ontologies describes the communicating and the transforming processes. According to our state of art, none of these ontologies describes completely these three topics: Sensor, Observation and Data. However, SSN ontology, which integrates several ontologies as CSIRO, models very well the two first topics. Hence we prefer to improve this ontology rather than developing a new one. Knowing that communication is the most energy-consuming process of the WSN node, we integrate it in the SSN ontology.

In order to enrich SSN ontology, WSN concepts will be described in the next section.

[3] http://www.w3.org/2005/Incubator/ssn/XGR-ssn-20110628/.

[4] Marine Metadata Interoperability, MMI Device Ontology: A Community Development Project.

[5] http://marinemetadata.org/references/ontswamo.

## 4 Extension of the SSN Ontology

SSN ontology was created by the W3C Semantic Sensor Network Incubator Group (The SSN-XG). One of the main objectives of this group is the development of ontologies that describe sensors and sensor networks for web applications. According to their review of sensor ontologies, the group has classified these ontologies in two categories: sensor topic and observation topic. The SSN ontology integrates these two topics in a single ontology and is based on the Stimulus-Sensor-Observation ontology design pattern. This pattern is specialized to cover sensor key concepts. In order to facilitate its evolution, SSN was built in a modular way and presents in [8] an overview of modules. For example, the main classes are: device, observation, feature of interest, sensing process, deployment, platform and measurement capability. The SSN ontology covers the general aspect of sensing applications and by specializing this ontology it is possible to describe some key concepts of WSN domain. However, SSN ontology does not focus on the communication process and new classes related to this process and to data processing are needed. So, to integrate all aspects of WSN domain, we have to describe the following points:

- WSN node devices: the SSN module “Device” does not contain all devices that a WSN node can have.
- Communication: the SSN ontology ignores the communication process of acquired data. However, this ontology describes precisely the acquisition process.
- Data stream: we want to differentiate at least two data stream involved in WSN.
    - Acquisitional Data Stream: data stream produced by the sensors of a WSN node.
    - Communication Data Stream: data stream communicated by the communicating device of a WSN node.
- State: in our approach, the WSN node will adapt its behaviour to its context composed of the phenomenon state and its own state. Thus, we need to define the WSN node states to optimize the network resources.

**Notation**: Most of the figures describing the ontology and the examples have been created with the help of the Concept-map (CMAP) Ontology Editor (or COE). In this figures, the new WSSN classes are presented with dotted shape.

### 4.1 WSN Node Devices

A Wireless Sensor Network (WSN) consists of a set of small entities called WSN nodes. These nodes have limited energy resources, memory and computing capacities. They communicate with each other using wireless technologies such as IEEE 802.15.4 (ZigBee) [1]. A WSN node is composed of several devices like the communicating device and the sensing device. The sensing device, also named sensor, measures the property of a feature of interest. In the SSN ontology, a SensingDevice is a Device, a specialization of System. A system is composed of several subsystems. Thus, a sensing system may be composed of several devices like battery and sensor. A sensing device is characterized by some measurement capability and measurement property as accuracy or precision. In WSSN ontology, a *WirelessSensorNetworkNode* is composed of four devices: *ProcessingDevice*, *CommunicatingDevice*, *EnergyDevice* and *ssn:SensingDevice* as shown in the Figure 1. A communicating device will also have some communication capability and communication property as frequency and bandwidth. A *WirelessSensorNetwork* is also a system composed of several *WirelessSensorNetworkNodes*.

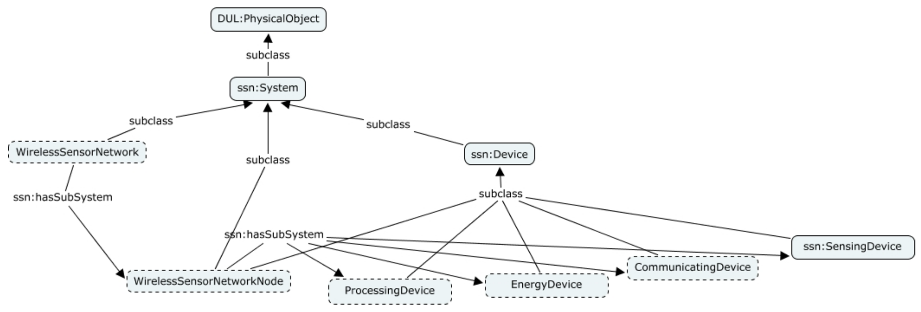


Fig. 1: Wireless Sensor Network (WSN)

The sensing device acquires a data. The processing device produces a new data, that can be an aggregation of a set of acquired data. The communicating device communicates a data acquired by the sensing device or produced by the processing device. In the next section, we present the communication process.

### 4.2 Communication Process

The SSN ontology is based on Stimulus-Sensor-Observation Ontology Design Pattern (ODP) proposed by [13]. This pattern, presented in Figure 2, describes the measurement process centered around the notions of stimuli, sensor and observations. However, to highlight this communication process, we need another pattern: the Stimulus-WSNnode-Communication ODP which describes this process. This new pattern is presented in the next section.
*Stimulus*: Stimuli are the starting point of any process as they act as triggers for sensors or communicating devices. A stimulus can be a detectable change in the observed phenomena which act as an unintentional stimulus for sensing process. It could also be an incomming communicating request which act as an intentional stimulus for communicating process.
*Sensor (SensingDevice)*: Sensors are physical objects that perform observations, i.e., they transform an incoming stimulus into another, often digital representation.
*CommunicationDevice*: Communicating Devices are physical objects that perform communications of data acquired by sensors.
*Observation*: Observations act as the nexus between incoming stimuli, sensor (sensing

device) and output of the sensor. Therefore, observations are social, not physical, objects. They define the context of the measurement process. For example, they can fix parameters such as time and location.
*Communication*: Communications act as a nexus between incoming stimuli, communicating device and output of this device. Like observations, communications are social objects. They define the context of communication processes of data acquired by sensors during observations.
*Procedure*: Procedure is a description of how a device works, i.e., how a certain type of stimuli is used to produce a digital representation: a data output of the associated process.
*Sensing*: Sensing is a description of how the sensing device works. It represents the measurement process.
*Communicating*: Communicating is a description of how the communicating device works. It represents the communication process.
*SensorOutPut*: Sensor output is the result of the observation: a data acquired by sensor. It is a symbol representing a value.
*CommunicationOutPut*: Communication output is the result of the communication: a data transmitted by the communicating device. A communicated data can be equal to a data acquired by sensor or to a data generated from a set of data acquired by sensors using some aggregation procedure for example.

We notice that an observation may be followed by a communication and a communication should precede an observation. The link between observation and communication is realized by the properties hasObservation and hasCommunication. A communication may associate to several observations if the communicated data correspond to the aggregation of data acquired during several observations.

**Aligning the Stimulus-WSN node-Communication ODP with DUL** To ease the integration of this new ODP in the SSN ontology this pattern has been aligned to the ultra light version of the DOLCE foundational ontology (DUL) and refined to match the content of the SSN ontology. Figure 3 presents the result of this integration.

The class Communication in the WSSN ontology provides the structure to represent a single communication. A communication is a situation that describes a communicating device, a communicating method and a single value communicated by a particular device. Thus the Communication class is a subclass of DUL: Situation. Note that a communication is linked to at least one observation which is the situation where a data is acquired by sensor. The WSSN ontology defines several properties for instances of the class Communication.

*observationTime*: points to the time when the acquired data applies to the feature of interest. This time is equivalent to the observation sampling time of the associated observation. As previously said each communication should be preceded by at least one observation. This property and observation sampling time property are redundant. Observation time property is optional and can be used at user's request.
*communicationSendingTime*: points to the time when the communicating device communicates its data.
*communicationReceptionTime*: points to the time when the communicated data arrived

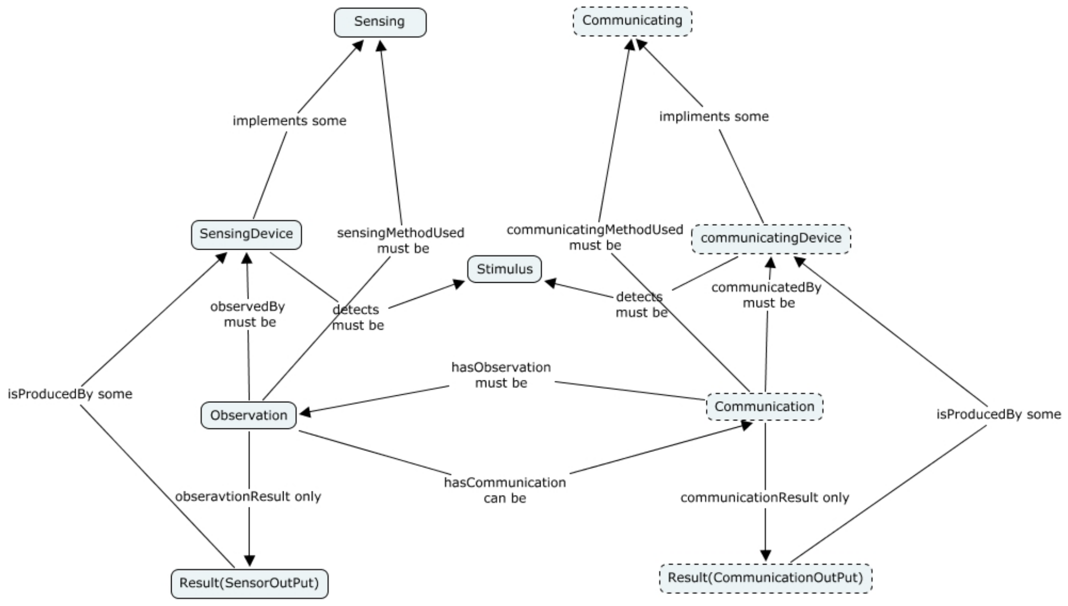


Fig. 2: Stimulus-WSN node-Communication ODP

to its final destination.
*communicationResult*: points to the communication output which is the result of each communication. The communication output has a value, instance of the class CommunicationValue.
*communicationMethodUsed*: points to the method used to perform the communication (an instance of the class Communicating). This procedure can have as an input the sensor output or other kind of data. For example, the communicated data can be an aggregation of a set of sensor outputs. *DUL:includesEvent* points to the stimulus. The stimulus is an event which triggers the communication process. The stimulus can be an incoming request. As soon as a WSN node receives the request, it triggers voluntary a communication process for transmitting data.
*communicatedBy*: points to the communicating device, that is to say the device that performs the communication.

The result of a communication is expressed by an instance of the class CommunicationOutPut. More details about data managed by a node will be presented in the next section.

## 4.3 Data Stream

The data stream is composed of a set of data provided by a source. Several types of sources ranging from a node to set of nodes (WSN) exists. In addition, a data stream may be the result of merging multiple data stream from different providers. In literature,

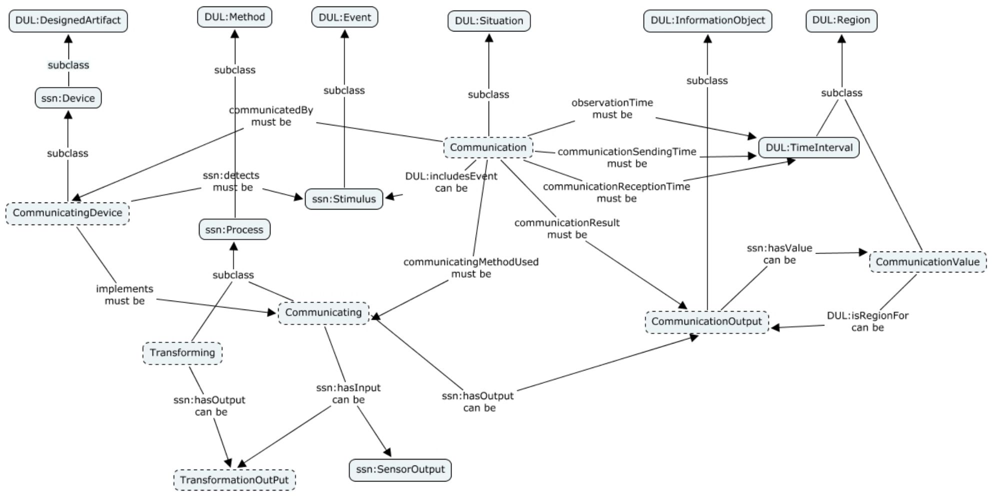


Fig. 3: Communication process in WSSN ontology

there are several definitions of data stream in the databases domain [10], [2] or sensor networks [11]. We defined data stream as: " a sequence of timestamped acquired data arriving continually to the DSS. Agri-environmental communicated WSN data volume depends on certain criteria such as the number of nodes constituting the WSN or the communication frequencies of these nodes". In our case, the data has a moderate size due to a moderate communication performance. As shown in Figure 4, a data stream is a set. Thus the class *DataStream* is a sub class of *DOLCE:Set*. We use the property *DUL:hasConstituent* to link a data stream to its elements.

In addition, the data stream can be separated into two categories: the acquisition and the communication data stream. The two types of data stream will be described in the following sections.

## 4.4 Acquisition data stream

In the WSSN ontology, the acquisition data stream is a set of observations. As shown in Figure 4, this data stream is defined by *AcquisitionDataStream* class. Following the *ssn:observationSamplingTime* property from *Observation*, we find the timestamp associated to the data. The data are instances of the *ObservationValue* class which is a subclass of *DUL:Region*.

## 4.5 Communication Data Stream

The communication data stream is defined as a set of communications. As shown in Figure 5, this data stream is defined by *CommunicationDataStream* class. Following the *observationTime* property from Communication, we find one timestamp associated

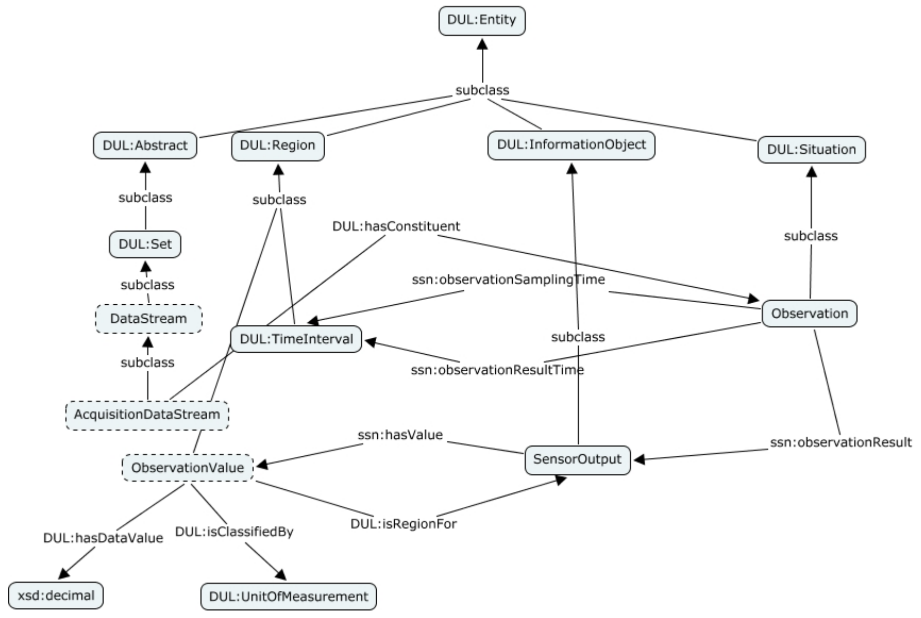


Fig. 4: Acquisitional Data Stream

to the data, the time when the data is acquired by the sensor. The *communicationSendingTime* property point to the time when the data are communicated. The communicated data are instances of the *CommunicationValue* class.

### 4.6 Entities States

The WSN node and the observed phenomenon pass through several states. In WSSN ontology, the state of any entity is represented by the *State* class. This class is defined as a subclass of *DUL:Concept*. A concept is defined by some descriptions and is used to classified entities. An entity is linked to its state by the *hasState* property. The Figure 6 presents the state of WSN node.

## 5 The use of Wireless Semantic Sensor Network Ontology (WSSN)

Our current research aims to limit the number of communications performed by a WSN node in order to enhance the lifetime of the network. In this purpose, a node should adapts its communication policy to its context. The WSSN Ontology will be used to describe the context and the communication policy of the nodes.

In order to validate our approach, we will implement some parts of the scenario about monitoring floods in watershed presented in section 2. The multi-agent platform JADE which can integrate an ontology developed under the Protégé software, is used

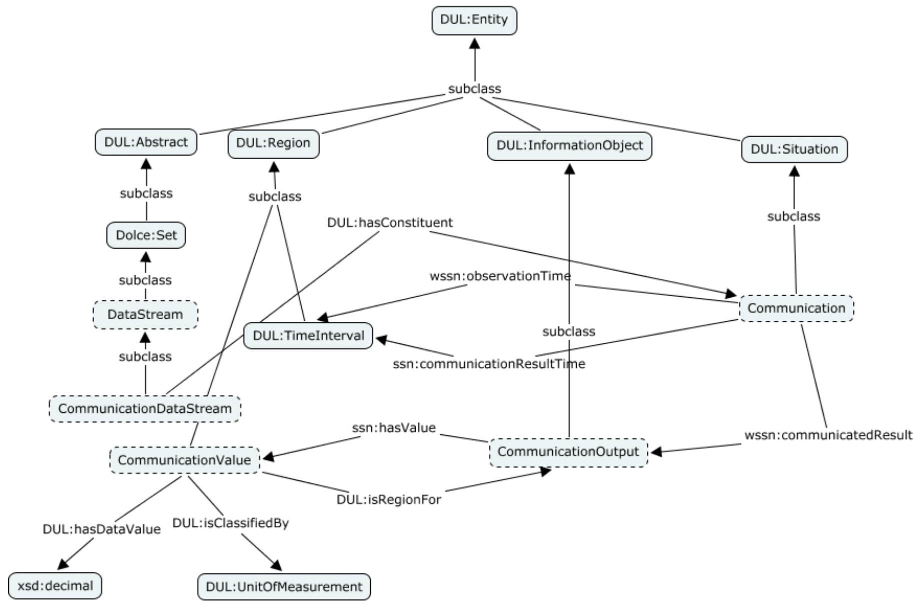


Fig. 5: Communication Data Stream

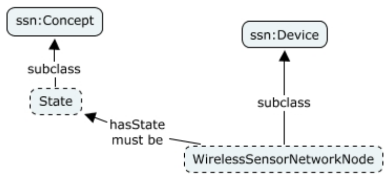


Fig. 6: States

to simulate this scenario. This ontology establishes a shared vocabulary understandable by all JADE agents making possible the definition of message formats between agents. Using JADE, we will simulate a WSN composed of several nodes, each node is a JADE agent. Some parts of our WSSN ontology will be integrated into the simulation in order to generate the communication data stream. Each message between nodes contains indications of the context.

### 5.1 Monitoring floods in a watershed

The WSN used in the flood scenario is composed of different types of nodes such as the “weather” nodes. In any case, a node is composed of a sensing device, an energy device and a communicating device. For example, the sensor of a weather nodes is a pluviometer measuring the amount of precipitation. Its energy device can be a battery associated to a solar panel. Our simulation will be based on two weather nodes. In our experiment, the context contains only the node state representing by its energy level which determine the communication frequency. If its energy level decreases below a fixed threshold, the node will decrease its communication frequency. In order to illustrate this method, we will detail the behaviour of a node in the next section.

## 5.2 Implementation of the scenario

In this section, we will detail the settings of two weather nodes: node 1 and node 2. Only the node 2 implements our approach and adapts its behaviour to the context. The Figure 8 presents the composition and the settings of the node 1 using the WSSN ontology. The default settings of the two nodes are the same:

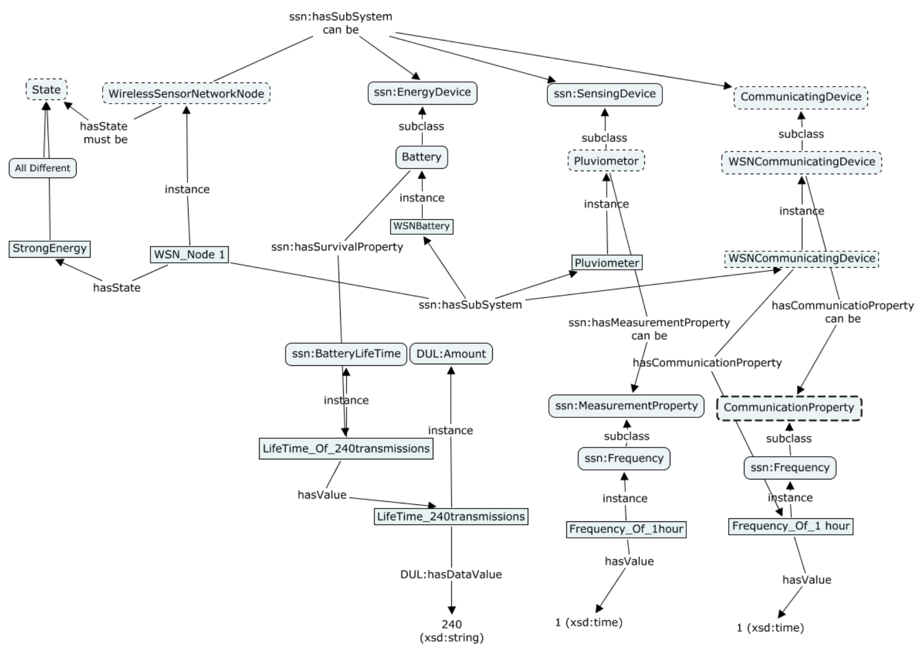


Fig. 7: Setting of the “weather” node ”node 1”

- A node has an energy device which allows to send 240 transmitting packets during its whole lifetime.
- The acquisition and the communication frequencies are equal. A node acquires and communicates one data per hour. Thus, it acquires and communicates 24 data per day. Therefore, the node 1 lifetime will end in the eleventh day.

The node 2 adapts its behaviour to its context described by its energetic state. Its energetic state can be one of the followings:

- “Strong Energy state”, when the current amount of transmitted packets performed by the node is under 120 packets.
- “Average Energy state”, when the current amount of transmitted packets is between 120 and 180 packets.
- “Low Energy state”, when the current amount of transmitted packets performed by the node is above 180 packets.

Depending of its state, the node 2 changes its communication frequency.

- “Strong Energy state”, its communication frequency is equal to the default settings that is one communication per 1 hour.
- “Average Energy state”, its communication frequency is changed to one communication per 2 hours. The acquisition frequency is not changed. So, the communicated data is the aggregation of the two data acquired by sensor during the last 2 hours.
- “Low Energy state”, its communication frequency is changed to one communication per 4 hours. The acquisition frequency is not changed. So, the communicated data is the aggregation of the four data acquired by sensor during the last 4 hours.

## 5.3 Implementation of the scenario

The Figure 8 presents the result of the simulation on JADE platform. The X axis represents the days. The Y axis represents the number of transmitted packets. As expected, the lifetime of the node 1 ends at day 11. The lifetime of the node 2, which implements our contextual approach, ends at day 21. Thus, its lifetime is increased. We notice that this node 2 reduces its communication frequency at day 6 and 11.

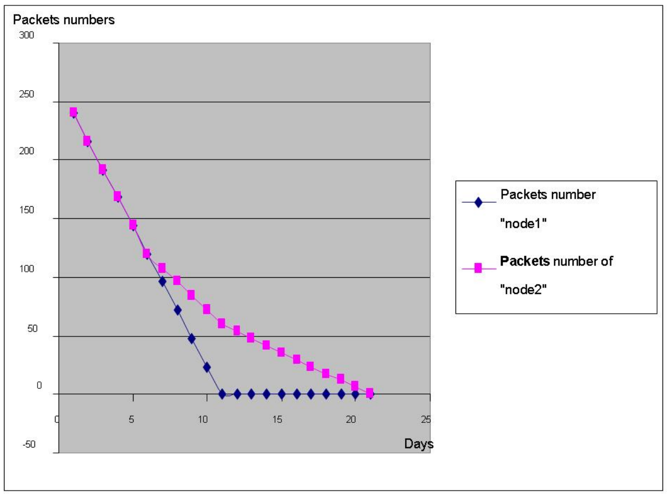


Fig. 8: Comparing the lifetime of the two ”weather” nodes

## 6 Conclusion and Perspectives

Wireless Sensor Networks (WSN) can be designed to collect large amounts of heterogeneous data for monitoring environmental phenomenon. Our aim is to adapt WSN node communication to the context in order to optimize the lifetime of the WSN. Our descriptions of context and WSN characteristics are based on ontologies. In the literature, there are several ontologies that formalize the WSN domain. Based upon a critical analysis of these ontologies, we determine that the Semantic Sensor Network (SSN) ontology is the most suitable to represent most of the WSN issues. However, we highlight that currently no ontology is able to characterize the communication data policy. That is why we propose a new ontology design pattern called Stimulus-WSNnode-Communication. We integrate this pattern in the SSN ontology. We also proposed to enrich the SSN ontology by adding new WSN concepts such as communication, data stream and state. The resulting ontology is named Wireless Semantic Sensor Network ontology (WSSN). In order to evaluate our approach, we implement on the multi-agent platform JADE a simple scenario involving two WSN nodes. Only one node implements our approach and adapts its communication to the context. This node increases consequently its lifetime compare to the other node. The next step will be to enrich our ontology in order to point out the difference between the communicated data and the acquired one. The communicated data can be for example an aggregation (the average value) of a set of acquired data. Finally, we want also to implement on physical WSN nodes our approach and make more complex experiments.

## 7 Acknowledgements

The authors wish to thank Anaïs WERMEILLE for her review of this paper and the "Conseil Régional d'Auvergne" for it financial support.

## References

1. I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, pages 393– 422, 2002.
2. B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. *Proceedings of the 21st Symposium on Principles of Database Systems, ACM Press*, page 1–16, 2002.
3. P. Barnaghi, S. Meissner, M. Presser, and K. Moessner. Sense and sens'ability: Semantic data modelling for sensor networks. *Proceedings of the ICT Mobile Summit 2009.*
4. R. Bendadouche, C. Roussey, G. De Sousa, J. P. Chanet, and K. Mean Hou. Etat de l'art sur les ontologies de capteurs pour une intégration intelligente des données. *INFORSID 2012*, pages 89–104, May 2012.
5. L Bermudez, Delory, E., T O'Reilly, and del Rio J Fernandez. Ocean observing systems demystified. *In OCEANS 2009, MTS/IEEE Biloxi - Marine Technology for Our Future: Global and Local Challenges*, pages 1–7, 2009.
6. M. Calder, obert A. Morris, and F. Peri. Machine reasoning about anomalous sensor data. *Advances in environmental information management*, pages 9–18, 2010.

7. M. Compton, , C. Henson, L. Lefort, and H. Neuhaus. A survey of the semantic specification of sensors. *Proceedings of the 2nd International Workshop on Semantic Sensor Networks, 8th International Semantic Web Conference (ISWC 2009).*
8. M. Compton, P. Barnagh, L. Bermudez, R. Garcia-Castro, O. Corcho, S. Cox, J. Graybeal, M. Hauswirth, C. Henson, A. Herzog, V. Huang, K. Janowicz, W. David Kelsey, D. Le Phuoc, L. Lefort, M. Leggieri, H. Neuhaus, A. Nikolov, A. Page, A. Passant, A. Sheth, and K. Taylor. The ssn ontology of the w3c semantic sensor network incubator group. *Web Semantics: Science, Services and Agents on the World Wide Web*, 0(0), 2012.
9. M. Compton, H. Neuhaus, K. Taylor, and Ki-N. Tran. Reasoning about sensors and compositions. *Proceedings of the Semantic Sensor Networks.*, pages 33–48, 2009.
10. A. Dani and G. Janusz. Conceptual modelling of computations on data streams. *Proceedings of the 2nd Asia-Pacific conference on Conceptual modelling*, 43, 2005.
11. A. L.L. de Aquino, C. M.S Figueiredo, E. F. Nakamura, L. S. Buriol, Antonio A.F. Loureiro, A. O. Fernandes, and C. J. N. Jr Coelho. Data stream based algorithms for wireless sensor network applications. *Proceedings of the 21st International Conference on Advanced Networking and Applications (AINA '07)*, pages 869–876, 2007.
12. M. Eid, R. Liscano, and A. El Saddik. A novel ontology for sensor networks data. *Proceedings of 2006 IEEE International Conference on Computational Intelligence for Measurement Systems and Applications*, pages 75 – 79, 2006.
13. J. Krzysztof and M. Compton. The stimulus-sensor-observation ontology design pattern and its integration into the semantic sensor network ontology. *Proceedings of 3rd International Workshop on Semantic Sensor Networks 2010 (SSN10)*, 2010.
14. O. Madin, S. Bowers, M. Schildhauer, S. Krivov, D. Pennington, and F. Villa. An ontology for describing and synthesizing ecological observation data. *Proceedings of the 5th International Conference on Ecological Informatics ISEI5*, pages 279 – 296, 2007.
15. F. Probst. An ontological analysis of observations and measurements. *Proceedings of the 4th International Conference of Geographic Information Science (GIScience)*, 2006.
16. D.J. Russomanno, C. Kothari, and O. Thomas. Sensor ontologies: from shallow to deep models. *Proceedings of the Thirty-Seventh Southeastern Symposium on System Theory*, pages 107 – 112, March 2005.

# Functional Composition of Sensor Web APIs

Ruben Verborgh[1], Vincent Haerinck[2], Thomas Steiner[3], Davy Van Deursen[1], Sofie Van Hoecke[2], Jos De Roo[4], Rik Van de Walle[1], and Joaquim Gabarro[3]

[1] Ghent University – IBBT, ELIS – Multimedia Lab
Gaston Crommenlaan 8 bus 201, B-9050 Ledeberg-Ghent, Belgium
{ruben.verborgh,rik.vandewalle}@ugent.be
[2] ELIT Lab, University College West Flanders
Ghent University Association, Graaf Karel de Goedelaan 5, 8500 Kortrijk, Belgium
{vincent.haerinck,sofie.van.hoecke}@howest.be
[3] Universitat Politécnica de Catalunya – Department LSI, 08034 Barcelona, Spain
{tsteiner,gabarro}@lsi.upc.edu
[4] Agfa Healthcare, Moutstraat 100, 9000 Ghent, Belgium
jos.deroo@agfa.com

**Abstract.** Web APIs are becoming an increasingly popular alternative to the more heavy-weight Web services. Recently, they also have been used in the context of sensor networks. However, making different Web APIs (and thus sensors) cooperate often requires a significant amount of manual configuration. Ideally, we want Web APIs to behave like Linked Data, where data from different sources can be combined in a straightforward way. Therefore, in this paper, we show how Web APIs, semantically described by the light-weight format RESTdesc, can be composed automatically based on their functionality. Moreover, the composition process does *not* require specific tools, as compositions are created by generic Semantic Web reasoners as part of a proof. We then indicate how the composition in this proof can be executed. We describe our architecture and implementation, and validate that proof-based composition is a feasible strategy on a Web scale. Our measurements indicate that current reasoners can integrate compositions of more than 200 Web APIs in under one second. This makes proof-based composition a practical choice for today's Web APIs.

**Keywords:** Semantic Web, Web APIs, sensors, composition, reasoning

## 1 Introduction

Sensors are gradually finding their way to the world of Web APIs. The REST principles of resource-oriented API design, as defined by Fielding [15], are gaining momentum on the Web of Things [17,40]. On top of this, Semantic Web technologies are then used to make the sensor data meaningful to machines [34,35]. A uniform way to access semantic sensor data is not the endpoint: machines need a way to select *what* sensor they need for a specific situation. This is the task of semantic Web API descriptions [33,39,42], which capture the functionality of

Web APIs in a semantic format. However, much more innovate power becomes available when different sensors are *combined* to deliver new and unprecedented functionality. Unfortunately, today, this involves a substantial amount of manual work: while Web APIs bare the potential to be composed straightforwardly, they lack the semantics to do this in an automated way [26].

The present paper addresses this issue by introducing a method to automatically compose Web APIs. On the one hand, this allows a faster and easier development of Web applications. On the other hand, it enables on-demand solutions for specific problems and questions, for which it would be impractical or infeasible to create ad-hoc solutions manually. Furthermore, the proposed method does *not* require specific tools or software, but rather works with generic Semantic Web reasoners. This ensures the maintainability and generalizability of the solution towards the future.

In the end, we want to enable for Web APIs what Linked Data [7] does for data: the automated integration of various, heterogeneous sources with the help of semantics, leading to composability. Web APIs are an excellent match for this, because of the many parallels between the Linked Data and REST principles [15,20,44]. Also, Web APIs allow us to move beyond the traditional input/output-based matching from the Web services world, instead delivering integration based on functionality.

This paper is structured as follows. In Section 2, we describe related work on Web API composition. Section 3 introduces a use case and explains the concepts of reasoning-based composition, followed by the principles of composition execution. Section 4 proposes an architecture and implementation to perform composition and execution in an automated way. This approach is evaluated in Section 5, and Section 6 concludes the paper by placing Web API composition in the broader Web context and provides an overview of future work.

## 2 Related Work

In the next subsections, we discuss related work in the fields of Semantic Web Service description, Web API description, and Semantic Web reasoners.

### 2.1 Semantic Web Service Description and Composition

Semantic Web service description has been a topic of intense research for at least a decade. There are many approaches to service description with different underlying service models. OWL-S [31] and WSMO [25] are the most known Semantic Web Service description paradigms. They both allow to describe the high-level semantics of services whose message format is WSDL [12]. Though extension to other message formats is possible, this is rarely seen in practice. Semantic Annotations for WSDL (SAWSDL [24]) aim to provide a more light-weight approach for bringing semantics to WSDL services. Composition of Semantic Web services has been well documented, but all approaches require specific software [18,19,32] and none of the solutions have found widespread adoption.

## 2.2 Web API Description

In recent years, more and more Web API description formats have been evolving. The link between the Semantic Web and Web APIs has been explored many times [37]. Linked Open Services (LOS, [33]) expose functionality on the Web using Linked Data technologies, namely HTTP [14], RDF [21], and SPARQL [38]. Input and output parameters are described with SPARQL graph patterns embedded inside RDF string literals to achieve quantification, which RDF does not support natively. Linked Data Services (LIDS, [39]) define interface conventions that are compatible with the Linked Data principles [7] and are supported by a lightweight formal model. RESTdesc [42] is a hypermedia API description format that describes Web APIs' functionality in terms of resources and links.

The Resource Linking Language (ReLL, [1]) features media types, resource types, and link types as first class citizens for descriptions. The RESTler crawler [1] finds RESTful services based on these descriptions. The authors of ReLL also propose a method for ReLL API composition [2] using Petri nets to describe the machine-client navigation. However, automatic, functionality-based composition is not supported.

Several methods aim to enhance existing technologies to deliver annotations of Web APIs. HTML for RESTful Services (hRESTS, [22]) is a microformat to annotate HTML descriptions of Web APIs in a machine-processable way. SA-REST [16] provides an extension of hRESTS that describes other facets such as data formats and programming language bindings. MicroWSMO [23,29], an extension to SAWSDL that enables the annotation of RESTful services, supports the discovery, composition, and invocation of Web APIs. The Semantic Web sErvices Editing Tool (SWEET, [27]) is an editor that supports the creation of mashups through semantic annotations with MicroWSMO and other technologies. A shared API description model, providing common grounds for enhancing APIs with semantic annotations to overcome the current heterogeneity, has been proposed in the context of the SOA4All project [28].

## 2.3 Semantic Web Reasoning

Pellet [36] and the various Jena [11] reasoners are likely the most-known examples of publicly available Semantic Web reasoners. Pellet is an OWL DL [8] reasoner, while the Jena framework offers transitive, RDFS [10], OWL [8], and rule reasoners. The rule reasoner is the most powerful, but uses a rule language that is specific to Jena and therefore not interchangeable.

Another category of reasoners use the Notation3 language (N3, [4]), a small superset of RDF that adds support for formulas and quantification, providing a logical framework for inferencing [5]. The initial N3 reasoner is the forward-chaining cwm [3], which is a general-purpose data processing tool for RDF, including tasks such as querying and proof-checking. Another important N3 reasoner is EYE [13], whose features include backward-chaining and high performance. A useful capability of both N3 reasoners is their ability to generate and exchange *proofs*, which can be used for software synthesis or API composition [30,45].

## 3 Concept

### 3.1 Example Use Case

To illustrate the theoretical framework, we first introduce an example that will be carried through the paper. The problem statement is as follows:

> *A user wants to reserve a nearby restaurant. He will take a table outside if the weather allows it.*

To solve this problem, the following Web APIs (and several others) are available:

| | |
|---|---|
| **Location API** | gets the current location; |
| **Temperature API** | reads a temperature sensor near a specific location; |
| **Pressure API** | reads an air pressure sensor near a specific location; |
| **Restaurant API** | makes a restaurant reservation. |

If we were to solve this problem manually with the given APIs, a straightforward solution would be to combine them as in Fig. 1. This graph shows how, starting from the **I**nitial state *(the user and his preferences)*, we can reach the **G**oal state *(inside or outside reservation in a nearby restaurant, depending on the weather)*. First, the **L**ocation API needs to look up the current location of the user. Then, the **T**emperature and **P**ressure APIs can be invoked with this location. Based on their results, the details of the reservation can be completed. Finally, the **R**estaurant API uses these parameters to make the reservation, thereby satisfying the **G**oal. The order in which the execution happens is governed by the *dependencies* between the APIs, as depicted in Fig. 1.

This composition can either serve as a one-time solution for a specific situation, or be reused in different scenarios, in which case it becomes a Web API itself. In any case, the goal is to create and execute this composition in a fully automated way. This process will be explained in the next subsections.

### 3.2 Universally Representing Compositions

In order to automatically create compositions, we need a universal way to represent these compositions and the APIs of which they consist, so machines can manipulate them easily. In essence, a composition can be seen as a logic entailment, since the **I**nitial state must entail the **G**oal state:

$$\mathbf{I}(\mathit{composition}) \Rightarrow \ldots \Rightarrow \ldots \Rightarrow \mathbf{G}(\mathit{composition})$$

This perfectly aligns with the notion of dependencies, since the satisfaction of **G** depends on the satisfaction of **I**. Analogously, each API can be seen as an implication. For instance, given a location of a place, the **T**emperature API allows to obtain its temperature. In that sense, the **T**emperature API fulfills the implication between a **location** and its **temperature**:

$$\mathbf{T} \quad \equiv \quad \mathbf{location}(\mathit{place}) \Rightarrow \mathbf{temperature}(\mathit{place})$$

That way, Web APIs can be represented as implications, and compositions as a chain of implications that leads to entailment.

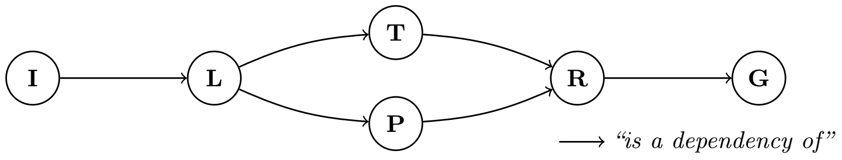


**Fig. 1.** By combining the **L**ocation, **T**emperature, **P**ressure, and **R**estaurant APIs, and respecting their dependencies, we can reach the **G**oal from the **I**nitial state.

## 3.3 Deriving Compositions

Not only are implications a straightforward representation to manipulate, the question whether we can solve a certain problem becomes a matter of entailment: *does the **I**nitial state entail the **G**oal state?* However, more important than whether the problem can be solved, is *how* it can be solved, in other words, which APIs are necessary to find the answer. In the logic world, this comes down to providing the *proof* of the entailment: ***why** does the **I**nitial state entail the **G**oal state?* For the restaurant example, a proof might look like this[1]:

$$\begin{aligned}
& & \mathbf{I} &\Rightarrow \mathbf{preferences}(\mathit{user}) & (1)\\
\mathbf{L} &\equiv & \mathbf{preferences}(\mathit{user}) &\Rightarrow \mathbf{location}(\mathit{place}) & (2)\\
\mathbf{T} &\equiv & \mathbf{location}(\mathit{place}) &\Rightarrow \mathbf{temperature}(\mathit{place}) & (3)\\
\mathbf{P} &\equiv & \mathbf{location}(\mathit{place}) &\Rightarrow \mathbf{pressure}(\mathit{place}) & (4)\\
\mathbf{R} &\equiv & \mathbf{demand}(\mathit{appointment}) &\Rightarrow \mathbf{reservation}(\mathit{appointment}) & (5)\\
& & \mathbf{reservation}(\mathit{appointment}) &\Rightarrow \mathbf{G} & (6)\\
& & \mathrm{consequent}(1) &\Rightarrow \mathrm{antecedent}(2) & (7)\\
& & \mathrm{consequent}(2) &\Rightarrow \mathrm{antecedent}(3) & (8)\\
& & \mathrm{consequent}(2) &\Rightarrow \mathrm{antecedent}(4) & (9)\\
& & \mathrm{consequent}(3), \mathrm{consequent}(4) &\Rightarrow \mathrm{antecedent}(5) & (10)\\
& & \mathrm{consequent}(5) &\Rightarrow \mathrm{antecedent}(6) & (11)\\
\mathbf{I} &\Rightarrow & (1),(7),(8),(9),(10),(11),(6) &\Rightarrow \mathbf{G} & (12)
\end{aligned}$$

In this proof, we immediately recognize the structure of the composition as depicted in Fig. 1. The characterisations of the **I**nitial and **G**oal states can be seen in (1) and (6) respectively, and the definitions of the APIs in (2) to (5). The dependency relations are contained in (7) to (11). For instance, the fact that the **L**ocation API is a dependency of the **T**emperature API in Fig. 1 corresponds to the implication in (8). Finally, (12) contains the combined proof elements that explain *why* **I** entails **G**, effectively generating the whole composition. This indicates how the proof of entailment explains how APIs can be combined to deliver the requested functionality. As a result, the proof is in fact an alternate and automatically reconstructed representation of the composition graph.

[1] Note that certain background knowledge is assumed (ontologies and/or rules).

### 3.4 Executing compositions

Once we have obtained a proof, we can execute all Web API calls within it. That way, it becomes a *pragmatic proof*, in which all of the inferences will actually be carried out. The execution order is governed by the dependencies between the APIs. Because the proof starts with the **I**nitial state and ends with the **G**oal state—and cycles are impossible within proofs—we are sure at each step of the execution to find at least one API whose dependencies have been resolved. This is obvious in the example proof, since every proof step only refers to steps with a lower number.

As a result, for the first API call, all parameters are known in advance. In the example, the sole option is to start with the **L**ocation API, since this is the only API with no other dependencies than the **I**nitial state. Its parameter, an address, is already known and will be present in the proof. The situation is different for the **T**emperature and **P**ressure APIs: they both depend on the **L**ocation API and have the resulting geographical coordinates as a parameter. This value is unknown at the time the proof is constructed. However, the proof does tell us how this value can be obtained: it is the result of the **L**ocation API invocation.

There are two ways to deal with these unknown values: A first approach is to do bookkeeping during the execution. Since the proof tells us the dependencies between the APIs, we can assign the values received from previous API calls to the associated variables. A second approach is to repeat the reasoning process after each execution of an API call. The **I**nitial state is thereby augmented with the information returned by the API call, giving rise to a new composition with fewer steps. The benefit is that this approach also works if the API provides a result that is different than expected.

Now that we understand how to manually compose and execute Web APIs, we will have a look at the automation of the process.

# 4 Architecture and Implementation

### 4.1 Overview

In this section, we describe how the concepts put forward in Section 3 have been automated and realized in a software platform. Fig. 2 shows an overview of the platform's architecture. It consists of three main components:

– the **reasoner**, which generates the composition;
– the **executor**, which governs the execution of compositions;
– the **client**, offering the interface to coordinate the above two components.

The platform expects the following inputs:

– various **APIs** and corresponding **descriptions**;
– a request, consisting of the **I**nitial and **G**oal states.

The APIs and descriptions are likely to be part of a reusable collection, for instance, an API repository. In contrast, the **I**nitial and **G**oal states will probably differ between invocations.

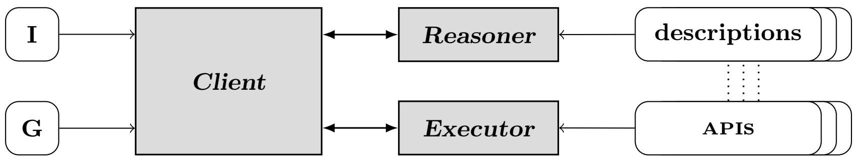


**Fig. 2.** The principal platform architecture, showing the client that interacts with the reasoner and executor.

Upon receiving a request, the client instructs the reasoner to verify whether the **I**nitial state entails the **G**oal state, reckoning with the provided API descriptions. At the same time, if this entailment holds, the client will ask for the proof. This proof will contain all details needed by the executor to actually invoke the APIs and obtain the desired result. If we apply this to the restaurant example, the available APIs are **L**ocation, **T**emperature, **P**ressure, **R**estaurant, and possibly many others, all of them with their corresponding description. To start the process, the user gives the address of his preferred restaurant to the client, along with the instruction to reserve this restaurant on the next sunny day for his nearby friends. In Subsections 4.3 and 4.4, we will zoom in on the implementation of the reasoner and the executor, but first, Subsection 4.2 will introduce and justify the description technology used in this implementation.

### 4.2 Description Technology

The first decision to make is what technology will describe the Web APIs, since the platform can only be as powerful as the expressivity of the description method permits. Candidate technologies should possess these characteristics:

- support **REST** or **hypermedia** APIs [15] (as opposed to RPC-style services);
- explain the **functionality** of the API in a machine-processable way (as opposed to detailing only input and output parameters);
- allow **composition** of any number of APIs.

For the implementation, we selected RESTdesc [42,43], since it explicitly targets hypermedia APIs and focuses on functionality. Furthermore, RESTdesc descriptions are expressed in Notation3 (N3, [4]), a Semantic Web logic language put forward by Tim Berners-Lee, which allows *generic* Semantic Web reasoners that take N3 as input to interpret RESTdesc descriptions directly. As a result, RESTdesc-described APIs can easily be composed with the proof-based technique.

Listing 1 displays the RESTdesc description of the **R**estaurant API. RESTdesc descriptions are N3 rules whose antecedent contains the preconditions and whose consequent contains the request and postconditions . The hypermedia nature of RESTdesc can be clearly seen: starting from a restaurant resource that has a `reservationList` link to a reservations resource ❶, a client can `POST` ❸ reservation details ❷ to attach a reservation to the restaurant resource ❹. RESTdesc descriptions indeed focus on resources and the links between them, which makes them an excellent fit to describe hypermedia APIs.

```
@prefix resto: <http://example.org/restaurant#>.
@prefix http:  <http://www.w3.org/2011/http#>.
{
  ?restaurant resto:reservationList ?reservations.  ❶
  ?place resto:isOutside ?outside.
  ?day resto:hasDate ?date.
}
=>
{
  _:request http:methodName "POST";  ❸
            http:requestURI ?reservations;
            http:body (?date ?outside);
            http:resp [ http:body ?reservation ].

  ?restaurant resto:hasReservation ?reservation. ❹
  ?reservation resto:onDate ?date;
               resto:place [ resto:isOutside ?outside ] .
}.
```

**Listing 1.** This example RESTdesc description explains the part of the **R**estaurant API that allows to make a reservation.

## 4.3 Reasoner

Since RESTdesc can be interpreted by generic N3 reasoners, we do not need to implement a specific reasoner for RESTdesc composition. This offers a considerable benefit in terms of portability and sustainability. Performance-wise, this choice is also beneficial, because several implementations of N3 reasoners exist [5], giving rise to an ongoing competition of reasoner developers who continue to improve reasoner performance. Reusing the implementation efforts and experience of the wider reasoning community is a faster and more durable decision than developing and maintaining a specific composition algorithm from the ground up.

We have tested our implementation with the EYE [13] and cwm [3] reasoners, both of which have the ability to generate a proof, such as the one we have crafted manually in Subsection 3.3. This proof must be understood by the client, because it represents the composition the executor will run.

Since the full proof of the example restaurant composition would be too lengthy to discuss, we will use another example from the sensor domain.[2] In the example, the background knowledge is that all temperature sensors are sensors, and that *MySensor* is a temperature sensor. In N3, this is expressed as:

```
<MySensor> a s:TemperatureSensor.
{ ?something a s:TemperatureSensor. } => { ?something a s:Sensor. }.
```

The reasoner also needs a goal query. In this case, we will ask to find all sensors:

```
{ ?x a s:Sensor. } => { ?x a s:Sensor. }.
```

Note that this is *not* an inference, but a query similar to SPARQL CONSTRUCT. The answer to this query is, after inference:

```
<MySensor> a s:Sensor.
```

To generate the proof, we invoke the reasoners with a command similar to:

```
eye sensors.n3 --query query.n3
cwm sensors.n3 --think --filter=query.n3 --why
```

```
@prefix s:   <sensors#>.
@prefix var: <var#>.
@prefix r:   <http://www.w3.org/2000/10/swap/reason#>.
@prefix n3:  <http://www.w3.org/2004/06/rei#>.

[ a r:Proof, r:Conjunction;     Ⓟ
  r:component
    [ a r:Inference;                       Ⓠ
      r:gives {<MySensor> a s:Sensor.};
      r:evidence (
        [ a r:Inference;                          Ⓡ
          r:gives {<MySensor> a s:Sensor};
          r:evidence ([ a r:Extraction;
                        r:gives {<MySensor> a s:TemperatureSensor.};
                        r:because [ a r:Parsing;
                                    r:source <sensors.n3>]]);
          r:binding [ r:variable [ n3:uri "var#x0"];
                      r:boundTo  [ n3:uri "MySensor"]];
          r:rule [ a r:Extraction;
                   r:gives {@forAll var:x0.
          {var:x0 a s:TemperatureSensor.} => {var:x0 a s:Sensor.}.};
                   r:because [ a r:Parsing; r:source <sensors.n3>]]]);
      r:binding [ r:variable [ n3:uri "var#x0"];
                  r:boundTo  [ n3:uri "MySensor"]];
      r:rule [ a r:Extraction;
               r:gives {@forAll var:x0. {var:x0 a s:Sensor.}
                                         => {var:x0 a s:Sensor.}.};
               r:because [ a r:Parsing; r:source <query.n3>]]];
  r:gives {
    <MySensor> a s:Sensor.
  }].
```

**Listing 2.** This example proof illustrates the important proof concepts.

Listing 2 shows the resulting proof. As we can see, a `Proof` Ⓟ consists of a `Conjunction` of components that `gives` the answer to our query. In this example, the only component is an `Inference` Ⓠ that applies the query rule to the statement "`<MySensor> a s:TemperatureSensor.`", which is done by `binding` the `?x` variable to `<MySensor>`. Then of course, the question is where this statement came from. The `evidence` relation explains it as another `Inference` Ⓡ, for which the "`{ ?s a s:TemperatureSensor. } => { ?s a s:Sensor. }`" rule was applied to the statement "`<MySensor> a s:TemperatureSensor.`", `binding` the `?s` variable to `<MySensor>`, and thereby leading to the desired result. The proof then indicates that all further evidence are `Extractions` as from `Parsing`.

This explanation reveals the dependency-oriented nature of proofs: the validity of the proof depends on the validity of an inference, which in turn depends on another inference, which ultimately depends on parsing an original source. This perfectly aligns with the dependencies of compositions. Since every Web API in the composition will be described by RESTdesc, which captures functionality in inference rules, the inferences in the proof will correspond to API invocations.

[2] The example is available at `http://notes.restdesc.org/2012/sensors/`.

Furthermore, the variable bindings detail the parameters that need to be used for each invocation. Eventually, the parsed source will correspond to the **I**nitial state, and the result of the proof will be the **G**oal state. To create compositions, it is therefore sufficient to start a reasoner with a command similar to:

```
eye initial.ttl descriptions.n3 --query goal.n3
cwm initial.ttl descriptions.n3 --think --filter=goal.n3 --why
```

Note that Web API calls do not have to be the only inferences in the proof: traditional implications (such as ontological constructs) can be carried out, too. This opens up the possibility to combine results from different API calls, and to compose APIs that have been expressed in different ontologies.

## 4.4 Executor

In order to execute a composition, the executor does not only need to know what APIs to execute, but also all details of what each HTTP request to these APIs should like. Fortunately, by using RESTdesc descriptions as part of the proof process, the variables in the descriptions will be instantiated with concrete values. For example, as part of the proof, the description from Listing 1 will be instantiated as in Listing 3.

As explained in Subsection 3.4, not all parameter values are known in advance. In Listing 3, we can see that the concrete URI of the restaurant and the reservation list have been instantiated: the executor thus already knows that it will have to perform a POST request to the URI `http://resto.example.org/reservations/`. It also know the date, which is part of the **I**nitial state. However, it does not know yet the concrete value of `?outside`, so it represents it as a blank node instead. Because this blank node will be linked to blank nodes in the instantiation of the **T**emperature and **P**ressure APIs, the executor understands that it has to use the output of these APIs as the input of the **R**estaurant API.

Since at each step at least one API request will be fully instantiated, the executor will always be able to proceed. Partially instantiated requests can be completed with the result of executed API requests, either by the executor or by performing subsequent reasoner runs with the new data.

```
<http://resto.example.org/> resto:reservationList
                              <http://resto.example.org/reservations/>.
_:place1 resto:isOutside _:outside1.
_:day1 resto:hasDate "2012/11/11".

_:request1 http:methodName "POST";
           http:requestURI <http://resto.example.org/reservations/>;
           http:body ("2012/11/11" _:outside1);
           http:resp [ http:body _:reservation1 ].

<http://resto.example.org/> resto:hasReservation _:reservation1.
_:reservation1 resto:onDate _:date1;
               resto:place [ resto:isOutside _:outside1 ].
```

**Listing 3.** The instantiation of the **R**estaurant API description by the reasoner.

Another important aspect of the executor is that it is not limited to a certain content representation format. While the RESTdesc descriptions are expressed in N3 or in RDF (when instantiated), the Web APIs do not have to produce or consume RDF. The executor acts as a hypermedia client that negotiates content types at runtime. This is why RESTdesc purposely does not describe the format of the exchanged messages. Such flexibility enables the APIs to communicate in any format the executor supports. For instance, the **T**emperature sensor could interact using JSON, while the **L**ocation sensor could provide answers in GML [35].

## 5 Evaluation

The crucial statement in this paper is that generic reasoners are able to create Web API compositions in an automated way. Our evaluation verifies whether this concept works on a Web scale, *i.e.*, with a large number of APIs and associated descriptions, assuming that any given task will require a combination of a relatively small subset of all available APIs. To this extent, we have developed a benchmark framework[3] that consists of two main components:

- a **description generator**, which is able to generate an arbitrary-length chain of RESTdesc descriptions that can form a composition;
- an **automated benchmarker**, which tests a reasoner for compositions of varying lengths and complexity.

These variations in complexity are obtained by modifying the number of dependencies between different descriptions. In the simplest case, every API exactly depends on one previous API in the chain. More complex cases involve multiple dependencies. We have tested three scenarios for $n$ going from 2 to 1024:

1. a chain of $n$ APIs with 1 dependency between each of them;
2. a chain of $n$ APIs with 2 dependencies between each of them;
3. a chain of $n$ APIs with 3 dependencies between each of them.

Additionally, we looked at the composition of a 1-dependency chain of 32 APIs, in presence of a growing number of "dummy" APIs that are meant to test how fast the reasoner can discriminate between relevant and non-relevant descriptions.

It is important to understand that most real-world scenarios will be a mixture of the above situations: compositions generally consist of API calls with a varying number of dependencies, created in presence of a non-negligible number of descriptions that are irrelevant to the composition under construction. Therefore, by measuring these aspects independently, we can predict how well a reasoner will perform in those situations.

The measurements have been split in *parsing*, *reasoning*, and *total* times. Parsing represents the time during which the reasoner internalizes the input into an in-memory representation. This was measured by presenting the inputs

[3] The RESTdesc composition benchmark suite is freely available for download at `http://github.com/RubenVerborgh/RESTdesc-Composition-Benchmark`.

| #descriptions ($n$) | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 | 1024 |
|---|---|---|---|---|---|---|---|---|---|---|
| $n$ APIs (1 dep.) | | | | | | | | | | |
| parsing | 53 | 53 | 54 | 55 | 58 | 64 | 78 | 104 | 161 | 266 |
| reasoning | 2 | 4 | 5 | 7 | 10 | 20 | 43 | 77 | 157 | 391 |
| total | 55 | 57 | 58 | 62 | 68 | 84 | 121 | 181 | 318 | 657 |
| $n$ APIs (2 deps.) | | | | | | | | | | |
| parsing | 53 | 53 | 59 | 56 | 60 | 67 | 85 | 117 | 184 | 331 |
| reasoning | 3 | 6 | 69 | 41 | 45 | 56 | 84 | 174 | 461 | 1,466 |
| total | 56 | 59 | 128 | 97 | 104 | 123 | 169 | 292 | 645 | 1,797 |
| $n$ APIs (3 deps.) | | | | | | | | | | |
| parsing | 53 | 53 | 68 | 56 | 61 | 70 | 90 | 129 | 208 | 371 |
| reasoning | 3 | 12 | 45 | 49 | 61 | 99 | 200 | 544 | 1,639 | 6,493 |
| total | 57 | 66 | 114 | 105 | 122 | 169 | 290 | 673 | 1,847 | 6,864 |
| **32 APIs, $n$ dummies** | | | | | | | | | | |
| parsing | 59 | 60 | 62 | 64 | 65 | 72 | 88 | 134 | 170 | 278 |
| reasoning | 10 | 10 | 10 | 10 | 11 | 12 | 12 | 12 | 12 | 14 |
| total | 69 | 70 | 72 | 74 | 76 | 84 | 100 | 146 | 182 | 292 |

**Table 1.** The EYE reasoner manages to create even lengthy compositions in a timely manner (*average times of 50 trials, expressed in milliseconds*).

to the reasoner, without asking for any operation to be performed on them. Since the parsing step can often be cached and reused in subsequent iterations, it is worthwhile evaluating the actual reasoning time separately. Parsing and reasoning together make up for the total time.

Table 1 shows the benchmark results of the EYE reasoner, as generated on a consumer computer (2.66 GHz Intel Core i7, 4GB RAM). The results in the first column teach us that a start-up overhead of $\approx 50ms$ is involved for starting the reasoner. This includes process starting costs and is highly machine-dependent. When looking at the parsing times for all 4 cases, we see that they increase linearly with the number and size of the descriptions, as expected from any parser. In each of the benchmarks for $n$ APIs with 1 to 3 dependencies, we see that the composition time increases linearly with the number of descriptions. As a consequence, the total time also increases linearly.

Finally, if we look at the composition of 32 APIs in presence of dummy APIs, we can see that the influence of the dummies on the reasoning time is minimal. Compared to the case where 32 APIs with one dependency were composed without dummies, we see that most overhead is introduces by the parsing of the extra descriptions, which can be cached. The reasoning time remains fairly close to the original time, even in presence of a large number of dummies.

## 6 Conclusion and Future Work

In this paper, we presented a novel approach to compose Web APIs, integrating sensors APIs with others. We showed how the description format RESTdesc enables functionality-based compositions, which are automatically created using the proof-generating capabilities of a generic Semantic Web reasoner. We described the architecture and implementation of a platform that is able to compose and execute these Web API compositions. It features a client that accepts API descriptions, an **I**nitial state and a **G**oal, which a reasoner then uses to create a composition that is carried out by an executor.

Based on the results of our evaluation, we can say that proof-based composition of Web APIs by generic reasoners is a feasible approach, even on a Web scale where thousands of sensors could be involved. Reasoning time evolves linearly with the number and size of the descriptions, with response times far below one second for typical composition sizes, even in presence of a large amount of descriptions. Moreover, the reasoner-based approach is much more sustainable than composition methods that are tied to a specific description method.

If we situate RESTdesc composition in the Semantic Web Stack [9] in Fig. 3, we see it is based on the fundamental elements. As a light-weight Web API description method, RESTdesc strives to express the functionality of APIs with the goal of integration and composability, maximizing technology reuse. In that way, we hope to put one of the steps required to bring Web APIs towards the high level of integration and composability of today's Linking Open Data cloud [6].

In the future, we want to improve composition further by taking optimization strategies into account. If several compositions can provide similar solutions, we need a mechanism to select the optimum, given a specified set of constraints. Our work on defining quality parameters of APIs and compositions [41] shows part of our progress in this field. An important challenge is dealing with the heterogeneity of Web APIs and the many different representations they offer. While RESTdesc is not tied to a specific representation format, the executor must be able to deal with a variety of formats across the Web. We strongly believe in content type negotiation, since the Web has been and always will be a diverse environment. Finally, we aim to improve the client so that it becomes usable by a wide audience. We consider browser-based implementations for computers and mobile devices, to bring the power of Web API composition to everyone.

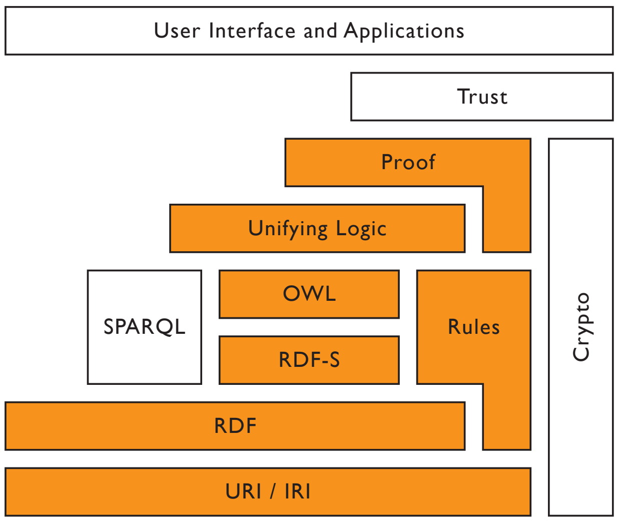


**Fig. 3.** Proof-based Web API composition, based on RESTdesc functional descriptions, maximizes technology reuse in the Semantic Web Stack.

**Acknowledgments** The authors would like to thank Maria Maleshkova for proofreading this paper thoroughly and providing us with invaluable suggestions.

The described research activities were funded by Ghent University, the Interdisciplinary Institute for Broadband Technology (IBBT), the Institute for the Promotion of Innovation by Science and Technology in Flanders (IWT), the Fund for Scientific Research Flanders (FWO Flanders), and the European Union. This work was partially supported by the European Commission under Grant No. 248296 FP7 (I-SEARCH project). Joaquim Gabarró is partially supported by TIN–2007–66523 (FORMALISM), and SGR 2009–2015 (ALCOM).

## References

1. Alarcón, R., Wilde, E.: RESTler: crawling RESTful services. In: Proceedings of the 19th international conference on World Wide Web. pp. 1051–1052. ACM (2010), http://doi.acm.org/10.1145/1772690.1772799
2. Alarcón, R., Wilde, E., Bellido, J.: Hypermedia-driven RESTful service composition. In: Service-Oriented Computing, Lecture Notes in Computer Science, vol. 6568, pp. 111–120. Springer (2011), http://dx.doi.org/10.1007/978-3-642-19394-1_12
3. Berners-Lee, T.: cwm. Semantic Web Application Platform (2000–2009), available at http://www.w3.org/2000/10/swap/doc/cwm.html
4. Berners-Lee, T., Connolly, D.: Notation3 (N3): A readable RDF syntax. W3C Team Submission (Mar 2011), http://www.w3.org/TeamSubmission/n3/
5. Berners-Lee, T., Connolly, D., Kagal, L., Scharf, Y., Hendler, J.: N3Logic: A logical framework for the World Wide Web. Theory and Practice of Logic Programming 8(3), 249–269 (2008), http://arxiv.org/abs/0711.1533
6. Bizer, C., Jentzsch, A., Cyganiak, R.: State of the LOD cloud (2011), http://www4.wiwiss.fu-berlin.de/lodcloud/state
7. Bizer, C., Heath, T., Berners-Lee, T.: Linked Data – The Story So Far. International Journal On Semantic Web and Information Systems 5(3), 1–22 (2009), http://tomheath.com/papers/bizer-heath-berners-lee-ijswis-linked-data.pdf
8. Bock, C., Fokoue, A., Haase, P., Hoekstra, R., Horrocks, I., Ruttenberg, A., Sattler, U., Smith, M.: OWL 2 Web Ontology Language. W3C Recommendation (Oct 2009), http://www.w3.org/TR/owl2-syntax/
9. Bratt, S.: Semantic Web, and other technologies to watch. INCOSE International Workshop (Jan 2007), available at http://www.w3.org/2007/Talks/0130-sb-W3CTechSemWeb/#(24)
10. Brickley, D., Guha, R.V.: RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation (Feb 2004), http://www.w3.org/TR/rdf-schema/
11. Carroll, J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., Wilkinson, K.: Jena: implementing the Semantic Web recommendations. In: Proceedings of the 13th international World Wide Web conference. pp. 74–83. ACM (2004), www.hpl.hp.com/techreports/2003/HPL-2003-146.pdf
12. Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.: Web Services Description Language (WSDL). W3C Note (Mar 2001), http://www.w3.org/TR/wsdl
13. De Roo, J.: Euler proof mechanism (1999–2012), available at http://eulersharp.sourceforge.net/
14. Fielding, R.T., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T.: Hypertext Transfer Protocol – HTTP/1.1 (Jun 1999), http://www.ietf.org/rfc/rfc2616.txt

15. Fielding, R.T., Taylor, R.N.: Principled design of the modern Web architecture. Transactions on Internet Technology 2(2), 115–150 (May 2002), http://dl.acm.org/citation.cfm?id=514185
16. Gomadam, K., Ranabahu, A., Sheth, A.: SA-REST: Semantic Annotation of Web Resources. W3C Member Submission, http://www.w3.org/Submission/SA-REST/
17. Guinard, D., Trifa, V., Wilde, E.: A resource-oriented architecture for the Web of Things. In: Internet of Things (Dec 2010), http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=5678452
18. Hashemian, S., Mavaddat, F.: A graph-based approach to Web services composition. In: Proceedings of the 2005 Symposium on Applications and the Internet. pp. 183–189 (2005), http://www.cin.ufpe.br/~redis/intranet/bibliography/middleware/hashemian-composition05.pdf
19. Hristoskova, A., Volckaert, B., De Turck, F.: Dynamic composition of semantically annotated Web services through QoS-aware HTN planning algorithms. In: Fourth International Conference on Internet and Web Applications and Services. pp. 377–382. IEEE (2009), http://dl.acm.org/citation.cfm?id=1586263
20. Kjernsmo, K.: The necessity of hypermedia RDF and an approach to achieve it. In: Proceedings of the Linked APIs workshop at the 9th Extended Semantic Web Conference (May 2012), http://lapis2012.linkedservices.org/papers/1.pdf
21. Klyne, G., Carrol, J.J.: Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation (Feb 2004), http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/
22. Kopecký, J., Gomadam, K., Vitvar, T.: hRESTS: An HTML microformat for describing RESTful Web services. In: Proceedings of the International Conference on Web Intelligence and Intelligent Agent Technology. pp. 619–625. IEEE Computer Society (2008), http://dx.doi.org/10.1109/WIIAT.2008.379
23. Kopecký, J., Vitvar, T.: MicroWSMO. WSMO Working Draft (Feb 2008), http://www.wsmo.org/TR/d38/v0.1/
24. Kopecký, J., Vitvar, T., Bournez, C., Farrell, J.: Semantic Annotations for WSDL and XML Schema. IEEE Internet Computing 11, 60–67 (2007), http://cms-wg.sti2.org/doc/IEEEIC2007-KopeckyVBF.pdf
25. Lausen, H., Polleres, A., Roman, D.: Web Service Modeling Ontology (WSMO). W3C Member Submission (Jun 2005), http://www.w3.org/Submission/WSMO/
26. Maleshkova, M., Pedrinaci, C., Domingue, J.: Investigating Web APIs on the World Wide Web. In: Proceedings of the 8th European Conference on Web Services. pp. 107–114. IEEE (2010), http://sweet-dev.open.ac.uk/war/Papers/mmaWebAPISurvey.pdf
27. Maleshkova, M., Pedrinaci, C., Domingue, J.: Semantic annotation of Web APIs with SWEET (May 2010), http://oro.open.ac.uk/23095/
28. Maleshkova, M., Pedrinaci, C., Li, N., Kopecky, J., Domingue, J.: Lightweight semantics for automating the invocation of Web APIs. In: Proceedings of the 2011 IEEE International Conference on Service-Oriented Computing and Applications (Dec 2011), http://sweet.kmi.open.ac.uk/pub/SOCA.pdf
29. Maleshkova, M., Kopecký, J., Pedrinaci, C.: Adapting SAWSDL for semantic annotations of RESTful services. In: Proceedings of the On the Move to Meaningful Internet Systems Workshops, Lecture Notes in Computer Science, vol. 5872, pp. 917–926. Springer (2009), http://dx.doi.org/10.1007/978-3-642-05290-3_110
30. Manna, Z., Waldinger, R.: A deductive approach to program synthesis. Transactions on Programming Languages and Systems (TOPLAS) 2(1), 90–121 (1980), http://dl.acm.org/citation.cfm?id=357084.357090

31. Martin, D., Burstein, M., Hobbs, J., Lassila, O.: OWL-S: Semantic Markup for Web Services. W3C Member Submission (Nov 2004), http://www.w3.org/Submission/OWL-S/
32. Milanovic, N., Malek, M.: Current solutions for Web service composition. Internet Computing, IEEE 8(6), 51–59 (2004), http://ieeexplore.ieee.org/iel5/4236/29773/01355922.pdf
33. Norton, B., Krummenacher, R.: Consuming dynamic Linked Data. In: Proceedings of the 1st International Workshop on Consuming Linked Data (Nov 2010), http://ceur-ws.org/Vol-665/NortonEtAl_COLD2010.pdf
34. Page, K., Frazer, A., Nagel, B., Roure, D.D., Martinez, K.: Semantic access to sensor observations through Web APIs. In: 5th IEEE International Conference on Semantic Computing. IEEE (September 2011), http://eprints.soton.ac.uk/272695/
35. Page, K., De Roure, D., Martinez, K., Sadler, J., Kit, O.: Linked sensor data: RESTfully serving RDF and GML. In: Semantic Sensor Networks (Oct 2009), http://eprints.soton.ac.uk/271743/
36. Parsia, B., Sirin, E.: Pellet: An OWL DL reasoner. In: Proceedings of the Third International Semantic Web Conference (2004), http://iswc2004.semanticweb.org/posters/PID-ZWSCSLQK-1090286232.pdf
37. Pedrinaci, C., Domingue, J., Krummenacher, R.: Services and the Web of Data: An unexploited symbiosis. In: Proceedings of the AAAI Spring Symposium on Linked Data Meets Artificial Intelligence (2010), http://people.kmi.open.ac.uk/carlos/wp-content/uploads/downloads/2010/09/linkedServices-AAAI.pdf
38. Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF. W3C Recommendation (Jan 2008), http://www.w3.org/TR/rdf-sparql-query/
39. Speiser, S., Harth, A.: Integrating Linked Data and Services with Linked Data Services. In: The Semantic Web: Research and Applications, Lecture Notes in Computer Science, vol. 6643, pp. 170–184. Springer (2011), http://people.aifb.kit.edu/aha/2012/sms/lids-eswc2011.pdf
40. Stirbu, V.: Towards a RESTful plug and play experience in the Web of Things. In: IEEE international Conference on Semantic Computing. pp. 512–517. IEEE (2008), http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4597240
41. Verborgh, R., Steiner, T., Gabarró Vallés, J., Mannens, E., Van de Walle, R.: A social description revolution—describing Web APIs' social parameters with RESTdesc. In: Proceedings of the AAAI 2012 Spring Symposia (Mar 2012), http://www.aaai.org/ocs/index.php/SSS/SSS12/paper/download/4283/4665
42. Verborgh, R., Steiner, T., Van Deursen, D., Coppens, S., Gabarró Vallés, J., Van de Walle, R.: Functional descriptions as the bridge between hypermedia APIs and the Semantic Web. In: Proceedings of the Third International Workshop on RESTful Design. ACM (Apr 2012), http://www.ws-rest.org/2012/proc/a5-9-verborgh.pdf
43. Verborgh, R., Steiner, T., Van Deursen, D., De Roo, J., Van de Walle, R., Gabarró Vallés, J.: Capturing the functionality of Web services with functional descriptions. Multimedia Tools and Applications (2012), http://www.springerlink.com/index/d041t268487gx850.pdf
44. Verborgh, R., Steiner, T., Van de Walle, R., Gabarró Vallés, J.: The missing links–How the description format RESTdesc applies the Linked Data vision to connect hypermedia APIs. In: Proc. of the Linked APIs workshop at the 9th Extended Semantic Web Conference (May 2012), http://lapis2012.linkedservices.org/papers/3.pdf
45. Waldinger, R.: Web agents cooperating deductively. In: Formal Approaches to Agent-Based Systems, Lecture Notes in Computer Science, vol. 1871, pp. 250–262. Springer (2001), http://dx.doi.org/10.1007/3-540-45484-5_20

# SPARQL-Based Applications for RDF-Encoded Sensor Data

Mikko Rinne, Seppo Törmä, and Esko Nuutila

Department of Computer Science and Engineering,
Aalto University, School of Science, Finland
`firstname.lastname@aalto.fi`

**Abstract.** Complex event processing is currently more dominated by proprietary systems and vertical products than open technologies. In the future, however, internet-connected people and things moving between smart spaces in smart cities will create a huge volume of events in a multi-actor, multi-platform environment. End-user applications would benefit from the possibility for open access to all relevant sensors and data sources. The work on semantic sensor networks concerns such open technologies to discover and access sensors on the Web, to integrate heterogeneous sensor data, and to make it meaningful to applications. In this study we address the question of how a set of applications can efficiently access a shared set of sensors while avoiding redundant data acquisition that would lead to energy-efficiency problems. The Instans event processing platform, based on the Rete-algorithm, offers continuous execution of interconnected SPARQL queries and update rules. Rete enables sharing of sensor access and caching of intermediate results in a natural and high-performance manner. Our solution suggests that with incremental query evaluation, standard-based SPARQL and RDF can handle complex event processing tasks relevant to sensor networks, and reduce the redundant access from a set of applications to shared sensors.

**Keywords:** Complex event processing, SPARQL, RDF, Sensor systems

## 1 Introduction

Our future, with internet-of-things, is filled with sensors. Devices in our personal area network communicate with each other, with the things we own and with the services we are interested in. Connected devices form smart spaces, made to assist us in our daily tasks. Smart spaces interact with infrastructural sensor networks, forming smart cities. In the scale of cities the number of sensors can reach billions, and sensor observations be extremely heterogeneous due to differences in stimuli, vendors, software versions, operators, administrative domains etc.

At the same time there will be increasing numbers of applications that would need to access sensor observations. It would be wasteful for each application - or a closed group of applications - to deploy its own set of sensors. A lot of duplication could be avoided, and hardware and communication resources used more efficiently, if sensors were openly exposed on the Web. Applications could be given broader access to sensors without locking application-sensor pairs to vertical silos. If access from applications to shared sensors is enabled, some new problems will arise [14]: How to discover, access and search sensor data? How to integrate the sensor data coming from heterogeneous sources? How to make sensor data meaningful to applications?

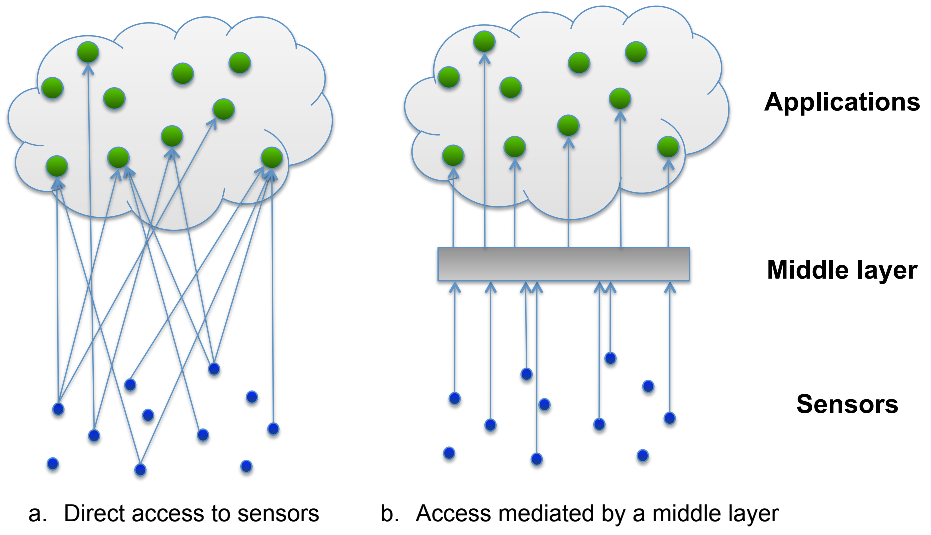

Applications
Middle layer
Sensors
a. Direct access to sensors
b. Access mediated by a middle layer

Fig. 1: Bridging between sensors and applications

A set of models to address these problems has been presented: Sensor Web Enablement (SWE) by OGC[1] [6], the model of Semantic Sensor Networks Incubator Group (SSN-XG) of W3C [14], the architecture of sensor Web applications in [7], and the Cloud, Edge, and Beneath model (CEB) by Xu et al [20]. They are all three-layer models where access from applications to sensors is mediated by a middle layer, as shown in Figure 1. In open sensor systems there are several needs for the middleware:

– *Abstraction*: The information from sensors needs to be layered to reasonable levels of abstraction, already for programmers but even more so for human end-users, who should only be notified or alerted with information significant to their personal contexts.
– *Interoperability*: These sensors, which can be mobile phones, thermometers, weather cameras or train positioning systems, are manufactured, owned and operated by various companies, public authorities and private persons. They

[1] Open Geospatial Consortium, http://www.opengeospatial.org/

will not operate under the same standard or service. There is a need for flexible representations for semantic relations of data from different origins.

- *Energy efficiency*: Many sensors will be depending on limited local power sources, and in the long run the applications, in total, can consume significant amounts of energy. Access to sensors should be managed in order to minimize unnecessary and wasteful work, in particular *redundant data acquisition* [20]. Redundancy can be minimized by sharing the work across applications to access the sensors, by caching intermediate results, and by suppression of irrelevant sensor input.

In this study we work on the basis of the *event* as an abstraction of a meaningful change in sensor readings. It is assumed that the primary operation of the system is based on sensors that *report events* in push-mode. This corresponds to the approach of Sensor Event Service (SES) [6] of the Sensor Web Enablement, although for the specification of complex events we rely on incremental evaluation of standard SPARQL queries and update rules [18] instead of the Event Pattern Markup Language (EML) [9] used in SES.

According to [15] a *complex event* is "an event that summarizes, represents, or denotes a set of other events". With this definition, "complex event processing" becomes defined by the *layering* of events, rather than the complexity of the underlying event processing problem, or the method used to solve it. This layering gives us the abstraction we need to hide the millions of events and come up with human-tangible conclusions like "the bus is late", "take this route to the office instead of your usual one" or "your house is probably on fire". So whereas 'simple event' is a production-oriented concept closer to sensor observations, 'complex event' is a consumption oriented concept: a specification of an event that an application or a user is interested in.

Interoperability is the central promise of semantic web technologies. They make it possible to establish relations both between ontologies and between instance data originating from different domains and sources. The expressive representation and query capabilities allow the flexible use of these technologies across domains. Event information can be enriched with linked data in the web, and the availability of inference tools enable the reasoning about event content.

If a set of applications were to access a shared set of sensors independently, there would be a lot of redundant acquisition, communication, and processing of sensor data. In this study we address the question of how a set of applications can efficiently access a shared set of sensors while avoiding unnecessary redundancy[2]. Our solution can be implemented using the Rete-algorithm that turns out to be a good fit for the task: it avoids the duplicate processing of events, it enables the sharing of sensor access and intermediate processing steps between applications, and it caches the intermediate results for efficient access later on. The big advantage of Rete is the high performance that manifests in short notification delays when the last piece of information that satisfies a query

[2] Some amount of controlled redundancy can be desirable for failure detection purposes [19]

is received. The natural place for Rete network is on the middle layer between sensors and applications but we discuss also its use on other layers.

The INSTANS event processing platform is based on continuous execution of interconnected SPARQL queries and update rules. We have presented how Rete-based incremental query evaluation enables efficient complex event processing with standard SPARQL and RDF [18]. This paper suggests that Rete is also suitable for reducing redundant access from applications to shared sensors.

The structure for the rest of this paper is the following: The principle of using collaborating SPARQL queries for event processing is introduced in Section 2. Our INSTANS platform is explained in Section 3. Section 4 explains the general approach of using Rete at different layers. Section 5 reviews some examples of potential application scenarios. Section 6 briefly reviews related work. Conclusions and future plans are presented in Section 7.

## 2 Event Processing Based on SPARQL

SPARQL is an expressive query language on RDF graphs. It can be used in a straightforward manner to filter events, construct new derived events, and specify complex patterns concerning the properties of multiple events. However, a single SPARQL query is not sufficient for many complex event processing applications [18]. SPARQL 1.1 Update[3] adds a critically important new feature: the capability to INSERT data into a triple store. When operated in an environment capable of continuous execution of multiple parallel SPARQL queries, the output of one query can be the input of other queries, as described in more detail in [18]. This way queries can store intermediate information for later use and pass information between each other, creating an entire event processing application out of a collaborative network of queries in standard SPARQL.

Compared to repeated execution of queries over time-based windows (as used in e.g. [3, 13]), continuous processing of SPARQL queries has clear benefits [18]:

1. *Instant availability of results.* For a memory-resident event processing application results are typically available in a few milliseconds. In most applications it would not be practical to re-run queries over event windows repeatedly with such rates.
2. *No duplicate detections due to overlapping windows.* With overlapping event windows same events can be processed more than once, resulting in duplicate notifications of exactly the same event instances.
3. *No missing detections on window borders.* If event windows do not overlap, event patterns crossing window borders will not be detected. To reduce the number of misses, the window length could be made longer but that would again increase the notification delays.
4. *No repeated processing over the same data.* The SPARQL queries and update rules can rely on the fact that each event is processed only once.

[3] http://www.w3.org/TR/sparql11-update/

The chaining and possibility to store intermediate results allows SPARQL queries to collaborate, forming an event processing application.

In window-based approaches to data stream processing such as [3, 13], the window lengths are typically based on either time duration or a number of triples. This approach is usually coupled with the assumption that each single RDF triple marks a standalone event. The input from sensors, however, could well contain any number of triples. There can be different sensor types providing partially overlapping information, different vendors and even different software versions of the same sensor. To be able to use all applicable sensors, we need to be able to support heterogeneous event formats. An example is illustrated in Figure 5, where the sensor sending location updates may or may not include altitude information. In open environments the existing event processing application should not be broken by additions of new event formats.

## 3 INSTANS Event Processing Platform

To address the requirement of near-real-time processing of complex, layered, heterogeneous events, we have created INSTANS[4] [1, 18]. Based on the Rete-algorithm [10, 11], INSTANS does continuous evaluation of incoming RDF[5] data against multiple SPARQL[6] queries. Intermediate results are stored into a $\beta$-node network. When all the conditions of a query are matched, the result is instantly available.

Again following the conventions of [15], an event processing network (EPN, Figure 2) consists of event processing agents (EPA, 1., 3., 4.) connected with event channels (2.). An output-only EPA is an event producer (1.), an input-only EPA is an event consumer (4.). In practice the same EPA can assume different roles in different contexts: The event consumer of one EPN can be the event producer of another one. In a sensor network context sensors are typical event producers and applications are typical event consumers. INSTANS can appear in any of the three EPA roles. Using RDF both for input and output means that different instances of INSTANS can be connected together. SPARQL queries can talk to each other both in micro-scale within one INSTANS-EPA and between different instances of INSTANS, offering very flexible possibilities to build a distributed system. For example:

– The computation of a single Rete can be distributed between parallel processors and / or processor cores
– When using multiple instances of Rete, the output of one EPA can be used to set the parameters of another EPA, e.g. the reporting trigger parameters of a sensor platform.

[4] Incremental eNgine for STANding Sparql, http://cse.aalto.fi/instans/
[5] http://www.w3.org/RDF/
[6] http://www.w3.org/standards/techs/sparql#w3c_all

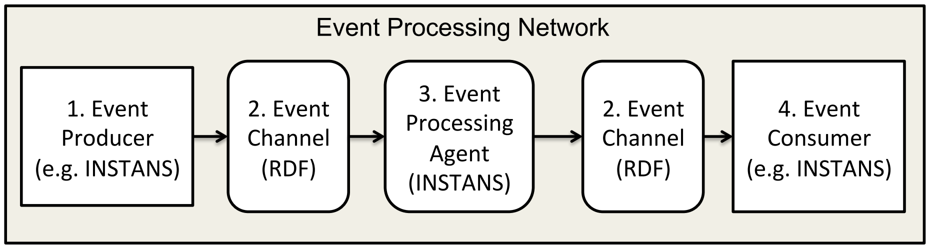

Event Processing Network
1. Event Producer (e.g. INSTANS)
2. Event Channel (RDF)
3. Event Processing Agent (INSTANS)
2. Event Channel (RDF)
4. Event Consumer (e.g. INSTANS)

Fig. 2: Event Processing Network (EPN) architecture showing INSTANS

The first version of INSTANS was coded on the Scala language[7]. Even though Scala has a lot of flexibility, it isn't built to support runtime generation of code. To exploit more dynamic programming techniques, we are working on a Rete implementation in Common Lisp. In Lisp, the Rete-net is compiled in setup-phase through macro expansion to executable Lisp code. The first results are highly encouraging: The close friends example, introduced in Section 5, produces the same results but runs 100-200 times faster than our original Scala implementation.

INSTANS processes each incoming triple immediately for every matching condition and saves intermediate results into its beta-node structure, as illustrated in Figure 3 using a query example from the application discussed in Section 5.2 involving the approach for timed events presented in Section 3.1. This example starts a timer to track the committed pickup time of a delivery task, when the task is assigned to a bidding driver.

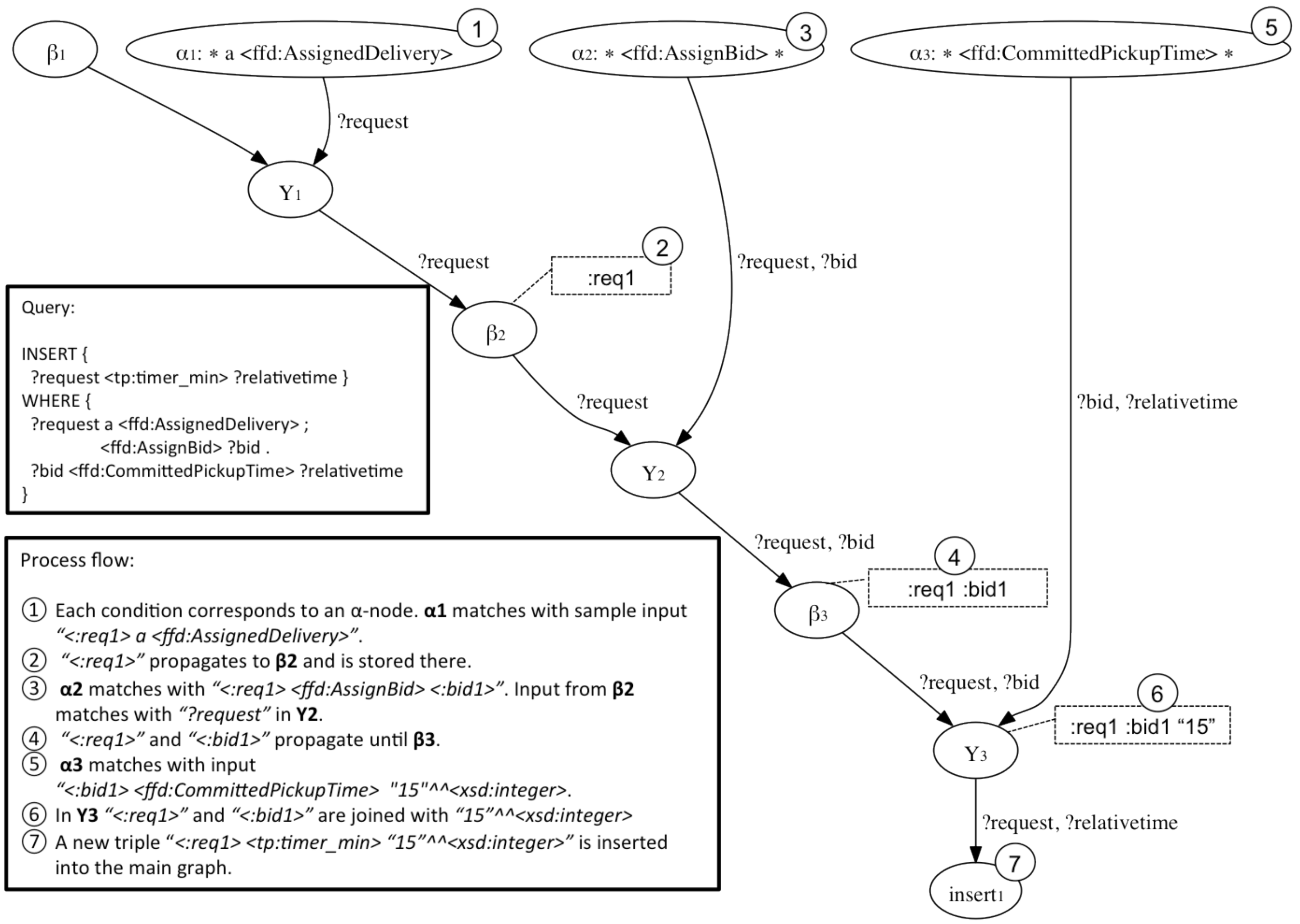
The main drawbacks of the Rete-algorithm are perceived to be memory consumption due to the saving of intermediate results within the structure and the slow processing of deletions [17, 16]. The memory consumption can be decreased by recognizing recurring patterns in queries and combining the corresponding nodes. Deletion can be made significantly faster by better indexing of the nodes.

## 3.1 Timed Events

The asynchronous nature of INSTANS means that all input is processed when it arrives. To support synthetic events at specific points in time like the detection of a missing event or the compilation of a report, the concept of timed events is needed. Timed events are built into INSTANS with the help of a special "timergraph" and a set of special predicates:

- **timer_sec / min / hour:** object (integer) specifies time-until-trigger in seconds / minutes / hours
- **timer_date:** triggers after the specified number of days at midnight
- **timer_month:** triggers after the specified number of months on the first day of the month at midnight

[7] http://www.scala-lang.org/



```
Query:

INSERT {
  ?request <tp:timer_min> ?relativetime }
WHERE {
  ?request a <ffd:AssignedDelivery> ;
              <ffd:AssignBid> ?bid .
  ?bid <ffd:CommittedPickupTime> ?relativetime
}
```

Process flow:

① Each condition corresponds to an α-node. **α1** matches with sample input *“<:req1> a <ffd:AssignedDelivery>”*.
② *“<:req1>”* propagates to **β2** and is stored there.
③ **α2** matches with *“<:req1> <ffd:AssignBid> <:bid1>”*. Input from **β2** matches with *“?request”* in **Y2**.
④ *“<:req1>”* and *“<:bid1>”* propagate until **β3**.
⑤ **α3** matches with input *“<:bid1> <ffd:CommittedPickupTime> "15"^^<xsd:integer>*.
⑥ In **Y3** *“<:req1>”* and *“<:bid1>”* are joined with *“15”^^<xsd:integer>*
⑦ A new triple *“<:req1> <tp:timer_min> “15”^^<xsd:integer>”* is inserted into the main graph.

Fig. 3: Example of SPARQL query processing in a Rete-net

– **timer_year:** triggers after the specified number of years on the first day of the year at midnight
– **timer_abs:** object (dateTime) specifies the absolute time for trigger

When inserted into the special timer queue, the objects are converted to absolute date-time values according to current system time and all predicates are set to <tp:waiting> status. Actors are used to schedule wakeup, at which time the predicate of triggered timers is changed to <tp:triggered>. Using the following query:

```
INSERT { GRAPH <http://example.org/timergraph> {
   ?event <tp:timer_sec> ?timevalue } }
WHERE { ?event <:seconds> ?timevalue }
```

A five-second timer would start, when receiving:

```
<:5sec_pulse> <:seconds> "5"^^<xsd:integer>
```

A SPARQL query tuned to monitor a certain timer triple can react to the change in predicate and carry out the necessary actions. One possibility is to set a new timer, creating a pulse generator:

```
WITH <http://example.org/timergraph>
INSERT { <:5sec_pulse> <tp:timer_sec> 5 }
WHERE { <:5sec_pulse> <tp:triggered> ?triggertime }
```

## 4 Use of Instans in Sensor Applications

Instans can be utilized on all three layers of Figure 1. Naturally, each of the applications on the uppermost layer could use Instans separately to process any kind of incoming events – simple or complex. This is especially pertinent to those processing needs that are specific to that particular application. Instans will still yield performance advantages when compared to complex event processing solutions based on repeated queries on stream windows.

At the middle layer significant efficiency improvements can be achieved in those event processing tasks – filtering, aggregation, enrichment, and pattern detection – that can be shared among multiple applications. When deployed at the middle layer, Instans would accept subscriptions from applications in the form of complex event patterns presented as SPARQL queries and update rules. The subscriptions of all applications are compiled into a common Rete network in which similar query structures are shared among different applications. As Instans receives events from the sensor layer, they are immediately processed though the Rete network triple by triple. Each triple is propagated as far as it can proceed through the network. It may be filtered away, or the propagation may stop in a join node in which no matching data from the other branch can be found. The data content of the triple remains in a beta memory waiting for potential future matches.

The efficiency benefits at the middle layer are a direct result of the well-known properties of the Rete algorithm. Each event is evaluated only once against similar query structure, the state of partially completed propagation is memorized, and the notifications of matches are produced immediately when a pattern becomes satisfied.

At the lowest layer, INSTANS can be used in sensor platforms that have enough processing power and memory to run the Rete network. The main role of INSTANS would be to construct meaningful events out of the mass of observations. The overall goal is to reduce the amount of communication – which is usually the most significant component in the energy consumption of sensor platforms – by some additional computation in event processing. As a result, a large volume of sensory observations may turn out to produce only few meaningful events that need to be communicated upwards.

A meaningful event is one that is *relevant* in the sense that it can match some query structures, and *significant* in the sense that it can affect the result of a query. In Rete only *changes in observed values* are significant. This can even be generalized: only deviations from expectations are significant. It means, for instance, that only those observed values that deviate from an expected trend should be reported. The upper layers are responsible to communicate to lower layers what kinds of events are relevant and significant. The reconfiguration can be done by executing SPARQL Update commands to lower layers with appropriate configuration data. The basic flow is illustrated in Figure 4.

First (1.) the application has a new reporting requirement such as an updated temperature threshold. This triggers a query in an instance of INSTANS (2.), emitting an RDF-format update request (3.):

```
<:outdoor_sensor1> <:min_reporting_temperature> "22"^^<xsd:integer>
```

Another instance of INSTANS in the sensor platform receives the configuration information and replaces the earlier local parameter with the new configuration (4.):

```
INSERT { GRAPH <http://sensorplatform.org/local_configuration> {
   <:outdoor_sensor1> ?parameter ?newvalue } }
WHERE { <:outdoor_sensor1> ?parameter ?newvalue }
```

After the new configuration is set, only temperature readings higher than 22 get reported (5.):

```
INSERT { GRAPH <http://sensorplatform.org/sensor_output> {
   <:outdoor_sensor1> <:temp> ?reading } }
WHERE {
   GRAPH <http://sensorplatform.org/sensor_input> {
      <:outdoor_sensor1> <:temp> ?reading }
   GRAPH <http://sensorplatform.org/local_configuration> {
      <:outdoor_sensor1> <:min_reporting_temperature> ?reading }
   FILTER (?reading > ?min_temperature)
}
```

Beyond this bare bone example, a more elaborate SPARQL-query would take into account other parameters and trigger the report either periodically using timed events, or only send the event once when the temperature rises above the threshold, depending on application requirements.

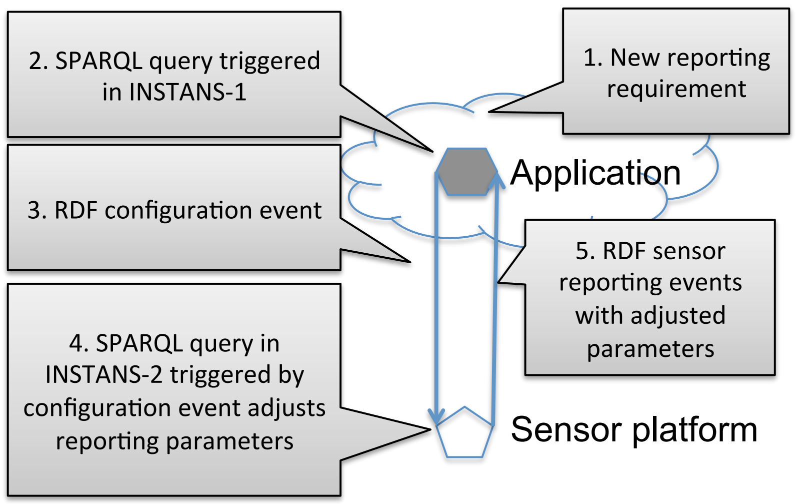

1. New reporting requirement
2. SPARQL query triggered in INSTANS-1
3. RDF configuration event
4. SPARQL query in INSTANS-2 triggered by configuration event adjusts reporting parameters
5. RDF sensor reporting events with adjusted parameters
Application
Sensor platform

Fig. 4: Flow of operation using Instans both in the layer of applications and as the reporting configuration platform at the middle layer.

In the wider context of a sensor network this approach could be used to manage the reporting from different sensors based on application requirements. Instead of each application going to an middle layer or all the way to the sensor to request (overlapping) measurements, service management in the application layer should compile all requests towards a sensor into a reporting scheme, which would satisfy the requirements from all applications with minimum access to the sensor.

## 5 Examples Scenarios

Below are two example scenarios used as test cases for Instans.

### 5.1 Close Friends [18]

Subscribers in the “Close Friends” scenario form a social network defined by foaf:knows properties. They report their locations with events like the one shown in Figure 5. When two friends are nearby, the system recognizes the situation and reports a “nearby” status. It is able to avoid repeated detections as long as the condition persists and build hysteresis into detecting, when the same persons are far enough to reset the “nearby” status.

Each subscriber would be running a Close Friends application in his or her mobile device. The sensor layer is formed by the location sensors of the same devices. The middle layer receives runs INSTANS: it receives location updates from mobile devices and sends nearby events to applications.

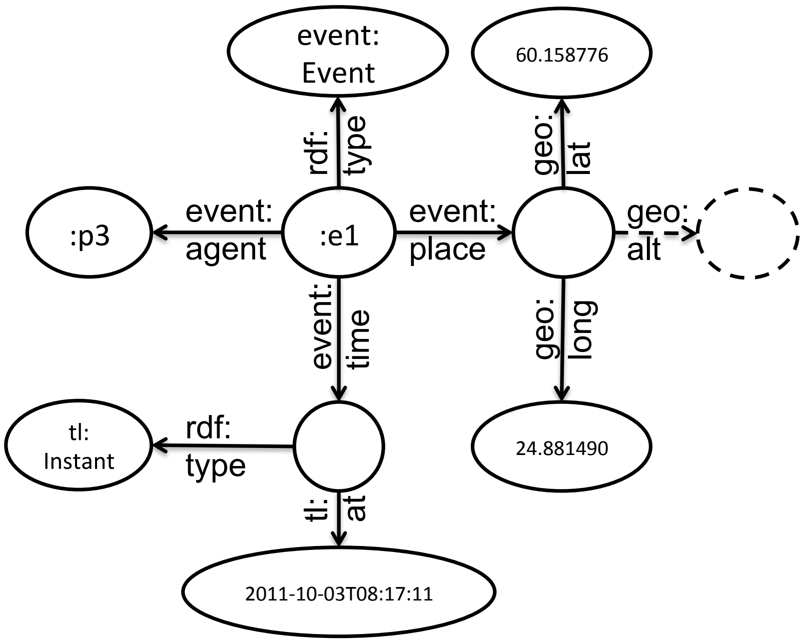
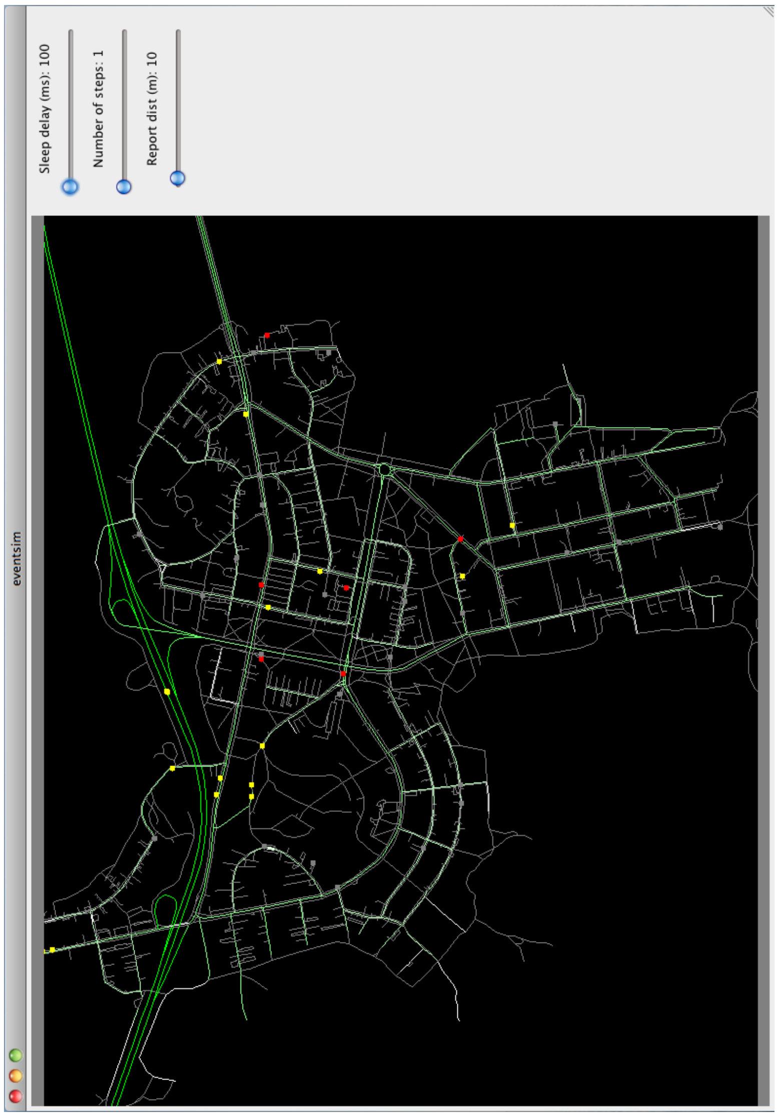


Fig. 5: Location Event Format

To generate input data, an event simulator has been created. The simulator can use map data from Open Street Map[8] and place a number of persons to move within the map area, generating location events. It is possible to output both to a file and to generate input data for INSTANS over a live TCP connection, slowing the simulator down to real-time operation. A screen capture of the event simulator is shown in Figure 6.

The detection of a "nearby" condition requires location update events from two separate persons. This setting works very well in the asynchronous environment of INSTANS: We only need to compare the two latest location events from two friends, checking that neither event is expired and we can get a match. In a system based on repeated execution of queries over time-based windows the requirements would contradict: To save power and network resources, location reporting frequency should preferably adapt to the movement of the person, but in a window-based system the achievable notification delay is limited by the

[8] http://www.openstreetmap.org/



Sleep delay (ms): 100

Number of steps: 1

Report dist (m): 10

Fig. 6: Screen capture of the event simulator showing 50 persons moving around the island of Lauttasaari in Helsinki

repetition rate of the window. If the window is very long compared to repetition rate, the same nearby-condition is detected multiple times. If the repetition rate is slow, detection delay increases. And if the window is short, the reporting frequency from terminals needs to be artificially increased to make sure that all users report within the window. In INSTANS none of these compromises are needed; reporting rate can be adjusted based on detected location changes and notifications are available only a few ms after reporting, much faster than would be feasible as a window repetition rate.

Support of heterogeneous event formats is also easier, when there are no windows defined by time or number of triples, which could break events consisting of multiple triples. Whether altitude is included in the example of Figure 5 or not, "close friends" operates the same way. As long as there is a way to link all the triples of an event together (e.g. a common subject), each application can use the triples they need and copying or deleting of the entire event can be done by matching the linking information. In "close friends" event identifier :e1 can be used to copy or delete all triples of the event without explicit knowledge of what parameters are included.

With the first-generation INSTANS programmed on Scala the notification delay from the creation of the second qualifying event to the reporting of the "nearby" detection was about 12 ms on a 2.26 GHz Core 2 Duo Mac running OS X 10.6.

### 5.2 Logistics Management

The "Fast Flowers Delivery" (FFD) event processing application, a typical logistics management task including reporting, is detailed in [8]. Flower stores request delivery service for flower orders, independent drivers bidding for the assignment based on availability, location and driver ranking. Demo implementations[9] on six different platforms are available at the book website, but none of them are coded in SPARQL.

Timed events are in heavy use in FFD. Each phase of the flower delivery needs to be monitored for time: Driver response to bid request, assignment of driver, pickup and delivery of flowers. Additionally the reports and driver ranking need to be processed at designated times.

When using a single heterogeneous event to describe a delivery task, taken through multiple phases, new IRIs need to be generated to keep the timers unique. SPARQL offers good tools for this: the subject IRI of a request can be bound together with a status string or another IRI to generate a new IRI, which can be used as a unique identifier.

## 6 Related Work

So far we have not been able to find another system in the research community, which would enable continuous processing of SPARQL 1.1 queries, including the

[9] http://www.ep-ts.com/content/view/79/

SPARQL Update methods of communication between queries. Sparkweave[10] [12] applies SPARQL queries to RDF data using an extended Rete-algorithm, but focusing on inference and fast data stream processing of individual triples instead of heterogeneous events. Sparkweave v. 1.1 also doesn't support SPARQL 1.1 features such as SPARQL Update.

Data stream processing focusing on fast filtering of individual triples, not events consisting of multiple triples, and time- or triple-based repetition of queries on windows have been the most popular approaches used by SPARQL-based stream processing systems such as C-SPARQL[11] [3–5] and CQELS[12] [13].

Jena[13] and Sesame[14], both popular platforms in the research community, have support for SPARQL Update, but not for multiple simultaneous queries.

The field of complex event processing is even more diverse. Popular tools include but are not limited to TIBCO BusinessEvents[15], StreamBase[16], Esper[17], Aleri[18], Apama[19] and ETALIS[20], which has a SPARQL pre-compiler called EP-SPARQL [2] supporting a subset of ETALIS features. The summary of CEP market at 2011[21] compiled by Paul Vincent of TIBCO includes 23 different systems. We haven't looked at all of them, but so far we have not come across any system based on the use of collaboration of standard-compliant SPARQL v. 1.1 in the way we have described.

## 7 Conclusions and Future Plans

Semantic web standards RDF and SPARQL are well suited for complex event processing due to their inherent strengths in managing multi-vendor multi-platform environments. In addition to simple conversions between vocabularies, inference capabilities and access to all linked open data they are also likely to be useful in developing semantic reasoning and enriching of event data.

Based on initial testing of the event processing tasks described in Section 5, the central paradigm of INSTANS – continuous incremental matching of multiple standard-based SPARQL queries supporting inter-query communication – seems to be managing well. When complemented with support for timed events we have

10 https://github.com/skomazec/Sparkweave
11 http://streamreasoning.org/download
12 http://code.google.com/p/cqels/
13 http://incubator.apache.org/jena/
14 http://www.openrdf.org/
15 http://www.tibco.com/
16 http://www.streambase.com/
17 http://esper.codehaus.org/
18 http://www.sybase.com/products/financialservicessolutions/complex-event-processing
19 http://www.progress.com/en/Product-Capabilities/complex-event-processing.html
20 http://code.google.com/p/etalis/
21 http://www.thetibcoblog.com/wp-content/uploads/2011/12/cep-market-dec2011.png

found no show stopper problems, which would render the approach unusable for any complex event processing task.

Compared to systems based on repeated execution of windows the approach used in INSTANS has multiple benefits. Notification delays in the order of milliseconds cannot be practically competed with by re-running queries at similar rates. While other systems re-run queries at fixed time intervals or after a fixed number of triples, INSTANS combines similar parts of the queries, processes each input triple as soon as it becomes available and memorizes intermediate results, resulting in significantly smaller amount of duplicate computation.

In the context of a sensor network, INSTANS could be deployed as a programmable and reconfigurable platform both close to the sensors and as a central agent. It is distributable on various levels. Configuration and operation are fully based on semantic web standards without any mandatory extensions, with SPARQL used as the programming language to create event-based applications and RDF used for communication. Due to the compatible approach of using RDF both for input and output, even large event processing tasks can be split into manageable sizes and layered abstractions.

After finalizing the transition to the Lisp-based platform, we are planning to attempt a more complete solution of the Fast Flowers Delivery application to verify that everything can be properly supported. To prepare for true big data usage, longer runs of the platform should be combined with testing of different kinds of garbage handling approaches. Support for querying external SPARQL endpoints as well as support for a selected set of inference rules are being planned.

## Acknowledgments

This work was partially supported by Tekes[22] through the funding to RYM-PRE[23] (Built Environment Process Re-Engineering) projects, and by European Commission through the SSRA (Smart Space Research and Applications) activity of EIT ICT Labs[24].

## References

1. Abdullah, H., Rinne, M., Törmä, S., Nuutila, E.: Efficient matching of SPARQL subscriptions using Rete. In: Proceedings of the 27th Symposium On Applied Computing. Riva del Garda, Italy (Mar 2012)
2. Anicic, D., Fodor, P., Rudolph, S., Stojanovic, N.: EP-SPARQL: a unified language for event processing and stream reasoning. In: Proceedings of the 20th international conference on World wide web (WWW'11). pp. 635–644. WWW '11, ACM, Hyderabad, India (2011)
3. Barbieri, D.F., Braga, D., Ceri, S., Grossniklaus, M.: An execution environment for C-SPARQL queries. In: Proceedings of the 13th International Conference on Extending Database Technology - EDBT '10. p. 441. Lausanne, Switzerland (2010)

[22] http://www.tekes.fi/en
[23] http://www.rym.fi/en
[24] http://eit.ictlabs.eu/ict-labs/thematic-action-lines/smart-spaces/

4. Barbieri, D.F., Braga, D., Ceri, S., Valle, E.D., Grossniklaus, M.: C-SPARQL: A Continuous query language for RDF data streams. International Journal of Semantic Computing 04, 3 (2010)
5. Barbieri, D.F., Braga, D., Ceri, S., Valle, E.D., Grossniklaus, M.: Querying RDF streams with C-SPARQL. ACM SIGMOD Record 39, 20 (Sep 2010)
6. Bröring, A., Echterhoff, J., Jirka, S., Simonis, I., Everding, T., Stasch, C., Liang, S., Lemmens, R.: New generation sensor web enablement. Sensors 11(3), 2652–2699 (2011)
7. Corcho, O., García-Castro, R.: Five challenges for the semantic sensor web. Semantic Web 1(1), 121–125 (2010)
8. Etzion, O., Niblett, P., Luckham, D.: Event Processing in Action. Manning Publications (Jul 2010)
9. Everding, T., Echterhoff, J.: Event Pattern Markup Language (EML). Discussion paper, Open Geospatial Consortium (2008)
10. Forgy, C.L.: Rete: A fast algorithm for the many pattern/many object pattern match problem. Artificial Intelligence 19(1), 17–37 (Sep 1982)
11. Forgy, C.L.: On the efficient implementation of production systems. Ph.D. thesis, Carnegie Mellon University, Pittsburgh, PA, USA (1979), aAI7919143
12. Komazec, S., Cerri, D.: Towards Efficient Schema-Enhanced Pattern Matching over RDF Data Streams. In: 10th ISWC. Springer, Bonn, Germany (2011)
13. Le-Phuoc, D., Dao-Tran, M., Parreira, J.X., Hauswirth, M.: A native and adaptive approach for unified processing of linked streams and linked data. In: ISWC'11. pp. 370–388. Springer-Verlag Berlin (Oct 2011)
14. Lefort, L., Henson, C., Taylor, K.: Semantic Sensor Network XG Final Report (2011), http://www.w3.org/2005/Incubator/ssn/XGR-ssn-20110628/
15. Luckham, D., Schulte, R.: Event processing glossary – version 2.0 (Jul 2011), http://www.complexevents.com/
16. Miranker, D.P.: TREAT: a new and efficient match algorithm for AI production systems. Ph.D. thesis, University of Texas at Austin, New York, NY, USA (1987)
17. Miranker, D.P.: TREAT: A better match algorithm for AI production systems. Tech. rep., University of Texas at Austin, Austin, TX, USA (1987)
18. Rinne, M., Abdullah, H., Törmä, S., Nuutila, E.: Processing Heterogeneous RDF Events with Standing SPARQL Update Rules. In: Meersman, R., Dillon, T. (eds.) OTM 2012 Conferences, Part II. pp. 793–802. Springer-Verlag (2012)
19. Silberstein, A., Puggioni, G., Gelfand, A., Munagala, K., Yang, J.: Suppression and failures in sensor networks: A bayesian approach. In: Proceedings of the 33rd international conference on Very large data bases. pp. 842–853. VLDB Endowment (2007)
20. Xu, Y., Helal, S., Thai, M.T., Schmalz, M.: Optimizing Push / Pull Envelopes for Energy-Efficient Cloud-Sensor Systems. In: MSWiM 2011 - 14th ACM international conference on Modeling, analysis and simulation of wireless and mobile systems. Miami (2011)

# A Framework for Acquiring Semantic Sensor Descriptions (Short Paper)

Luka Bradeško, Alexandra Moraru, Blaž Fortuna, Carolina Fortuna, Dunja Mladenić

Jožef Stefan Institute, Ljubljana Slovenia
{luka.bradesko, alexandra.moraru, blaz.fortuna, carolina.fortuna, dunja.mladenic}@ijs.si

**Abstract.** There has been great effort in developing ontologies for modeling sensor networks, describing various types of sensors and their context. However, when faced with a large scale deployment, the process of acquiring and managing semantic sensor metadata is challenging. This paper focuses on acquiring contextual metadata of sensors, such as location and surrounding environment, as opposed to technical metadata which can be derived from sensor's firmware. More specifically, the paper proposes a framework for collecting contextual metadata information with help of the mobile devices, which allows usage on the deployment site and as such lowers the cost.

**Keywords.** Semantic Sensor Web, Semantic Sensor Networks, Knowledge Acquisition, Mobile Applications

## 1 Introduction

With the rapid development and increasing number of real world applications of large scale sensor networks it became obvious that there is a need for software solutions supporting communication, sensor data retrieval and storage. In parallel to that, there is a need for an infrastructure to cover sensor descriptions, deployment and maintenance data. This paper focuses on the infrastructure supporting sensor metadata acquisition and management based on semantic technologies.

Semantic technologies have been identified as one of the key enabling technologies for sensor networks [1], contributing to understanding and managing of the sensors and measurements. One of the advantages of applying semantic technologies to sensor networks is the interoperability support, which in terms of comparison and data merging of different sensor networks, enables new solutions in solving problems.

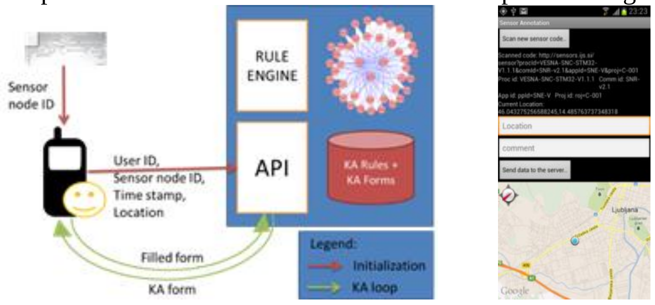
Existing systems that semantically annotate data require it to be inserted manually via xml configuration files or wiki. Attempts to use meta-data freely inserted by users haven't proved too successful [3]. More recent approaches use custom network protocols such as the Device Identification Protocol (DIP) to automatically obtain the technical meta-data [4], or they manually annotate a small number of sensors and then, based on the similarity of measurements they label other sensors [5]. The existing applications mostly focus on the measurements and insufficient attention has been paid to the sensor context information. The Open Geospatial Consortium leads efforts to define the Sensor Web Enablement standards i.e. SensorML and Observation-and-Measurement [1]. More recently, the need to more expressivity has been recognized, therefore the SWE standards have also been mapped into the Semantic Sensor Network (SSN) ontology by the W3C Semantic Sensor Network Incubator Group [2].

The contextual specifications depend on the concrete placement of the sensor, are more difficult to obtain and are not covered well within existing solutions. For example, an important piece of information is the geo positioning [6]. Fixed sensor platforms typically do not feature a GPS positioning module for it increases the size and cost of the platform and consumes power without providing much added value over time. Knowing the GPS coordinates enables the acquisition of the following (non-exhaustive list of) contextual information: geographical context (i.e. position on a map, plain or hills, proximity to river, etc.), details about the region such as population density, level of industrialization. All this contextual information can be automatically collected using Linked Data[1]. Besides the geographical information there is other meta-data which is best acquired at the time of the deployment: the actual time of installation, the configuration of the sensor box i.e. (Does it have external sensors, antenna? Is it waterproof?), the placement of a box, surroundings (Are there trees in the immediate vicinity, interferences on the same frequency?), etc.

## 2 Framework for Acquiring Semantic Sensor Descriptions

To start the KA (knowledge acquisition) process, the mobile terminal collects the sensor node ID directly from the sensor at its location site. Next, the mobile terminal sends an initialization message containing User ID, sensor node ID, time stamp and the GPS coordinates to the server. This is marked as the initialization step in Fig. 1.

The server validates the incoming data and adds it to the knowledge base. This starts an iterative process, in which the server uses predefined KA rules and the domain ontologies to generate KA forms and sends them to the mobile terminal. Next, the user fills forms via mobile terminal and submits the results back to the server. The server adds new data to the knowledge base and uses it to generate additional KA form. The KA loop then continues as described above and depicted in Fig. 1.



Scanned code: http://sensors.ijs.si/sensor?procid=VESNA-SNC-STM32-V1.1.1&comid=SNR-v2.1&appid=SNE-V&proj=C-001
Proc id: VESNA-SNC-STM32-V1.1.1 Comm id: SNR-v2.1
App id: ppid=SNE-V Proj id: roj=C-001
Current Location:
46.0432752566588245,14.485763737348318

**Fig. 1.** Architecture of the system (left) and mobile application (right)

[1] http://linkeddata.org/

The proposed framework is generic as its components can be implemented using a variety of existing tools and technologies. For instance, the acquisition of the sensor node ID can be done using barcodes, QR codes, RFID, NFC, etc. With respect to the vocabulary, the Semantic Sensor Network (SSN) ontology, geospatial ontologies such as Basic GeoWGS84 Vocabulary, GeoNames, time representation with W3C time ontology and observed properties using SWEET ontologies can be used. Further, application specific extensions can be added or generated when using the system.

## 3 Knowledge Acquisition

For the knowledge acquisition we took similar approach as is used in crowdsourcing the ontology learning process. We present to the user a variety of forms/questions, which are dynamically generated based on previous answers and based on the sensor instance. In order to define when and which forms to show, we introduced special vocabulary [7] in the ontology, to mark which knowledge to elicit. When the particular property is marked for KA, then the appropriate form gets generated.

We had to create a set of rules which trigger knowledge elicitation. For example if we want to get the bounding for property *<ssn:observes>* on all instances of class *<ssn:Sensor>*, we define the rule as follows:

```
[IF <?concept><rdf:type> <ssn:Sensor> THEN
<?concept> <ijs:generateForms> <ssn:observes>]                    (1)
```

This means that the system will ask for all the sensors what physical quantity is measured by it. These rules can be stacked to form a follow up conversation. When the first form is filled, the next one responds based on the previous answer, e.g.:

```
[IF <?concept>  <rdf : type> <ssn:Sensor> AND
<?concept> <ssn: observes> <_:observationProperty1> AND
<_:observationProperty1> <rdf: value> <”temperature”> THEN
<?concept> <ijs:generateForms> <ijs:hasTemperatureSensorType>]    (2)
```

This rule is triggered only when the assertion based on the previous answer states that the particular sensor measures temperature.

Theoretically it is possible that the user is presented with the RDF property and part of the ontology which he/she has to fill with the missing data. But this would be tedious work and requires the user of the system to be skilled in ontology engineering. To make the knowledge acquisition as easy and errorless as possible, the user is presented with the forms where he can enter knowledge without the RDF burden. These forms can be anything from HTML templates, to natural language questions and can be stored inside the ontology using special vocabulary or in an external database.

These forms are hooked up on the *<rdf:property>* resources and the knowledge about domain and the range is used to propose the predefined values which the user can enter. The example of the form which was issued by rule (1) can be seen in Fig. .

```
<form subject = “?$1” object =”?$2” predicate = ”ssn:observes”>
        “This sensor can measure ?$2”
</form>
```

**Fig. 2.** Example knowledge acquisition form.

Before presented to the user, the form is enriched with the constraints and possible answers, inferred from the *<rdf:range>* part of the ontology. In this example, the options would be “temperature”, “mass”, “current”, “voltage”, etc. If the user enters something new, the system informs him that he is creating a new concept, which is then added to the ontology, so everything stays consistent.

## 4 Implementation and Case Studies

As an example we look at an application around a plant care scenario, which shows the variety of data that can be acquired. There are numerous factors influencing the development of house plants, such as temperature, humidity, soil moisture, etc. Most of the information about the conditions required by house plants is available on the web; however, connecting it to the actual setting of the environment can be facilitated using a KA system.

Let us consider a scenario where three sensors are available for measuring the living conditions of our plant: a temperature sensor, a humidity sensor and soil moisture sensor. Further, we have defined a set of rules using concepts from SSN and SWEET ontologies and few extensions for sensor types and properties observed by sensors.

A simple example of a rule in this scenario would be “*If a sensor is attached to the flower pot, then ask what plant is in the pot?*” The rule can be represented with the <*sweet:hasOrganism*> predicate from SWEET ontology.

```
[IF <?concept> <rdf:type> <ssn:Sensor> AND
<?pot> <ssn:attachedSystem> <?concept> AND
<?pot> <rdf:type> <ijs:PlantPot> THEN
<?pot> <ijs:generateForms> <sweet:hasOrganism>]                    (3)
```

A more elaborate example is the acquisition of lighting conditions. Possible answers are fixed to the following list: direct, indirect sunlight, shadow, artificial light. Depending on the answer, further information can be inferred or asked. If the answer is one of the natural lights, the time interval can be calculated based on the geographical location and date. If the answer is not natural light, then a question about the exposure interval is issued. The above example translates to the following two KA rules:

```
[IF <?plant> <rdf:type> <sweet:Organism> THEN
<?plant> <ijs:generateForms> <sweet:assimilate>]

[IF <?plant> <sweet:assimilate> <?light> AND
<?light> <rdf:type> <ijs:ArtificialLight> THEN
<?light> <ijs:generateForms> <sweet:hasDuration>]
```

Other relevant meta-data can refer to the growing stage of the plant (germination, growth), last fertilization, size of the plant and pot, etc. The purpose for collecting all this metadata is to provide external applications the information needed to automatically detect when watering or fertilization is needed, etc.

The framework was also tested in an ongoing real world sensor deployment. In the CREW[2] project we installed 50 boxes with ISM and TV band energy detection sensors on public infrastructure in the town of Logatec. We manually prepared a list of technical configurations for each of the boxes. These configurations include: processing module ID, communication module ID, application module ID and project specific label. Next, QR codes with box-specific URIs were attached to their corresponding sensor boxes. In the field, the technician installing the sensor box read the QR code using his mobile terminal, and used the KA application (shown in Fig. 1) to geo tag the sensor and provide additional meta-data such as installation timestamp, infrastructure ID, infrastructure type (light pole, wall, and roof), particularities of the vicinity (tress, obstacles) and line of sight to other sensor boxes.

## 5 Conclusions

Comparison with the other solutions shows that our framework lowers the price of the meta-data acquisition. It allows users to collect data on the fly, using mobile devices and provides a controlled acquisition environment. The collected data is automatically validated and linked with LOD datasets. The proposed framework was compared feature wise with other similar systems, and the results are presented in Table 1. It was further validated in two case studies, demonstrating its usability for gathering technical data from the common users (plant caring) and in the real world deployment of sensors.

GSN[3] system uses XML files to manually label and describe each sensor with arbitrary data. The configuration process is easy for non-experts, but the disadvantage is that its terminology will differ across deployments, as it is hard to align different keywords used across deployments. Recently, work on using SSN, DOLCE and others as standardized vocabulary has been done, however, the focus is search rather than knowledge acquisition, therefore the process is not as automated and controlled as in the proposed framework.

Cosm[4] (the former Pachube) uses a predefined schema, therefore non-experts users can insert information such as title of the sensor setup, feed, id, location, etc. There's no standard vocabulary and no extension by the user seems possible. The scope of the platform is federating data from sensor deployment and eventually monetizing on it, rather than building an interoperable data management system.

In the SPIRFIRE project [5], work on automatically labeling streams of measurements is ongoing. Standard vocabulary is used, but the approach is not geared towards meta-data acquisition.

[2] http://www.crew-project.eu/

[3] http://sourceforge.net/apps/trac/gsn/

[4] https://cosm.com/docs/v2/

Linked Stream Middleware-LSM [8] is a platform where users register their sensors and annotate the data stream. It uses standard ontologies, with the possibility of importing new ones. The drawback of this system is that it can be only used by expert ontologists and the annotation has to be done after the sensor is already deployed.

As part of future work, we plan to use the system in a large sensor deployment scenario, as described in Section 4. We also plan to further abstract the system by making the definition of KA rules and forms simpler and more automated.

**Table 1.** Feature wise comparison of existing sensor meta-data systems

| Solution | Schema | Vocabulary | Ontology | Non-expert use |
|---|---|---|---|---|
| GSN | No schema | Custom | Fixed | Yes |
| COSM | Predefined | Custom | Fixed | Yes |
| SPITFIRE | Flexible | Standard | Fixed | No |
| LSM | Flexible | Standard | Extendible | No |
| Our KA system | Flexible | Standard | Extendible | Yes |

**Acknowledgments.** This work was partially supported by the Slovenian Research Agency and the ICT Programme of the EC under, ENVISION (ICT-2009-249120) and PlanetData (ICT-NoE-257641). Thanks to our colleagues in SensorLab.

## References

1. A. Sheth, C. Henson, S. Sahoo: Semantic Sensor Web. IEEE Internet Computing, 2009.
2. M. Compton, et al.: The SSN Ontology of the W3C Semantic Sensor Network Incubator Group. Journal of Web Semantics, Elsevier, 2012.
3. A. Moraru, C. Fortuna, D. Mladenić: Using Semantic Annotation for Knowledge Extraction from Geographically Distributed and Heterogeneous Sensor Data. SensorKDD (colocated with KDD), July 2010.
4. A. Moraru, M. Vucnik, M. Porcius, C. Fortuna, M. Mohorcic, D. Mladenic: Exposing Real World Information for the Web of Things. IIWeb (co-located with WWW), March 2011
5. M. Hauswirth, I. Chatzigiannakis: Provisioning Semantic IoT Services, Project: SPITFIRE.FIRE Hands-on FIA, Aalborg, Denmark.
6. C. Fortuna, B. Fortuna, K. Kenda, M. Vucnik, A. Moraru, D. Mladenic: Towards Building a Global Oracle: a Physical Mashup Using Artificial Intelligence Technology. Third International Workshop on the Web of Things, June 2012.
7. M. Witbrock, C. Matuszek, A. Brusseau, R.C Kahlert, C.B. Fraser, D.B. Lenat: Knowledge Begets Knowledge: Steps towards Assisted Knowledge Acquisition in Cyc. AAAI Spring Symposium on Knowledge Collection from Volunteer Contributors (KCVC), pp. 99-105. Stanford, California, March 2005.
8. D. Le Phuoc, H. N. Mau Quoc, J. X. Parreira, M. Hauswirth: The Linked Sensor Middleware - Connecting the real world and the Semantic Web. In Semantic Web Challenge 2011, ISWC 2011, http://challenge.semanticweb.org

# Short Paper: An Ontology Design Pattern for Spatial Data Quality Characterization in the Semantic Sensor Web

Auriol Degbelo

Institute for Geoinformatics, University of Muenster, Muenster, Germany
`degbelo@uni-muenster.de`

**Abstract.** Quality is an important aspect of data discovery in the Semantic Sensor Web. This work extends current endeavors to make the Sensor Web more semantic by introducing an ontology design pattern which facilitates the modeling of aspects of spatial data quality. The implementation of a software program over two scenarios demonstrates the usefulness of the ontology design pattern for the Semantic Sensor Web.

## 1 Introduction

The Semantic Sensor Web (SSW) is defined after [12] as a framework for providing enhanced meaning for sensor observations and enabling the awareness of the situations that sensors observe. The SSW complements the efforts of the Sensor Web Enablement (SWE) Initiative of the Open Geospatial Consortium (OGC) [1]. The main technologies of the SSW are OGC standards (e.g. the Sensor Observation Service), ontologies (e.g. the Semantic Sensor Network Ontology) and rules (e.g. the Semantic Web Rule Language).

A review of the recent developments of the Sensor Web Enablement framework of the OGC was done in [3], and it identified the following open challenge:

> "Knowledge about the quality ... of sensor outputs is essential for making the right decisions based upon observations. At the moment, such information is often missing in observations and there is no unique way of how to incorporate it".

This work suggests an ontology design pattern [2] (ODP) to describe quality aspects of sensor observations [3]. It builds upon ontologies for sensors and observations presented in [9,4] and aims at complementing them. Section 2 presents a brief introduction to spatial data quality components. Section 3 introduces the ODP to describe quality aspects, section 4 discusses the validation of the ODP and section 5 concludes the paper.

[1] See an introduction to the OGC SWE in [2].

[2] An ontology design pattern is defined after Gangemi and Presutti [6] as a modeling solution to solve a recurrent design problem. Ontology design pattern as used here refers to *content ontology design pattern* (see [6] for details).

[3] Throughout the paper the terms 'sensor observation', 'observation', 'data source', 'data' and 'dataset' are used interchangeably.

## 2 Quality and components of spatial data quality

Two major aspects of quality characterization in the SSW can be distinguished: quality aspects that have to do with the *characteristics of the data sources*, and those that have to do with *the creation of applications based on these data sources*. These two categories of quality aspects were already summarily mentioned in [5]. The current work discusses only quality aspects of data sources. In particular, it focuses on *spatial* data quality, in view of the fact that the SSW reposes on standards developed by the Open *Geospatial* Consortium. The following definition (reflecting a data consumer perspective) to the term is suggested. It is adapted from [13].

> *Quality is the degree to which a data or service fulfills the needs of a consumer. It is a function of intangible properties (of the data) considered pertinent to the satisfaction of the consumer's needs.*

The intangible properties considered pertinent to the satisfaction of the consumer's needs are also called *components of data quality*. As regards spatial data, quality components vary from author to author. For instance, ISO 19113 includes completeness, logical consistency, positional accuracy, temporal accuracy and attribute accuracy (see [10]); Paradis and Beard [11] includes accuracy, resolution, consistency and lineage. This section will not review all the components of spatial data quality. Instead, for the purposes of the illustration (see section 4), the quality component 'resolution' is chosen. *Resolution* is defined here as the amount of detail in the dataset. Spatial resolution of raster data can be measured using the size of the raster cells; for spatial resolution of vector data, several measures are possible (see [7] for the discussion).

## 3 ODP for spatial data quality characterization

Gangemi and Presutti [6] suggest four ways of creating content ontology design patterns (CPs), namely: (i) reengineering from other data models, (ii) specialization/composition of other CPs, (iii) extraction from reference ontologies, and (iv) creation by combining extraction, specialization, generalization, and expansion.

The pattern presented in this section is obtained by *extraction from the Stimulus Sensor Observation (SSO) ontology design pattern* [4]. Two classes are left out: the stimulus and the sensor. They are not included because for a quality assessment operation, there is no need to describe the stimulus, nor is there a need to describe a sensor that performs a measurement. Instead, it suffices that the data consumer describes the *procedure used for quality assessment*.

The documentation of the pattern uses the fields suggested in [6]. A small difference though, is that the field **Diagram** points to a conceptual map depicting the ODP aligned to the foundational ontology Dolce Ultra Light (DUL), instead of a UML class diagram as initially suggested in [6]. The pattern is encoded in the Web Ontology Language (OWL) using Protégé [5].

[4] or the Semantic Sensor Network (SSN) Ontology.

[5] http://protege.stanford.edu/.

**Name:** ontology design pattern for spatial data quality characterization
**Intent:** to describe spatial data quality components of sensor observations
**Competency questions:** (i) what is the value of the spatial data quality component for this sensor observation? (ii) what is the quality criterion used to assess the quality of the sensor observation?
**Elements:** The pattern has 5 elements: Data, DataQualityCriterion, DataQualityComponent, DataQualityObservation and DataQualityResult. A *data* is the output of an observation process involving a sensor, a stimulus, a sensed property and a feature. It is equivalent to 'Observation' as defined in the SSN ontology. A *DataQualityComponent* is any property of the data that the consumer would like to approximate. Examples of spatial data quality components were mentioned in section 2. A *DataQualityCriterion* is a criterion defined by the data consumer to get information about the quality of the data. A *DataQualityObservation* is an operation by which a data quality value is assigned to a data quality component using a data quality criterion. The outcome of a data quality observation is a *DataQualityResult*.
**Diagram:** see Figure 1
**Consequences:** *Benefits:* (i) Reasoning and inference of spatial data quality component values for existing sensor observations (ii) Detection of inconsistencies during the integration of observations with different quality levels (iii) Detection of inconsistencies during the integration of observations for which different quality criterion have been used to assess the spatial data quality. *Trade-offs:* The pattern does not give a quality value like 'high quality' or 'low quality' as an end result. Instead, it helps to infer the value of a spatial data component (e.g. resolution = 20m) and it is left up to the data consumer to decide whether 'resolution = 20m' means high or low quality for the task at hand.
**Known uses:** see examples of uses in section 4
**Extracted from:** the SSO ontology design pattern / the SSN ontology
**Building block:** `http://wsmls.googlecode.com/svn/trunk/global/Patterns/Quality/DataQuality/dataqualitymodule.owl`

## 4 Validation of the ODP

A software program was developed to test the usefulness of the ODP. This software program serves as a proof that (i) the pattern can be used to perform inference of quality component values and (ii) the pattern can be used to warn against the integration of datasets for which different quality criteria are used for quality assessment. This method for validation falls into the category 'Empirical validation' introduced in [8]. Regarding the technologies, Java was used as programming language, inference rules were written using the Semantic Web Rule Language (SWRL), Pellet was used as reasoner, the ODP was accessed using the OWL API, and Jena was used to perform SPARQL queries over the ontology. Ontology instances were added to the ODP in order to answer the

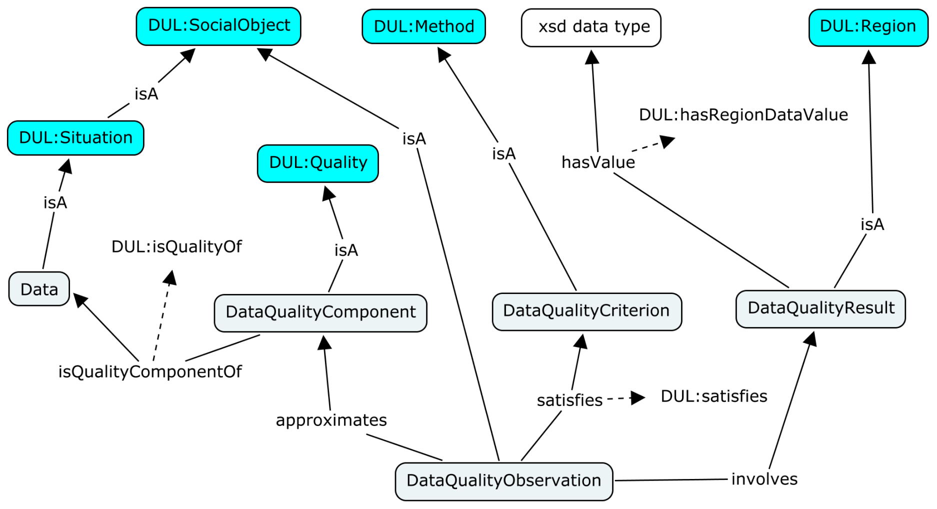


**Fig. 1.** Ontology Design Pattern for spatial data quality characterization aligned to Dolce Ultra Light (the prefix DUL indicates concepts from Dolce Ultra Light). The alignment of the pattern to Dolce Ultra Light is inherited from the alignment of the SSO to Dolce Ultra Light.

two questions posed in ScenarioA and ScenarioB. The Java code is available at `http://ifgi.uni-muenster.de/~degbelo/Resources/SSN2012Degbelo` or upon request to the author.

**ScenarioA** - *Inference of data quality value.* A decision-maker has at his/her disposal different OGC services delivering data about an observed property. He/she would like to answer the question - what is the value of the spatial resolution for an observation offering?[6] - in order to compare them and select the most appropriate for his/her task. A quality criterion for the observation offering is the sampling density[7] (i.e. number of sensors per square meters).
**Comment:** Information about the spatial resolution of an observation offering can be deduced from the GetCapabilities file of an OGC service. Using the ODP introduced earlier for this scenario, it is only required to parse the GetCapabilities file and assert (or store) the spatial extent of the observation offering as well as its number of sensors. The spatial resolution can then be inferred using a SWRL rule (see details in the Java code). Inference in turn is useful to address one of the drawbacks of adding semantic annotations to sensor nodes in sensor networks. In effect, [1] pointed out that adding semantics to sensor nodes in a

[6] The question is an application of the competency question (i) from section 3 to the spatial data quality component ‘Resolution’.
[7] This is only one way of assessing the resolution of an observation offering. Resolution as defined in the SSN ontology can be inferred from the characteristics of the sensor.

sensor networks implies more data to be exchanged, which in turn leads to an increase of sensor nodes' power consumption. Therefore, the less the amount of semantic data to store, the better.

**ScenarioB** - *Detection of inconsistencies.* This scenario is adapted from the Oil Spill scenario[8] of the European project ENVISION[9]. The project aims at developing an infrastructure for environmental web services with ontologies.

Scenario: accidental oil releases to the sea may have severe consequences on both natural resources and human enterprises. For oil spill decision making, it is essential to be able to predict the fate and effects of the spilled oil. Fate prediction requires data on the spill (location, time, amount, oil type), the environmental conditions (wind, current), and geography (sea depths, coast line). A decision-maker has different datasets for oil spill prediction at his/her disposal and would like to combine them.
**Comment:** For the purposes of the illustration, it is assumed that - within this scenario - the different types of data for oil spill prediction are available in vector format. It is also assumed that the decision-maker has done a preliminary look-over where all the datasets available were found to have a similar spatial resolution (say 100meters).

Given that there are various ways of defining the spatial resolution for vector data, an additional question to answer is: what criterion is used to assess the spatial resolution of the observation offering? [10] in order to ensure that heterogeneous datasets which have the same data quality value are effectively compatible with respect to their resolution. This check is possible with the ODP proposed through a simple SPARQL query (see details in the Java code).

## 5 Conclusion

Knowledge about the quality of sensor observations is an important aspect of the discovery of resources in the Semantic Sensor Web. This work has suggested an ontology design pattern (ODP) to characterize the quality of sensor observations. The ODP is relevant for the annotation of sensor observations with spatial data quality components. It can be used to infer spatial data quality component values for existing sensor observations and warn against the integration of sensor observations assessed with different quality criteria. The ODP was aligned to the foundational ontology Dolce Ultra Light and validated through the development of a software program.

[8] See a detailed presentation of the oil spill scenario at `http://envision.brgm-rec.fr/OS-UseCase.aspx` (last accessed: August 30, 2012).
[9] See a presentation of the project at `http://www.envision-project.eu`.
[10] This question is an application of the competency question (ii) from section 3 to the spatial data quality component 'Resolution'.

## Acknowledgments

The work is funded by the German Academic Exchange Service (DAAD A/10 /98506), and the European Commission through the ENVISION Project (FP7-249170). It is conducted within the International Research Training Group on Semantic Integration of Geospatial Information (DFG GRK 1498). The ideas presented were shaped through discussions with Patrick Maué, Alejandro Llaves, Werner Kuhn, Alkyoni Baglatzi, Henry Michels and Marcell Roth.

## References

1. Barnaghi, P., Meissner, S., Presser, M., Moessner, K.: Sense and sens'ability: Semantic data modelling for sensor networks. In: Cunningham, P., Cunningham, M. (eds.) Proceedings of the ICT-MobileSummit 2009. Santander, Spain (2009)
2. Botts, M., Robin, A.: Bringing the sensor web together. Geosciences 6, 46–53 (2007)
3. Bröring, A., Echterhoff, J., Jirka, S., Simonis, I., Everding, T., Stasch, C., Liang, S., Lemmens, R.: New generation sensor web enablement. Sensors 11(3), 2652–2699 (2011)
4. Compton, M., Barnaghi, P., Bermudez, L., García-Castro, R., Corcho, O., Cox, S., Graybeal, J., Hauswirth, M., Henson, C., Herzog, A., Huang, V., Janowicz, K., Kelsey, W.D., Le Phuoc, D., Lefort, L., Leggieri, M., Neuhaus, H., Nikolov, A., Page, K., Passant, A., Sheth, A., Taylor, K.: The SSN ontology of the W3C semantic sensor network incubator group. Web Semantics: Science, Services and Agents on the World Wide Web (2012)
5. Corcho, O., García-Castro, R.: Five challenges for the semantic sensor web. Semantic Web 1(1), 121–125 (2010)
6. Gangemi, A., Presutti, V.: Ontology design patterns. In: Staab, S., Studer, R. (eds.) Handbook on ontologies, pp. 221–243. Springer Berlin Heidelberg (2009)
7. Goodchild, M.F.: Scale in GIS: an overview. Geomorphology 130(1-2), 5–9 (2011)
8. Hammar, K., Sandkuhl, K.: The state of ontology pattern research: A systematic review of ISWC, ESWC and ASWC 2005-2009. In: Blomqvist, E., Chaudhri, V., Corcho, O., Presutti, V., Sandkuhl, K. (eds.) Proceedings of the 2nd International Workshop on Ontology Patterns - WOP2010. CEUR-WS.org, Shanghai, China (2010)
9. Janowicz, K., Compton, M.: The Stimulus-Sensor-Observation ontology design pattern and its integration into the semantic sensor network ontology. In: Taylor, K., Ayyagari, A., De Roure, D. (eds.) The 3rd International workshop on Semantic Sensor Networks. CEUR-WS.org, Shanghai, China (2010)
10. Kumi-Boateng, B., Yakubu, I.: Assessing the quality of spatial data. European Journal of Scientific Research 43(4), 507–515 (2010)
11. Paradis, J.R., Beard, K.: Visualization of spatial data quality for the decision maker: a data quality filter. URISA Journal 6(2), 25–34 (1994)
12. Sheth, A., Henson, C., Sahoo, S.S.: Semantic sensor web. IEEE Internet Computing 12(4), 78–83 (2008)
13. Veregin, H.: Data quality parameters. In: Longley, P.A., Maguire, D.J., Goodchild, M.F., Rhind, D.W. (eds.) Geographical information systems: principles and technical issues, chap. 12, pp. 177–189. John Wiley and Sons, New York, 2nd edn. (1999)

# Short Paper: From Streaming Data to Linked Data – A Case Study with Bike Sharing Systems

Edna Ruckhaus[1,2], Jean-Paul Calbimonte[2], Raúl García-Castro[2], and Oscar Corcho[2]

[1] Universidad Simón Bolívar, Venezuela
ruckhaus@ldc.usb.ve
[2] Ontology Engineering Group, Universidad Politécnica de Madrid, Spain
{jpcalbimonte,rgarcia,ocorcho}@fi.upm.es

**Abstract.** Current methods and tools that support Linked Data publication have mainly focused so far on static data, without considering the growing amount of streaming data available on the Web. In this paper we describe a case study that involves the publication of static and streaming Linked Data for bike sharing systems and related entities. We describe some of the challenges that we have faced, the solutions that we have explored, the lessons that we have learned, and the opportunities that lie in the future for exploiting Linked Stream Data.

## 1 Introduction

The process of publication of a dataset as Linked Data is composed of several activities: specification, modeling, generation, publication, and exploitation [1]. Current methods and tools that support Linked Data publication have been designed for static data without considering the growing amount of streaming data available on the Web. Nevertheless, streaming data is being used in a large number of domains (financial, environment, transport, energy, among others), and may be generated by physical sensors, by software systems or even by humans.

The representation of streaming data following the principles of Linked Data facilitates its integration to the diverse datasets in the Linked Data cloud, and also to other private Linked Data datasets. Streaming data usually coexists with static data, either because there is static data associated with the sensors that produce data streams, usually in the form of descriptive information on the sensor platform and observations (e.g., platform name, location, observed property), or because there are links to static data published in another dataset. Additionally, user requirements may be related to statistics on streaming data that have been accumulated over time, so historical streaming data needs to be considered in the different steps of linked stream data publication.

In this paper we report on our experience on exposing as Linked Data some static and dynamic data available in the Spanish cities of León and Zaragoza. More specifically, we have focused on static data about points of interest, available from Government open data portals, and on dynamic data about bicycle sharing systems.

## 2 Case Study: Bicycle Sharing Systems

Several bicycle sharing systems in cities all over the world have made their data available on the Web. Bike rental stations are distributed in different points in the city and the bike sharing systemusually allows users to pick up a bike at any station and drop it off at any (other) station. The goal of this case study is to publish and exploit up-to-date Linked Data about the availability of bikes and free slots in the stations, and links to related resources like travel guides and points of interest (e.g., museums, restaurants).

At this first stage, we are using the data rendered by the services provided by the CityBikes API[1], focusing on the bike sharing systems from the Spanish cities of León and Zaragoza. Besides, we have connected this streaming data to static data from open data portals from these cities, on museums and libraries (for León), and on restaurants (for Zaragoza), and to data about travel guides from El Viajero [2].

## 3 Data Publication Activities

In this case study, we followed a process inspired by the method proposed in [1], which envisions a continuous process that consists of five main activities: specification, modelling, generation, publication and exploitation.

### 3.1 Specification

The data sources selected for this case study were related to resources that could be useful to locals and visitors that use bike sharing systems in the different cities.

The data for bike sharing systems were obtained in JSON through the CityBikes API. Data for restaurants, museums, libraries and guides is published in RDF by open data portals from the cities of León and Zaragoza[2]. While data from Zaragoza was generally of high quality, we found some problems with data from León, namely: invalid URIs, points of interest that were either not geo-located or with a geo-position but no location name, different coordinate systems for geographic positions, and timestamps that did not represent the timezone where they were located. After some interactions, these problems were corrected by the data providers themselves; this shows that data reuse can help in the curation of data sources. In the case of the different coordinate systems, the inconsistency was solved during the RDF generation step.

### 3.2 Modelling

We have built the citybikes ontology network[3] to represent knowledge related to available bikes and free slots in bike sharing systems. These measurements represent the state of a bike station in a particular place and time and are measured through a sensor in each station. The citybikes ontology network follows a modular structure consisting

[1] `http://api.citybik.es/`
[2] `http://www.datosabiertos.jcyl.es/`, `http://www.zaragoza.es/ciudad/risp/`
[3] `http://transporte.linkeddata.es/files/citybikesontologynetwork.zip`

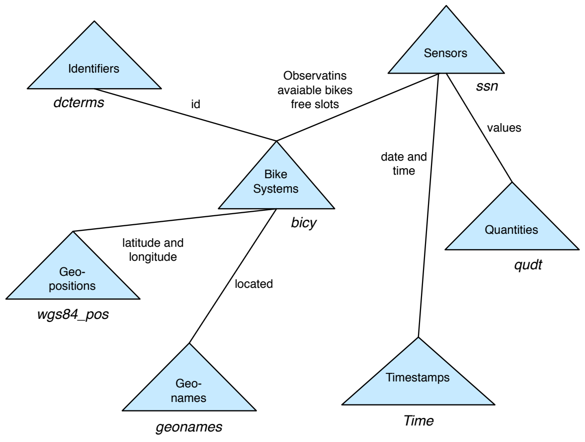


**Fig. 1.** Bike Sharing Ontologies Network

of a central ontology that is related to a set of ontologies that describe the different sub-domains involved in the modelling of the bike station measurements (Figure 1).

The central ontology is the bike sharing system ontology, which contains concepts such as the bike system and its name, the station and its name, number, internal id, status, description, number of boxes, free slots and free bikes. We reuse the following ontologies: Semantic Sensor Network[4] for sensors and observations, Geonames[5] to define the location of the systems and the stations based on their latitude and longitude, W3C time ontology[6] to represent the timestamp as an instant, Dublin Core[7] for identifiers, WGS_84[8] for geo positioning bike sharing systems and stations with latitude and longitude, and QUDT[9] for the number of available bikes and free slots.

Once the ontology was developed, we defined a resource naming strategy to ensure that every class in the ontology can have individuals with unique identifiers (i.e., URIs). For this, it was necessary to identify the cardinalities of the properties in the ontology, since information on the "conceptual schema" of the data sources was not explicit.

## 3.3 RDF Data Generation and Publication

In our use case and in sensor applications in general, we require publishing and consuming stored and streaming data. The first type, also referred to as static, is often related to the metadata and contextual information about sensor data, including geographical location, sensor and station characteristics, observed features, and also includes data

[4] `http://purl.oclc.org/NET/ssnx/ssn#`
[5] `http://www.geonames.org/ontology#`
[6] `http://www.w3.org/2006/time#`
[7] `http://purl.org/dc/terms/`
[8] `http://www.w3.org/2003/01/geo/wgs84_pos`
[9] `http://qudt.org/`

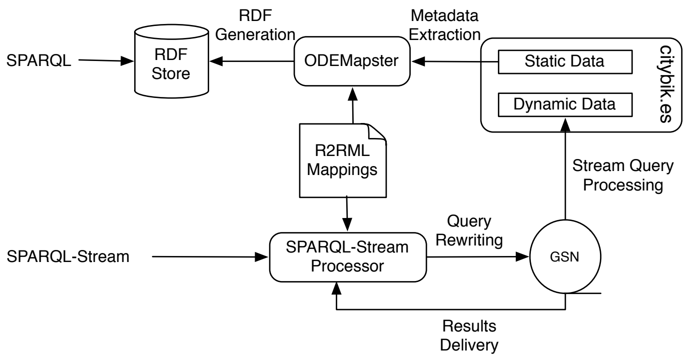


**Fig. 2.** Approach to generate static RDF and to query RDF streams with SPARQL$_{\text{Stream}}$

about points of interest (museums, libraries, etc.). Streaming data, in contrast, are highly dynamic and are centered on the observations of free slots and available bikes.

For static data, there are methods and tools for generating RDF and publishing them as Linked Data. Some of these tools generate the RDF data using declarative mappings such as R2RML[10] from relational databases, in bulk load operations. This approach is effective for static data, as it is seldom updated. However, for the dynamic sensor observations and for performing continuous queries, this is not the case. The available tools are unsuitable for materializing data in real time, and even if they were, the SPARQL query language does not consider streaming data operators such as time windows.

Therefore, we used a different approach (see Figure 2), based on the idea of reusing streaming sensor data processing engines, that are able to process highly dynamic data efficiently, and using temporal constructs. In order to use such engines, we rely on ontology-based query rewriting of SPARQL queries with streaming extensions. We used one such extension, SPARQL$_{\text{Stream}}$ [3], that is able to use Data Stream Management Systems (DSMS), Complex Event Processors (CEP) and Sensor middleware as underlying query processors. In particular, we used GSN [4] (Global Sensor Networks), a widely used sensor data processor to which we added a wrapper to the CityBikes API. One of the advantages of using SPARQL$_{\text{Stream}}$ is that it uses R2RML mappings to rewrite queries into expressions that can be instantiated and executed by streaming query engines [5]. This allowed us to use the same set of mapping definitions for both static and streaming data. We used ODEMapster[11] for static data and SPARQL$_{\text{Stream}}$ for dynamic data. Notice that both approaches follow very different RDF management strategies: for static data we generate materialized RDF triples that can be later queried in a standard triple store; for dynamic data we pose queries to a virtual streaming RDF dataset, and the queries are rewritten by a SPARQL$_{\text{Stream}}$ processor to the underlying stream processing engine, which throws the query results.

When defining the R2RML mappings it was sometimes necessary to define an object map that has a join condition specifying a child and a parent triples map. As the

[10] W3C RDB2RDF Mapping Language: `http://www.w3.org/TR/r2rml/`

[11] ODEMapster RDB2RDF Tool: `http://neon-toolkit.org/wiki/ODEMapster`

generated RDF property flows from child to parent, it was sometimes necessary to define an inverse property in the ontology only for this purpose. This is the case of the property *isStationOf* which is the inverse of the property *hasStation*.

An interesting requirement in our use case is the possibility of accessing historical data from the dynamic observations. In order to perform data analysis or compare live data with historical records, we cannot directly use the SPARQL$_{\text{Stream}}$ approach as it stands, because it only considers recent observation values. As an alternative, we chose to use the live SPARQL$_{\text{Stream}}$ service in order to periodically pose `CONSTRUCT` queries that generate RDF triples, which can be later imported to the static RDF store. Then, users requiring statistical or analysis queries, can directly use such a store.

The linking activity was completed with the Silk platform [6]. The activity consisted in geo-linking the stations with the points of interest and the travel guides. Silk provides a similarity measure for geo-linkage that requires that both datasets be annotated with the WGS_84 ontology. Some of the data sources, e.g., museums in the city of León, that were considered in this use case were not annotated with this vocabulary so it was necessary to use numeric similarity and aggregated euclidean distance to compute the similarity between two entities. It is not clear if this gives us a precise estimate of the proximity of two points and it was difficult to validate the adequacy of the links.

Care was taken to ensure that the publication technologies satisfy the licensing and access policies previously defined.

### 3.4 Exploitation

Queries can be posed against static data, dynamic data, and a combination of both. For dynamic data there is no RDF data generation; instead, queries are rewritten by a SPARQL$_{\text{Stream}}$ processor, to the underlying stream processing engine which throws the query results. Our SPARQL$_{\text{Stream}}$ processor can not yet construct in an optimised manner queries that combine streaming and static data, and this is a pending task for which we will adopt some existing strategy [7, 8]. For historical dynamic data, the SPARQL$_{\text{Stream}}$ service is used to periodically pose CONSTRUCT queries that generate RDF. In our case, we have made the data available in two different types of endpoints (one for static data and the other one for dynamic data), and we will soon provide a map-based interface based on the Map4RDF platform[12]. The complete description of the application is available online[13] along with sample queries.

## 4 Discussion and Open Questions

Some of the main challenges that we have faced in the process of producing and exploiting data in the context of this case study are:

- The SSN ontology has been applied to the domain of bike sharing systems, which falls outside areas where the SSN ontology has been extensively used, such as environmental measurements and agriculture. The ontology has proven to be useful

[12] `http://www.oeg-upm.net/index.php/es/downloads/172-map4rdf`
[13] `http://transporte.linkeddata.es/`

to model the observations and measurements. However, for people not used to the Observation and Measurements approach, which underlies the SSN ontology, this type of modelling is complex to understand. Clear descriptions of usage patterns are needed so that the SSN ontology can be more easily exploited by developers.

– For ontology developers, the SSN ontology still poses some challenges when it has to be extended. For instance, the treatment of properties that are measured by sensors is represented through a class, and when it needs to be extended, it is sometimes unclear how this extension has to be done (e.g., using a subclass or an instance) and it is unclear how properties can be modelled for better reuse across other ontologies and domains.
– There is a need to explore when it makes sense to follow a native RDF stream approach to deal with stream data (e.g., as done in [7, 8]) or an R2RML-based query rewriting approach, as we do here in our work.
– When using the R2RML approach, there are some limitations in the treatment of direct and inverse properties and how they can be defined in the R2RML mappings. This forces ontology developers to overspecify some properties, by defining properties that are inverse to others that are defined. While this does not really represent a major problem from an ontology development perspective, it is important to give clear guidelines to developers in this respect.

## Acknowledgements

This work has been supported by the Ciudad2020 INNPRONTA project (IPT-20111006). We want to thank Boris Villazón-Terrazas and Freddy Priyatna for their help during the RDF generation process.

## References

1. Villazón-Terrazas, B., Vilches-Blázquez, L., Corcho, O., Gómez-Pérez, A.: Methodological guidelines for publishing government linked data linking government data. In Wood, D., ed.: Linking Government Data. Springer New York (2011) 27–49
2. Garijo, D., Villazón-Terrazas, B., Corcho, O.: A provenance-aware linked data application for trip management and organization. In: 7th Int. Conference on Semantic Systems. (2011)
3. Calbimonte, J.P., Corcho, O., Gray, A.J.G.: Enabling ontology-based access to streaming data sources. In: Proc. 9th International Semantic Web Conference. (2010) 96–111
4. Aberer, K., Hauswirth, M., Salehi, A.: A middleware for fast and flexible sensor network deployment. In: 32nd International Conference on Very Large Databases. (2006) 1199–1202
5. Calbimonte, J.P., Jeung, H., Corcho, O., Aberer, K.: Enabling query technologies for the semantic sensor web. Int. J. on Semantic Web and Information Systems (to appear) **8** (2012)
6. Julius, V., Christian, B., Martin, G., Georgi, K.: Discovering and maintaining links on the web of data. In: ISWC2009. (2009) 650–665
7. Barbieri, D.F., Braga, D., Ceri, S., Valle, E.D., Grossniklaus, M.: C-SPARQL: a continuous query language for RDF data streams. Int. J. Semantic Computing **4**(1) (2010) 3–25
8. Le-Phuoc, D., Dao-Tran, M., Parreira, J.X., Hauswirth, M.: A native and adaptive approach for unified processing of linked streams and linked data. In: ISWC2010. (2011)

# Short Paper: Semantic Annotations for Sensor Open Data

Mikel Emaldi, Jon Lázaro, Unai Aguilera, Oscar Peña, and Diego López de Ipiña

Deusto Institute of Technology - DeustoTech, University of Deusto
Avda. Universidades 24, 48007, Bilbao, Spain
{m.emaldi, jlazaro, unai.aguilera, oscar.pena, dipina}@deusto.es

**Abstract.** Since the creation of the Open Data Euskadi (ODE) initiative in 2009, one of its challenges has been the publication of government Open Data following the Linked Data principles. On the other hand, one of the challenges for the Semantic Sensor Web is the integration and fusion of data from heterogeneous sensor networks. In this short paper we present the efforts made at the Bizkaisense[1] project on the alignment of different ontologies with the objective to fulfil these two challenges.

## 1 Introduction

Many governments across the world have realised the importance of opening data both as a service to promote transparency and as a way to enable businesses to make a better use of publicly available information. Open Data Euskadi[2] is a good example of it, in this case fostered by the Basque Country Government in Spain. Despite the efforts made by this initiative towards Linked Data, they have been mainly focused on the publication of raw data, directly taken from the computer systems from different administrations. The fact is that there is a limited number of datasets published as Linked Data or, at least, in any of the RDF serializations; but the good news is that external providers such as research centres or companies can treat raw data and publish them as Open Linked Data.

One of the datasets requiring this treatment is the one containing the data generated by the pollution sensors deployed throughout Basque Country[3]. According to [4] there are five challenges for the Semantic Sensor Web: 1) the first is about the abstraction level of the data extraction, process and management; 2) quality and Quality of Service of sensor data; 3) integration and fusion of sensor data; 4) identification and location of relevant sensor-based data sources; and 5) rapid development of applications. The Bizkaisense project is focused on the accomplishment of the third challenge. We think that the first step to integrate

[1] This research is founded by Bizkailab 2010 program, Bizkaiko Foru Aldundia - Diputación Foral de Bizkaia.

[2] http://opendata.euskadi.net/

[3] http://www.ingurumena.ejgv.euskadi.net/r49-n82/es/vima_ai_vigilancia/indice.apl?lenguaje=c

this Sensor Open Data with heterogeneous data sources is to publish them as Linked Data. In this short paper we present the efforts made at this first step, which are related to the mapping of the raw data to appropriate ontologies and the alignment between them.

The remainder of the paper is organized as follows. Section 2 discusses related work. Section 3 exposes the Open Data available about pollution sensors. Section 4 exposes the adopted solution to map the raw data to appropriate ontologies. Finally, Section 5 concludes and outlines the future work.

## 2 Related Work

Several efforts have been done in the field of semantic sensor networks; in this section we expose some examples. In [2], the SSN and SWEET ontologies are used to model sensor data and to allow a federated query system among them. However, these approaches do not use any ontology to represent the units of the measurements made by sensors. [3] presents a survey about the different ontologies to model different aspects of sensor networks. Linked Stream Middleware (LSM) provides wrappers for real time data collecting and publishing, a web interface to visualize and publish data and a SPARQL endpoint for querying data from heterogeneous sensor networks in an unified way [7]. In LSM, the user can import different ontologies to represent her/his sensors. This approach allows to annotate a wide variety of sensors, however, it can be an obstacle to manage the interoperability among different sensor networks. Related with the usage of custom ontologies, [1] uses its own ontology to represent the location of the sensors with a high granularity (floor, room, etc.). In [5], they map the sensors of Android powered smartphones to SSN and DUL ontologies, extending SSN with the proper instances to represent these sensors. The approach presented by [10] extends SSN with a collection of observations that their sensor network observes.

The AEMET Linked Data[4] project has a strong relationship with our Bizkaisense project. In this project the weather stations of AEMET (Agencia Estatal de Meteorología) have been annotated semantically. The data related to these stations have been extracted from CSV files provided by AEMET[5]. To annotate these weather stations, they have combined SSN ontology with *aemetonto*, a custom ontology made for the project which is used to represent the different meteorological phenomena that the stations can measure.

Summarizing, we can see that although there are many projects related to semantic sensor networks, usually, the way to fulfil the limitations of the ontologies is to introduce new custom ontologies, instead of reingenieering existing ontological resources. Similar reingeneering work can be seen at [5] or [10]. We think that this second approach, which is adopted by Bizkaisense, is more suitable to achieve the interoperability among heterogeneous sensor networks.

[4] `http://aemet.linkeddata.es/`
[5] `ftp://ftpdatos.aemet.es/datos_observacion/observaciones_diezminutales/`

## 3 Pollution Sensoring in Basque Country

There are 72 pollution sensors set up along the Basque Country, managed by the Basque Government. These sensors measure the air quality based on: chemical substances (xylene, sulfur dioxide, toluene, carbon monoxide, ozone, particulate matter (10 and 2,5 $\mu g/m^3$), nitrogen dioxide, hydrocarbons, hydrogen sulfide, ammonia, ethylbenzene, volatile organic compounds, benzene and smoke) and solar and ultraviolet radiation. They also measure different atmospheric and meteorological phenomena like wind speed and direction, temperature, barometric pressure and humidity. The data generated by these sensors is very useful, for example, to track the evolution of the air quality over time. However, the provided raw data is very difficult to analyse without the possibility of making complex queries over it. To solve this issue, all of these features have been semantically annotated as can be seen at Section 4.

The data gathered by these sensors can be accessed in two different ways: 1) through the historical records stored into CSV files since 2000 and 2) through real-time data published at each sensor's web page[6] extracted via web-scraping techniques. Two simple Python scripts have been built to parse both data sources. The generated RDF data is stored into an OpenLink Virtuoso semantic store and served through Pubby Linked Data interface. We do not go into this process of data transformation and publication in any depth because this paper focuses into the work done with ontologies used along the project.

## 4 Semantic Annotation of Pollution Sensors

Different ontologies have been used to semantically annotate these sensors' data, as depicted in Figure 1. The main ontology of the model is SSN (Semantic Sensor Network Ontology) [8]. This ontology, developed by the W3C Semantic Sensor Incubator Group, is used in Bizkaisense to annotate different aspects of sensors and their measurements. The location of a sensor is represented by an instance of the *Point* class from WGS84 Vocabulary[7] through `dul:hasLocation` property, and it is linked with the nearest *Feature* instance of GeoNames[8] through `dul:nearTo` property.

More reingenieering work has been done extending SWEET and MUO ontologies. SWEET 2.3 (Semantic Web for Earth and Environmental Terminology) is a collection of ontologies that describes both orthogonal concepts (space, time, physical quantities, etc.) and integrative science knowledge concepts (phenomena, events, etc.) [9]. Although the ontology is very complex and describes a wide variety of chemical substances, there are some concepts that it does not include. Concretely, we have extended the ontology[9] with classes representing *Ethylbenzene ($C_6H_5C_2H_5$)* and *HydrogenSulfide ($H_2S$)*, as can be seen in Figure 2.

[6] `http://www.ingurumena.ejgv.euskadi.net`
[7] `http://www.w3.org/2003/01/geo/`
[8] `http://www.geonames.org/`
[9] `http://helheim.deusto.es/bizkaisense/sweetAll-extended.owl`

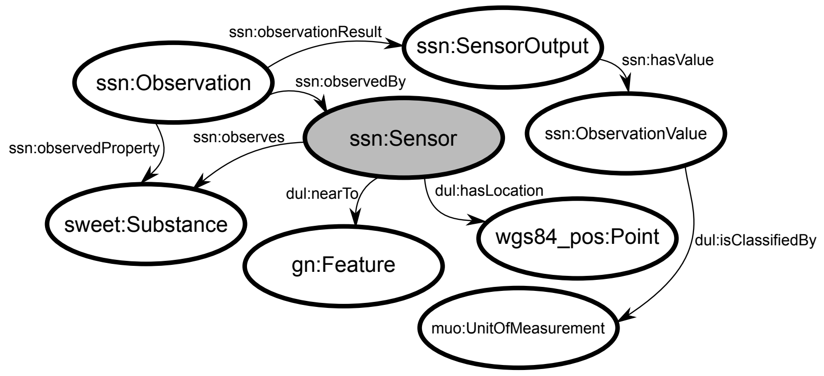


**Fig. 1.** Aligning the SSN ontology with SWEET, GeoNames, MUO, DUL and WGS84 ontologies.

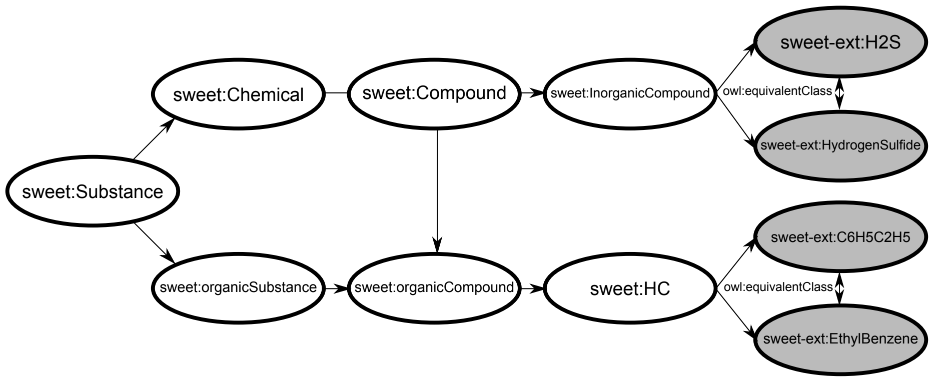


**Fig. 2.** Classes added to SWEET (colored) and their partial class hierarchy.

Regarding to MUO ontology (Measurement Units Ontology)[10], we have extended its instances[11] from the data extracted from UCUM (Unified Code for Units of Measure)[12]. These new instances are *cubic-meter, cubic-squared, microgram, millibar, milligram, milliwatt, meter-per-second, micro-gram-per-cubic-meter, milligram-per-cubic-meter, milliwatt-per-squared-meter, watt-per-squared-meter and watt-per-squared-meter*, as can be seen in Figure 3.

Finally, the Dublin Core [6] vocabulary is used to annotate common attributes like dates, titles, descriptions and so on. Code 1 shows an example of an observation made by a pollution sensor. More examples can be found at the project web page[13] and its SPARQL endpoint[14].

[10] `http://idi.fundacionctic.org/muo/muo-vocab.html`
[11] `http://helheim.deusto.es/bizkaisense/ucum-ext.owl`
[12] `http://idi.fundacionctic.org/muo/ucum-instances.html`
[13] `http://helheim.deusto.es/bizkaisense/`
[14] `http://helheim.deusto.es/bizkaisense/sparql`

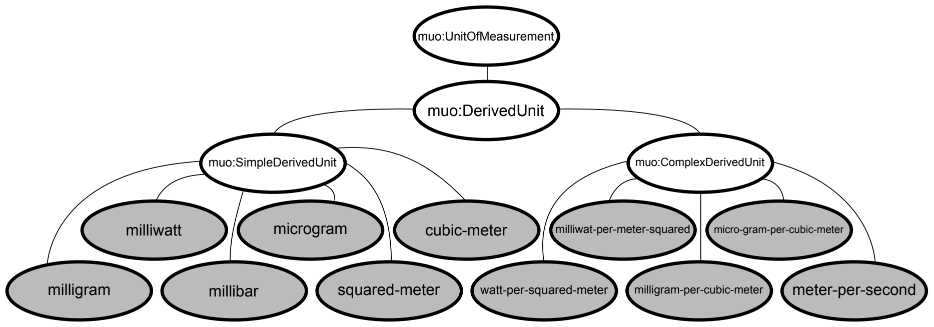


**Fig. 3.** Instances added to UCUM. The prefix `ucum-ext:` has been omitted from instances to ease the comprehension of the figure.

**Code 1** Example of an observation made by a pollution sensor.

```
@prefix bizkaisense:   <http://helheim.deusto.es/bizkaisense/resource/station/> .
@prefix observation:   <.../bizkaisense/resource/station/ELCIEG/NO/01012011/00#> .
@prefix ssn:           <http://purl.oclc.org/NET/ssnx/ssn#> .
@prefix sweet:         <http://http://sweet.jpl.nasa.gov/2.3/propSpeed.owl#> .
@prefix dul:           <http://www.loa-cnr.it/ontologies/DUL.owl#> .
@prefix ucum-extended: <http://helheim.deusto.es/bizkaisense/ucum-extended.owl#> .

observation:             rdf:type               ssn:Observation ;
                         dc:date                "2011-01-01T00:00:00" ;
                         ssn:observedProperty   sweet:NO ;
                         ssn:observationResult  observation:sensoroutput ;
                         ssn:observedBy         bizkaisense:ELCIEG .
observation:sensoroutput rdf:type               ssn:SensorOutput ;
                         ssn:hasValue           observation:outputvalue .
observation:outputvalue  rdf:type               ssn:ObservationValue ;
                         dul:hasDataValue       3 ;
                         dul:isClassifiedBy     ucum-extended:microgram-per-cubic-meter .
```

## 5 Conclusion and Future Work

On this paper we have described the efforts made to semantically annotate the Sensor Open Data provided by Open Data Euskadi. These efforts include the analysis of different ontologies from the domains of sensor networks, chemistry and meteorology; and the extension of these ontologies to fulfil all the requirements of these pollution sensors. Even though the SWEET ontology belongs to a concrete domain, the extension of UCUM instances of MUO ontology can be reused in a wide variety of cross-domain projects. In addition, this semantic representation of pollution sensors allows us to make complex queries over their data, e. g. the queries used in Bizkaisense for calculating averages of certain substances in a region, as the one we can see in Code 2. Furthermore, this semantic model can be adopted by other sensor networks of the same domain. The approach of extending existing ontologies in contrast of creating new ad-hoc ontologies allows the interoperability among different sensor networks. On the other hand, this paper demonstrates the usefulness of Open Data platforms like ODE.

**Code 2** Example of a complex query used in Bizkaisense.

```
SELECT (AVG(?value) as ?avg) WHERE {
    ?medition ssn:observedBy       ?station .
    ?station  dul:nearTo           <http://sws.geonames.org/3104499> .
    ?medition dc:date              ?date .
    ?medition ssn:observationResult ?res .
    ?medition ssn:observedProperty  sweet:NO .
    ?res      ssn:hasValue         ?val .
    ?val      dul:hasDataValue     ?value .
    ?val      dul:isClassifiedBy    ?obsunit .
    FILTER (xsd:dateTime(?date) >= xsd:dateTime("2011-02-17T00:00:00")) .
    FILTER (xsd:dateTime(?date) <= xsd:dateTime(("2011-02-21T00:00:00"))) . }
```

The next goal in Bizkaisense is the integration of data about pollution sensors with other data sources related to environmental domain, like solid and liquid wastes production of Basque Country. With the integration of more data sources we expect to appeal the experts of the domain to increase the features of the system and to demonstrate the real value of the Sensor Open Data vision.

## References

1. Barnaghi, P., Presser, M., Moessner, K.: Publishing linked sensor data. In: Proceedings of the 3rd International Workshop on Semantic Sensor Networks (2010)
2. Calbimonte, J.P., Jeung, H., Corcho, O., Aberer, K.: Semantic sensor data search in a large-scale federated sensor network. 4th International Workshop on Semantic Sensor Networks (2011)
3. Compton, M., Henson, C., Lefort, L., Neuhaus, H., Sheth, A.: A survey of the semantic specification of sensors. Proc. Semantic Sensor Networks 17 (2009)
4. Corcho, O., García-Castro, R.: Five challenges for the semantic sensor web. Semantic Web 1(1), 121–125 (2010)
5. d'Aquin, M., Nikolov, A., Motta, E.: Enabling lightweight semantic sensor networks on android devices. 4th International Workshop on Semantic Sensor Networks (2011)
6. Initiative, D.C.M.: Dublin core metadata element set, version 1.1: Reference description (1999), `http://dublincore.org/documents/1999/07/02/dces/`
7. Le-Phuoc, D., Quoc, H.N.M., Parreira, J.X., Hauswirth, M.: The linked sensor Middleware–Connecting the real world and the semantic web. Proceedings of the Semantic Web Challenge (2011)
8. Lefort, L., Henson, C., Taylor, K., Barnaghi, P., Compton, M., Corcho, O., Garcia-Castro, R., Graybeal, J., Herzog, A., Janowicz, K.: Semantic sensor network XG final report. W3C Incubator Group Report (2011)
9. Raskin, R.G., Pan, M.J.: Knowledge representation in the semantic web for earth and environmental terminology (SWEET). Computers & Geosciences 31(9), 1119–1125 (2005)
10. Stasch, C., Schade, S., Llaves, A., Janowicz, K., Bröring, A.: Aggregating linked sensor data. 4th International Workshop on Semantic Sensor Networks p. 46 (2011)

# Semantic Processing of Sensor Event Stream by Using External Knowledge Bases
## Short Paper

Kia Teymourian and Adrian Paschke

Freie Universitaet Berlin, Berlin, Germany
{kia, paschke}@inf.fu-berlin.de

**Abstract.** Usage of domain background knowledge about sensor data can improve the expressiveness and flexibility of event processing in sensor network applications. Huge amount of domain background knowledge stored in external knowledge bases can be processed in combination with sensor data stream in order to achieve more knowledgeable event processing. In this paper, we discuss the benefits of background knowledge for event processing and describe a simple classification of event query rules.[1]

**Keywords:** Knowledge-Based Complex Event Processing

## 1 Motivation

Detection, prediction and mastery of complex situations are crucial to the competitiveness of networked businesses, the efficiency of Internet of Services and dynamic distributed infrastructures in manifold domains such as logistics, automotive, telecommunication, e-health and life sciences. Event Processing is an emerging technology to achieve actionable, situational knowledge from large-scale event streams in real-time.

Semantic models of events can improve event processing quality by using event meta-data in combination with ontologies and rules (knowledge bases). The successes of the knowledge representation research community in building standards and tools for technologies such as formalized and declarative rules, are opening novel research and application areas. The combination of event processing and knowledge representation can lead to novel semantic-rich event processing engines. The identification of critical events and situations requires processing vast amounts of data and metadata within and outside the systems.

Using external background knowledge about sensor data stream is one of the promising approaches for detection of real-world complex events. Knowledge about event types and their hierarchies, i.e., specialization, generalization, or

[1] This work has been partially supported by the "InnoProfile-Corporate Semantic Web" project funded by the German Federal Ministry of Education and Research (BMBF) and the BMBF Innovation Initiative for the New German Länder - Entrepreneurial Regions.

other forms of relations between events can be useful. Huge amount of background knowledge about sensor event data can be integrated with the main data stream to improve the expressiveness of event processing by inferencing on background knowledge. Semantic inference is used to infer relations between events such as, e.g., transitivity or equality between event types and their properties. Temporal and spatial reasoning on events can be done based on their data properties.

In the rest of this paper, we describe in Section 2 the benefits of using external knowledge bases for event processing and in Section 3 different event query categories are discussed.

## 2 Knowledge-Based Complex Event Processing

Previously, we proposed in [7,6] a new approach for semantic enabled complex event processing (SCEP). We proposed that the usage of the background knowledge about events and other related concepts can improve the quality of event processing. We described how to formalize complex event patterns based on a logical knowledge representation (KR) interval-based event/action algebra, namely the interval-based Event Calculus [2,3,4].

The fusion of background knowledge with data from an event stream can help the event processing engine to know more about incoming events and their relationships to other related concepts. We propose to use one or several *Knowledge Bases (KB)* which can provide background knowledge (conceptual and assertional, T-Box and A-Box of an ontology) about the events and other non-event resources. This means that events can be detected based on reasoning on their type hierarchy, temporal/spatial relationships, or their relationship to other objects in the application domain. The connections to other relevant concepts and objects means for example the relationship of a food item to a particular drug.

The benefits of using background knowledge in complex event processing can be seen as two major advantages over the state of the art CEP systems. The first benefit is its higher *expressiveness* and the second one its *flexibility*. Expressiveness means that an event processing system can precisely express complex event patterns and reactions to events which can be directly translated into business operations. Flexibility means that a CEP system is able to integrate new business changes into the systems in a fraction of time rather than changing the whole event processing rules. Complex event patterns are independent of current businesses and are defined in a higher level of abstraction based on business strategies. When something is changed in the business environment, it can be considered simply as an update in the background knowledge and the complex event detection patterns which are defined based on the business plans should not be changed.

In many event processing use cases, the amount of background knowledge about events and the relevant objects can be very high, so that they can not be included as rule sets in the main memory of event processing agents for reasoning. Therefore, we propose to use external KBs for the storage and reasoning on background knowledge. The background knowledge about events and other non-event concepts/objects is described in description logic. The knowledge in the KB can be stored in the Resource Description Framework (RDF) data format[2] in an external triple store (special kind of databases for storage and management of RDF data). This knowledge can be queried from the event processing agents based on the demands of the event query rules. The external KB includes a description logic inference engine to reason on the relations between events and other relevant non-event objects in the application domain. The KB can be queried by using SPARQL [5] queries and the results are then included in the event processing engine.

## 3 Event Query Rules and Their Categories

Event query rules are declarative rules which are used to detect complex events from streams of raw events. The aggregated knowledge from event streams and background KB can be queried by different types of event queries. These event queries have a hybrid semantic, because they use event operation algebra to detect events and they use SPARQL queries to include background knowledge about these events and their relationships.

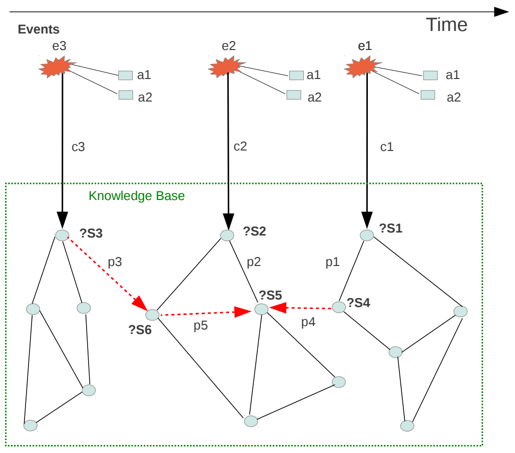
Lets consider an Event type $E_1$ which can be instantiated with $n$ (attribute, value) tuples like: $e_1^1((a_1, v_1), \ldots, (a_n, v_n))$. The figure 1 shows the event stream and the relationships of events to resources in the background knowledge. An event instance $e_1$ can be connected to one or more resources in the background knowledge by using a connecting predicate $c_1$ using one or more attribute value pairs of the event instance.

We use attribute-value tuples of an incoming event instance to query an external KB about the relevant background knowledge of that event instance. These tuples are used to build basic graph patterns (BGPs) which are used in SPARQL queries as sets of triple patterns defined in SPARQL queries. The usage of SPARQL queries allows the event processing agent to include external knowledge and combine it with the event stream to know more about the incoming events.

Our event query rules allow simple event algebra operations, similar to Snoop [1] (i.e. event operations like AND, SEQ, OR, NOT), to query the event stream as well as higher interval-based event operations like (BEFORE, MEETS, OVERLAP, . . . ). Our event query rules can include SPARQL query predicate to query external KBs. The results of SPARQL queries are used in combination with event stream to detect complex events. This means that a complex event pattern is defined based on the event operation algebra in combination with SPARQL queries (basic graph patterns plus inferencing on knowledge graph).

One event detection pattern of the relationship shown in the figure 1 can be represented by the given pseudocode. The event e1 is connected to the resource

[2] http://www.w3.org/RDF/



```
{ { (?e1, c1, ?s1) .
    (?s1, p*, ?s) . }
% SEQ event algebra operation
[?e1 SEQ ?e2][Within 5 min.]

 { (?e2, c2, ?s2) .
   (?s2, p*, ?s) . } }

[?e2 SEQ e3] [Within 5 min.]

 { (?e3, c3, ?s3) .
   (?s3, p*, ?s) . } }
```

**Fig. 1.** Relation of Events to Resources in the Background Knowledge and Pseudocode of Event Detection Pattern

s1 in the background knowledge by the predicate c1. In the same way the event e2 is connected to the resource s2 by predicate c2. The predicate p4 connect the two resources s4 and s5, so that it connects the two sub-graphs.

The above query can written in declarative rules for event detection. The precise semantics of event query rules is based on interval-point semantics [6] which allows us to include event algebra operators based on time-point semantics. This kind of event query rules can also include SPARQL query predicate to query external KBs. The semantics of the whole event query is a hybrid semantic of description logic and event operation algebra which defines the semantics of event detection.

Event query rules can be categorized into several categories based on the usage of knowledge queries (SPARQL queries) inside the event query rule. Our criteria for these categorization are based on the following different factors: 1. Number of SPARQL queries in each event processing step are sent to the KB *(Single SPARQL or Several SPARQLs)* 2. Whether the SPARQL query depends on incoming event data and is generated based on their attributes or is independent and describes only some attributes or the type of events. *(Generated or Not Generated)* 3. By Generated SPARQL queries from event attributes, the number of event attributes used for generating SPARQL query *(Single attribute or Several attributes)* 4. By generated SPARQL queries whether they are generated from several events in a sliding window or they are generated from a single event instance. *(Sliding Window or Single Event)*

In this paper, we describe the most important and interesting categories of event query rules. This categorization is not a complete classification of all possible rule combinations, our aim is more to emphasize interesting rule combinations which can be processed using different event processing approaches. Our implementation of these event query rules and our initial experiments with these rules are described in [8].

**Category A - Single SPARQL Query:** In this category, the event query rule includes only one single knowledge query and uses its results in one or more variables within the event detection rule. A SPARQL query is used to import knowledge about event instances or types. One or more attributes of events are used to build the basic triple pattern inside the SPARQL query. Category A event query rules can be categorized into three subcategories:

**Category A1 - Raw SPARQL:** This category of event query rule is the simplest form of these event query rule. The included SPARQL query is only about the resources in the background knowledge. The background knowledge query is independent from the event stream, however the complex event detection is defined on the results of this query in combination with the event stream. In some cases, on each event the SPARQL query should be resent to the KB to update the latest results from the KB.

**Category A2 - Generated SPARQL:** In this category of event query rules with each incoming event a different SPARQL query is generated and sent to the target knowledge base. The attribute/values of an event instance are used to generate basic triple patterns of a SPARQL query. Based on user definitions some of the tuples (attribute, value) of an event instance are selected and used to generate a single SPARQL Query.

**Category A3 - Generated SPARQL from Multiple Events:** The query is similar to A2, but the SPARQL query is generated from multiple events. Within a data window (e.g., a sliding time window) from two or more events a single SPARQL query is generated. Multiple events are used to generate the single SPARQL query, the event processing waits for receiving some new events and then generate a SPARQL query based on the emitted events, and query for the background knowledge about them.

**Category B - Several SPARQL Queries:** Queries of this category include several SPARQL queries and combine them with event detection rules. This means that several A category rules are combined together which can build a category B. The category B of rules are able to combine results from KBs with events using event operation algebra.

**Category B1 - Several SPARQL Queries in AND, OR and SEQ Operations:** The category B1 is based on the category B, but the results from the SPARQL query predicates are combined with AND, OR, SEQ or similar event algebra operations. The whole query is evaluated on sliding windows of event streams. The SPARQL query predicates are not depending on each other, i.e., the results from one is not used in another SPARQL predicate, so that they are not depending on the results of the other SPARQL query.

**Category B2 - Chaining SPARQL Queries:** In category B2 several SPARQL queries are generated and executed in sequence. They can be generated based on the results of the previous SPARQL query. Each SPARQL query can be generated from a set of events (e.g., included in a slide of event stream by means of a sliding window, a counting or timing window). This means that different data windows can be defined to wait until some events happened and then a SPARQL query is executed. SPARQL queries might be defined in a sequence chain. The results are directly used for event processing or used in another following SPARQL query.

**Category B3 - Chained and Combined SPARQL Queries:** In this category SPARQL queries are used in combination with all possible event algebra operations like, AND $\bigwedge$, OR $\bigvee$ , SEQ $\bigoplus$, Negation $\neg$ , etc. The event operations are used for combining the results from several SPARQL queries or several SPARQL queries are used in combination with event algebra operations like: $((sparql_1 \bigoplus sparql_2) \bigwedge sparql_3 \bigvee \neg sparql_4)$. This category of event query rules is the general form of queries and has the highest possible complexity, because the results from external KBs are used in combination with event operations or the attribute/values from incoming events are used for generation of complex SPARQL queries.

## 4 Conclusion and Future Work

In this paper, we have the different categories of event query rules which use special rule predicates for importing data from external KBs and its combination with event algebra operations. Our future steps are to work on details of different event processing algorithms for each of the different rule categories, e.g. by rewriting complex event query to several simple queries which can be distributed over an event processing network.

## References

1. Chakravarthy, S., Mishra, D.: Snoop: an expressive event specification language for active databases. *Data Knowl. Eng.*, 14:1–26, November 1994.
2. Paschke, A.: Eca-lp/eca-ruleml: A homogeneous event-condition-action logic programming language. In *RuleML-2006*, Athens, Georgia, USA, 2006.
3. Paschke, A.: Eca-ruleml: An approach combining eca rules with temporal interval-based kr event/action logics and transactional update logics. *CoRR*, abs/cs/0610167, 2006.
4. Paschke, A., Bichler, M.: Knowledge representation concepts for automated sla management. *Decis. Support Syst.*, 46(1):187–205, 2008.
5. Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF. W3C Recommendation, 2008.
6. Teymourian, K., Paschke, A.: Semantic rule-based complex event processing. In *RuleML 2009: Proceedings of the International RuleML Symposium on Rule Interchange and Applications*, 2009.
7. Teymourian, K., Paschke, A.: Towards semantic event processing. In *DEBS '09: Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, pages 1–2, New York, NY, USA, 2009. ACM.
8. Teymourian, K., Rohde, M., Paschke, A.: Fusion of background knowledge and streams of events. In *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems.* ACM.

# Demonstration: Dynamic Sensor Registration and Semantic Processing for ad-hoc MOBile Environments (SemMOB)

Pramod Anantharam, Alan Smith, Josh Pschorr, Krishnaprasad Thirunarayan, and Amit Sheth

Ohio Center of Excellence in Knowledge-Enabled Computing (Kno.e.sis), Wright State University, Dayton, USA
{pramod,alan,josh,tkprasad,amit}@knoesis.org
http://knoesis.org

**Abstract.** SemMOB enables dynamic registration of sensors via mobile devices, search, and near real-time inference over sensor observations in ad-hoc mobile environments (e.g., fire fighting). We demonstrate SemMOB in the context of an emergency response use case that requires automatic and dynamic registrations of sensor devices and annotation of sensor observations, decoding of latitude-longitude information in terms of human sensible names, fusion and abstraction of sensor values using background knowledge, and their visualization using mash-up.

## 1 Introduction

Mobile devices are pervasive and increasingly becoming powerful (in terms of memory, processing power, on-board sensors). They have access to two sources of sensor information (a) internal sensors (embedded into mobile devices), and (b) external sensors (present in the vicinity of mobile devices), both reporting observations of the physical world. Only way to understand the physical world from observations is to associate semantics with the observations by grounding them in events in the physical world. For example, sensor observations about temperature, $CO_2$, and wavelength of light can be used to infer a *fire type* event. This can be used by the first responders to decide on the type of extinguisher to be used. SemMOB is a system that embodies such semantic processing of observations from ad-hoc mobile sensors. It extends the capability of the SemSOS [1] for mobile devices while preserving the use of semantics in the form of background knowledge. Our prototype showcases the capabilities of SemMOB using various sensors reporting observations through an Android device.

The rest of paper is organized as follows. In Section 2, we describe the system architecture and review SemSOS and the reasoner. After we describe the nature of queries that can be issued to SemMOB and its visualization capabilities in Section 3, we conclude in Section 4.

## 2 Architecture

SemMOB has two major components: (a) SemSOS and (b) Reasoning engine, as shown in Figure 1. Mobile devices with an Android client registers with SemSOS using the standard SOS register sensor request. There are internal sensors such as GPS, gyroscope and camera, and external sensors such as activity, temperature, and gaseous sensors, reporting observations to the mobile device via bluetooth. All observations are sent to SemSOS and annotated using the SSN-XG ontology [3, 4] for further reasoning. SemSOS backend also has background knowledge about sensors described in SSN-XG ontology to support reasoning. The reasoner fetches observations and sensor metadata from the knowledge base, reasons over them, and puts back the inferred triples into the knowledge base. The first responders can monitor the sensor observations using the real-time observation monitor or the query interface as shown in Figure 1.

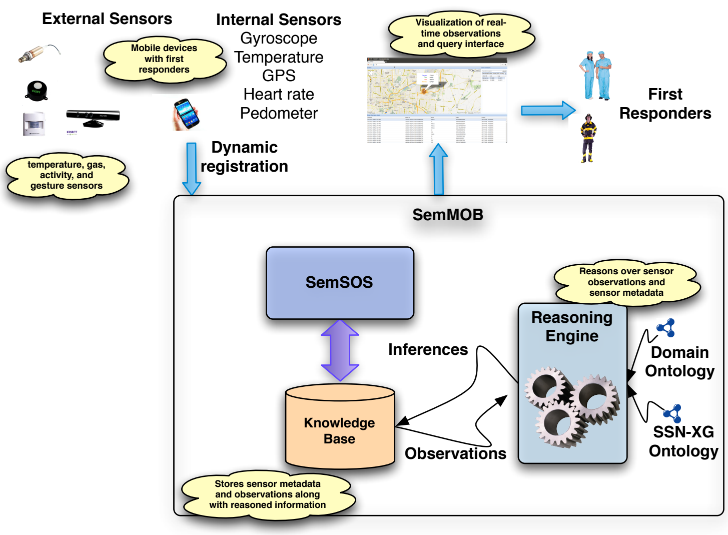

External Sensors
Internal Sensors
Gyroscope
Temperature
GPS
Heart rate
Pedometer
Mobile devices with first responders
temperature, gas, activity, and gesture sensors
Visualization of real-time observations and query interface
Dynamic registration
First Responders
SemMOB
SemSOS
Inferences
Knowledge Base
Observations
Reasons over sensor observations and sensor metadata
Reasoning Engine
Domain Ontology
SSN-XG Ontology
Stores sensor metadata and observations along with reasoned information

**Fig. 1.** SemMOB System Architecture

### 2.1 SemSOS

SemSOS provides a semantic layer over Sensor Observation Service (SOS). SOS is a XML-based standardization of sensor data discovery and access on the web proposed by Semantic Web Enablement (SWE) group of Open Geospatial Consortium (OGC). SemSOS enhances SOS deployments to be semantically rich. SemSOS provides the same interface as SOS while it replaces the backend for processing semantically rich metadata associated with the sensors and observations using a semantic database.

In this work, we have further extended SemSOS to accommodate and integrate diverse mobile devices which enter the sensor network in an ad hoc manner and is required to be integrated. Each mobile device may have different capabilities and different sensors connected to it. SemMOB deals with the sensor network dynamism, sensor diversity, and near real-time reasoning over sensor metadata and its observations.

### 2.2 Reasoning Engine

The reasoner fetches newly inserted sensor metadata and sensor observations from the knowledge base and inserts inferred triples to the knowledge base so that it can be queried by semantically intelligent clients. Part of the ontology for modeling sensor capability along with its location is shown in Figure 2. Once a sensor registers with SemMOB, the reasoner uses latitude and longitude of the sensor to obtain the named location from GeoNames [2] dataset. The reasoner module can be used to reason about *fire types* using observations such as temperature, $CO_2$ level, and wavelength of light emitted using appropprite background knowledge.

Other reasoning procedures may use knowledge of poisonous gases, fire extinguishers, etc., to decide emergency operation related equipments and chemicals. In the context of fire fighting, we modeled the sensors (e.g., gaseous sensors) by extending the SSN-XG ontology [3, 4].

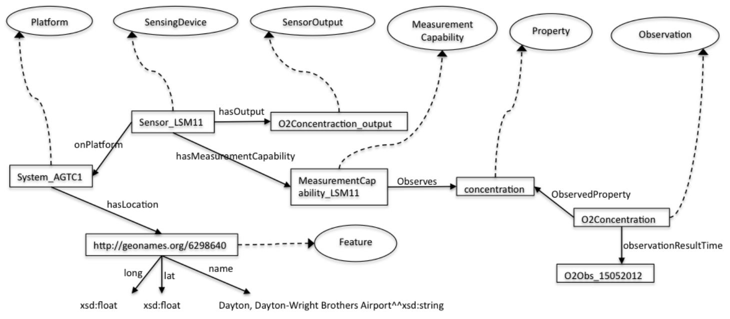


**Fig. 2.** SSN-XG ontology extension to model gaseous concentration

## 3 System Capabilities and Visualization

The SemMOB system users can (1) monitor sensor observations in near real-time, and visualized on a map, (2) reason over sensor observations, and (3) query sensor metadata, sensor observations, and inferred knowledge. SemMOB provides unique capability of dynamic registration, search, reasoning, visualization, and querying of ad hoc mobile sensors in a unified framework. SemMOB query interface and the Android application is shown in Figure 3.

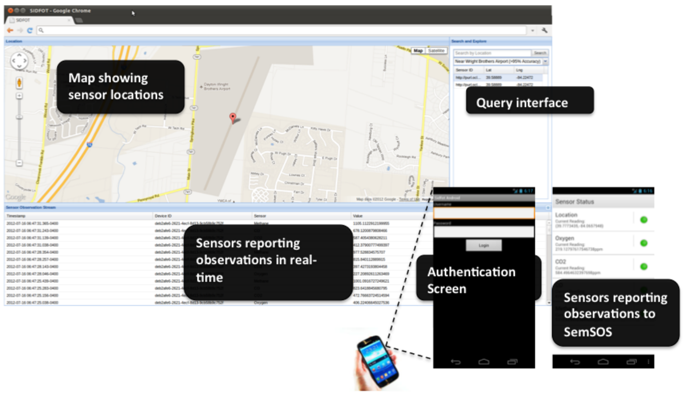

Map showing sensor locations
Query interface
Sensors reporting observations in real-time
Authentication Screen
Sensors reporting observations to SemSOS

**Fig. 3.** Visualization of sensor observations and reasoned information along with query interface

## 4 Conclusions

We demonstrated the functionalities of SemMOB including dynamic registration, querying, reasoning, and visualization of observations from mobile ad-hoc devices and sensors attached to them.

## References

1. Henson, C.A., Pschorr, J.K., Sheth, A.P., Thirunarayan, K.: Semsos: Semantic sensor observation service. In: Proceedings of the 2009 International Symposium on Collaborative Technologies and Systems. CTS '09, Washington, DC, USA, IEEE Computer Society (2009) 44–53
2. GeoNames: Geonames web service and dataset. `http://www.geonames.org/`
3. Compton, M., Barnaghi, P., Bermudez, L., Garcia-Castro, R., Corcho, O., Cox, S., Graybeal, J., Hauswirth, M., Henson, C., Herzog, A., Huang, V., Janowicz, K., Kelsey, W.D., Phuoc, D.L., Lefort, L., Leggieri, M., Neuhaus, H., Nikolov, A., Page, K., Passant, A., Sheth, A., Taylor, K.: The ssn ontology of the w3c semantic sensor network incubator group. Web Semantics: Science, Services and Agents on the World Wide Web **0**(0) (2012)
4. Lefort, L., Henson, C., Taylor, K., Barnaghi, P., Compton, M., Corcho, O., Castro, R.G., Graybeal, J., Herzog, A., Janowicz, K., Neuhaus, H., Nikolov, A., Page, K.: Semantic Sensor Network XG Final Report. Technical report, W3C Semantic Sensor Network Incubator Group (SSN-XG) (June 2011)