Open Geospatial Consortium

Publication Date: 2016-08-20

Approval Date: 2016-08-20

Posted Date: 2016-08-30

Reference number of this document: OGC 15-118

Reference URL for this document: http://www.opengis.net/doc/PER/IMIS Profile Recommendations for OGC Web Services ER

Category: Draft Engineering Report

Editors: Simon Jirka;Christoph Stasch

Incident Management Information Sharing Profile Recommendations for OGC Web Services Engineering Report

The research in this presentation was conducted under contract with the U.S. Department of Homeland Security (DHS) Science and Technology Directorate (S&T), contract # HSHQDC-13-C-00119. The opinions contained herein are those of the contractors and do not necessarily reflect those of DHS S&T.

Document type:	OGC® Engineering Report
Document subtype:	NA
Document stage:	Approved for public release
Document language:	English

License Agreement

Contents

Page

Open Geospatial Consortium i

Incident Management Information Sharing Profile Recommendations for OGC Web Services Engineering Report i

Abstract vii

Business Value vii

Keywords vii

Testbed-11 Incident Management Information Sharing Profile Recommendations for OGC Web Services Engineering Report 1

1 Introduction 1
 1.1 Scope 1
 1.2 Document Contributor Contact Points 1
 1.3 Revision History 2
 1.4 Future Work 2
 1.5 Foreword 2

2 References 3

OGC 06-121r3, *OGC® Web Services Common Standard* 3

3 Terms and Definitions 3

4 Conventions 3
 4.1 Abbreviated Terms 3
 4.2 Unified Modeling Language Notation 5

5 Overview of Existing Standards 6
 5.1 Overview on Data Models and Encoding Standards 6
 5.2 Overview on Service Interface Standards 7

6 Catalog Service for the Web 9
 6.1 Implemented Solution (HubCat) for Dynamic Registration and Discovery of Things 9
  6.1.1 Compusult Implementation 9
 6.2 Pros/Cons 13
  6.2.1 Compusult 13
  6.2.2 Envitia 13
 6.3 Recommended Changes 14
  6.3.1 CSW SensorThings API Profile 14
  6.3.2 Service Object Model 14
  6.3.3 Catalog WMS Service 14

7 Web Feature Service (WFS) 17

7.1 Implemented Solution 17
7.1.1 52°North 17
7.2 Pros/Cons 18
7.2.1 52°North 18
7.3 Recommended Changes 19

8 Web Maps Service (WMS) 20
8.1 Implemented Solution 20
8.1.1 52°North 20
8.1.2 Compusult 22
8.2 Pros/Cons 26
8.2.1 52°North 26
8.2.2 Compusult 26
8.3 Recommended Changes 26

9 Web Processing Service (WPS) 27
9.1 Implemented Solution 27
9.1.1 52°North 27
9.1.2 Compusult 28
9.1.3 University of Melbourne 30
9.2 Pros/Cons 35
9.2.1 52°North 35
9.2.2 Compusult 35
9.2.3 UM 36
9.3 Recommended Changes 36
9.3.1 WEPS 36

10 Sensor Observation Service (SOS) 38
10.1 Implemented Solution 38
10.1.1 Compusult 38
10.1.2 UM 38
10.1.3 OpenSensorHub (OSH) 41
10.1.4 52°North (Client Only) 43
10.2 Pros/Cons 44
10.2.1 Compusult 44
10.2.2 UM 44
10.2.3 OSH 45
10.2.4 52°North 46
10.3 Recommended Changes 46
10.3.1 SOS/Sensor Things Profiling 46
10.3.2 Visualization of Sensors 47
10.3.3 Determine Available Observation Data 47
10.3.4 OWS Context Document Extension Suggestions 47

11 SensorThings API (STA) 51
11.1 Implemented Solution 51

11.1.1 Compusult 51
11.1.2 SensorUp 51
11.2 Pros/Cons 54
11.2.1 Compusult 54
11.2.2 SensorUp 54
11.3 Recommended Changes 55
11.3.1 Discovery of URL and Port Number of MQTT Assoicates 55
11.3.2 Navigation Links 55
11.3.3 Relationship to the SOS Standard 55
11.3.4 Harmonization with Similar Activities 55

12 Observations and Measurements (O&M) - XML Encoding 56
12.1.1 Implemented Solution 56
12.2 Pros/Con 56
12.2.1 Pros 56
12.2.2 Cons 56
12.3 Recommended Changes 56

13 Data Streaming with SWE Common Data 57
13.1 Implemented Solution 57
13.2 Pros/Cons 57
13.2.1 Pros 57
13.2.2 Cons 57
13.3 Recommended Changes 58

14 Sensor Model Language (SensorML) 59
14.1 Implemented Solution 59
14.2 Pros/Cons 59
14.2.1 Pros 59
14.2.2 Cons 60
14.3 Recommended Changes 60

Annex A Extensible Markup Language (XML) Example 61

A.1 General 61

A.2 Web Feature Service (WFS) Capabilities Example 61

A.3 Observations and Measurements (O&M) Chemical Measurement Example 68

A.4 O&M Pedestrian Count Example 68

Bibliography 69

Figures

Page

Figure 1: Possible Model 14
Figure 2: UML Component Diagram the Implementation of the WFS 17
Figure 3: Sequence Diagram of the Interactions Between WFS and SOS 18
Figure 4: Coupling of SOS and WFS (Source: OGC 12-006; p. 147) 19
Figure 5: UML Component Diagram of 52°North's WMS Implementation 20
Figure 6: Sequence Diagram of Interactions Between Clients, WMS, WFS and SOS 22
Figure 7: Compusult WMS Output 23
Figure 8: Layer Manager 24
Figure 9: Feature Information Page 25
Figure 10: Event Processing Architecture Based on the WEPS 27
Figure 11: Insert Workflow 29
Figure 12: Update Workflow 29
Figure 13: Delete Workflow 30
Figure 14: Event Detection Workflow 31
Figure 15: Deployed Panel in UM Client for Defining UM WPS Process Input Parameters 32
Figure 16: UM Client Visualizes a Threshold Notification that is Pulled from Data Store for Notifications 33
Figure 17: UM SOS Together with the Components Developed for Real Time 39
Figure 18: GUI of UM SOS Simulating Wizard 40
Figure 19: UM Client Provides Concurrent Access to Multiple Layers of Live SOS Observations 41
Figure 20: UM Client Provides Chart-view of Live SOS Observations 41
Figure 21: Interconnected Sensor Hubs 42
Figure 22: 52°North Client Application for Accessing SOS and STA 43
Figure 23: Architecture of the 52°North SOS/STA Client Developments 44

Abstract

The Incident Management Information Sharing (IMIS) Internet of Things (IoT) Pilot established the following objectives:

- Apply OGC principles and practices for collaborative development to existing standards and technology to prototype an IoT approach to sensor use for incident management;
- Employ an agile methodology for collaborative development of system designs, specifications, software and hardware components of an IoT-inspired IMIS sensor capability;
- Develop profiles and extensions of existing Sensor Web Enablement (SWE) and other distributed computing standards to provide a basis for future IMIS sensor and observation interoperability; and
- Prototype capabilities documented in engineering reports and demonstrated in a realistic incident management scenario.

Based on the findings gathered during the implementation and work on these objectives, this Engineering Report describes recommendations on profiles for OGC Web services that shall be used to build IMIS systems.

Business Value

The IMIS IoT Pilot aimed to develop, test and demonstrate the use of networked sensor technologies in a real-world scenario developed in collaboration with the Department of Homeland Security and first responder stakeholders. This pilot demonstrated an IoT approach to sensor use for incident management. Prototype capabilities include ad hoc, nearly automatic deployment, discovery and access to sensor information feeds, as well as derivation of actionable information in common formats for use in computer aided dispatch, emergency operations centers and geographic information systems, as well as mobile devices.

Within this Engineering Report, guidance and recommendations on profiles for OGC Web services in IMIS systems are provided. These recommendations shall help to further advance the applicability of OGC Web services in incident management and thus increase interoperability within this domain.

Keywords

OGC ocs, imis iot pilot, sensor web

Testbed-11 Incident Management Information Sharing Profile Recommendations for OGC Web Services Engineering Report

1 Introduction

This Engineering Report (ER) provides findings of the Open Geospatial Consortium (OGC) Incident Management Information Sharing (IMIS) Internet of Things (IoT) Pilot on profile recommendations for OGC standards. During the IMIS IoT Pilot several OGC standards were implemented and applied with the aim to develop, test and demonstrate the use of networked sensor technologies in a real-world scenario.

One important result of these implementation and testing activities was a set of experiences and ideas for improvements for applying the selected OGC standard in emergency management scenarios. This ER documents these findings. For each standard applied within the IMIS IoT Pilot the different implementations and resulting experiences are introduced. From these finding this document derives several recommendations for optimizing future versions of the used OGC standards or defining profiles for increasing interoperability.

1.1 Scope

This OGC® document gives guidelines and recommendations on the development of profiles for OGC standards to support IMIS based on IoT and Sensor Web technology. It summarizes the corresponding findings of the OGC IMIS IoT Pilot.

1.2 Document Contributor Contact Points

All questions regarding this document should be directed to the editor or the following contributors:

Name	Organization
Simon Jirka	52°North Initiative for Geospatial Open Source Software GmbH
Christoph Stasch	52°North Initiative for Geospatial Open Source Software GmbH
Farzad Alamdar	The University of Melbourne

Mike Botts	Botts Innovative Research Inc.
Roger Brackin	Envitia
Chris Clark	Compusult
Flavius Galiber	Northrup Grumman Corporation
Mohsen Kalantari	The University of Melbourne
Steve Liang	SensorUp
Greg Schumann	Exemplar City, Inc.
Josh Lieberman	Tumbling Walls

1.3 Revision History

Date	Release	Editor	Primary Clauses Modified	Description
2015-11-05	0.0.1	Flavius Galiber	All	Document initialized
2015-11-12	0.0.2	Simon Jirka	All	Definition of document structure
2016-02-03	0.0.3	Christoph Stasch Simon Jirka	All	First version integrating contributions from pilot participants
2016-03-09	0.0.4	Simon Jirka	All	Integration of all contributions into a first consolidated version
2016-06-08	0.9	Simon Jirka	All	Version posted on the OGC portal
2016-08-18	1.0	Josh Lieberman	All	Editorial changes and response to DHS comments

1.4 Future Work

This ER is intended to provide recommendations on the development of IMIS profiles of different OGC standards. Thus, the recommendations on future work can be found at the end of each section.

1.5 Foreword

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The OGC shall not be held responsible for identifying any or all such patent rights.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the standard set forth in this document, and to provide supporting documentation.

2 References

The following documents are referenced in this document. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. For undated references, the latest edition of the normative document referred to applies.

OGC 06-121r3, *OGC® Web Services Common Standard*

OGC 06-042, *OGC® Web Map Service (WMS)*

OGC 07-006r1, *OGC® Catalog Services*

OGC 08-094r1, *OGC® SWE Common Data Model*

OGC 09-001, *OGC® SWE Service Model*

OGC 09-025r2, *OGC® Web Feature Service (WFS)*

OGC 10-025r1, *OGC® Observations and Measurements (O&M) - XML Implementation*

OGC 12-000, *OGC® Sensor Model Language (SensorML)*

OGC 12-006, *OGC® Sensor Observation Service (SOS)*

OGC 14-065, *OGC® Web Processing Service (WPS)*

NOTE This OWS Common Standard contains a list of normative references that are also applicable to this Implementation Standard.

In addition to this document, this report includes several Extensible Markup Language (XML) files as specified in Annex A.

3 Terms and Definitions

For the purposes of this report, the definitions specified in Clause 4 of the OWS Common Implementation Standard [OGC 06-121r3] shall apply.

4 Conventions

4.1 Abbreviated Terms

API	Application Program Interface
AVL	Automated Vehicle Location
AWS	Amazon Web Services
CSW	Catalog Service for the Web
EML	Event Pattern Markup Language
ER	Engineering Report
GML	Geography Markup Language
IMIS	Incident Management Information Sharing
IoT	Internet of Things
JSON	Java Script Object Notation
KVP	Key-Value Pair
MQTT	Message Queue Telemetry Transport
O&M	Observation & Measurements
OSH	OpenSensorHub
OWS	OGC Web Services
POX	Plain Old XML
PTZ	Pan–Tilt–Zoom
SAS	Sensor Alert Service
SES	Sensor Event Service
SensorML	Sensor Model Language
SLD	Styled Layer Descriptor
SOS	Sensor Observation Service
STA	Sensor Things API
SWE	Sensor Web Enablement
UAS	Unmanned Aerial Sensor
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UUID	Universally Unique Identifier

WEPS	Web Event Processing Service
WFS	Web Feature Service
WMS	Web Map Service
WPS	Web Processing Service
XML	Extensible Markup Language

4.2 Unified Modeling Language Notation

Most diagrams that appear in this ER are presented using the Unified Modeling Language (UML) static structure diagram, as described in Subclause 5.2 of [OGC 06-121r3].

5 Overview of Existing Standards

This section provides an overview on the existing standards that have been applied in the IMIS IoT Pilot (for an overview on the architecture see the OGC IoT Architecture ER (OGC 16-014)). The overview is divided into two subsections: Section 5.1 gives an overview on the standards for data models and encodings and Section 5.2 introduces the different standards specifying the service interfaces.

5.1 Overview on Data Models and Encoding Standards

The *Geography Markup Language (GML)* Encoding Standard (OGC 07-036) defines a modeling language for geographic information and XML encoding for transferring geographic information between applications. While GML defines the models and encodings for geometries of geographic features such as points, lines and polygons, it does not prescribe the attributes of these features. Therefore, domain-specific application profiles should be defined.

One such profile is the *Observations & Measurements (O&M)* standard, which has been defined within the Sensor Web Enablement (SWE) initiative for exchanging observation data. It consists of two specifications: the conceptual model (OGC 10-004r3/ISO 19156) is based on the general feature model and defines basic properties of observations, e.g., temporal attributes, information about the procedure used to generate the observation result, or the observed property. It also defines a model for sampling features. XML encodings for basic observation types defined in the conceptual model are specified in the O&M XML Implementation Standard (OGC 10-025r1).

Observations encoded in O&M contain a reference to the procedure used to generate the observation result. The description of this procedure is usually provided using the *Sensor Model Language* (SensorML, OGC 12-000). SensorML defines a model and XML encoding for processes associated with the measurement and post-transformation of measured values. These processes may be implemented as sensors, actuators or computational processes. Both, O&M and SensorML, rely on a common model for describing and encoding sensor data (streams), the SWE Common Data Model (SWE Common, OGC 08-094r1).

Finally, for transferring data from the low-level devices to OGC services and vice versa and for sending tasking information to such devices, the Message Queue Telemetry Transport (MQTT) protocol has been used. It has become an OASIS standard in v3.1.1[1]. MQTT defines a lightweight publish/subscribe messaging protocol for Machine to Machine (M2M) communication and is hence in particular used for IoT applications.

[1] http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html

5.2 Overview on Service Interface Standards

The *OGC Web Service Common (OWS Common)* Standard (OGC 06-121r9) specifies aspects that are common to all OGC interface standards. These include the definition of the GetCapabilities operation and the Capabilities response structure as well as the definition of XML and Key-Value Pair (KVP) encodings of operation requests and responses. Each service described below builds upon these common aspects specified in OWS Common.

The *OGC Catalog Service* Implementation Specification (OGC 07-006r1) has been defined to enable clients to publish and/or discover geospatial datasets and services and to provide the metadata needed to decide whether clients could use these datasets and services. The standard specifies interfaces and bindings for publishing and accessing digital catalogs of metadata for geospatial data, services and related resource information. The interface provides operations for managing the metadata records, e.g., for harvesting records, as well as operations for discovering the metadata records, e.g., for describing record types or querying certain records.

For publishing and retrieving maps as images, e.g., for providing background maps or pre-rendered satellite data, the OGC has defined the *OpenGIS Web Map Server (WMS)* Implementation Specification (OGC 06-042). A WMS lists its available map layers in the Capabilities document and allows retrieving these layers with several query parameters, e.g., BoundingBox, using the GetMap operation. The optional GetFeatureInfo operation allows providing additional information for a certain pixel.

The *Web Feature Service* (WFS, OGC 09-025r1/ISO 19142) specifies a service interface for retrieving geographic features (vector data) encoded in GML. The supported feature types are listed in the Capabilities document. A description of a certain feature type can be retrieved using the DescribeFeatureType operation. The central operation is the GetFeature operation that allows querying features from a WFS server. Further optional operation are specified, for example the Transaction operation for inserting, updating or deleting features.

While the WFS specifies a general interface for access to geographic features, the Sensor Observation Service (SOS, OGC 12-006) defines an interface for the pull-based retrieval of sensor observations and sensor descriptions. It thereby utilizes the models and encodings defined by the O&M, SWE Common Data, and SensorML standards (see above). Available observation datasets are described with spatial and temporal extents, generating procedures (usually sensors) and observed properties in the capabilities document of the service. Using the DescribeSensor operation, clients can retrieve relevant metadata about sensors encoded in SensorML.[2] The GetObservation operation is the core operation for retrieving observations using several optional filters for different

[2] The DescribeSensor operation is specified in the SWE Service Model Implementation Standard (OGC 09-001) and referenced from the SOS specification.

observation properties. Several extensions exist for transactional retrieval or result handling in case the same request and response metadata should not be repeated in each request and response message. For example, results from sensors and processes can also be retrieved in a highly efficient data stream, using the GetResultTemplate and GetResult requests. The GetResultTemplate is usually called once by a client to get a SWE Common-based data description of the data structure and encoding for a particular offering. Subsequent GetResult requests return only the values of the observations according to the record structure and encoding described in the GetResultTemplate response. The GetResult request can also support continuous data streaming.

Similar to SOS, the *Sensor Things API (STA)*[3]provides an interface for the retrieval of observation data relying on the Observations and Measurements (O&M) model of sensor information. In contrast to SOS, the STA interface relies fundamentally on Representational State Transfer (REST) principles and specifies Java Script Object Notation (JSON) as encoding for the observations. As such, it is lightweight and eases the development of browser-based client applications for developers who favor REST and JSON approaches.

[3] The STA is not yet an official OGC implementation standard. A draft of the standard has been released for public comments at http://www.opengeospatial.org/standards/requests/134.

6 Catalog Service for the Web

6.1 Implemented Solution (HubCat) for Dynamic Registration and Discovery of Things

6.1.1 Compusult Implementation

The Compusult Catalog Service for the Web (CSW) is an implementation of the HTTP binding defined in OGC's OpenGIS Catalog Services 2.0.2 specification (OGC 07-006r1). As its data store, it uses the OASIS ebXML Registry Information Model (ebRIM 3.0).

As a service-oriented registry, it carefully catalogs each supported OGC service (i.e., WMS, WFS, SOS, WMTS, etc.) using the suggested guidelines set forth in both CSW-ebRIM Registry Service - Part 1: ebRIM profile of CSW (OGC 07-110r4) and CSW-ebRIM Registry Service - Part 2: Basic extension package (OGC 07-144r2). This not only enables the Compusult CSW to store information about the various service types in an adaptable and manageable manner, but also enables it to be interoperable with other 2.0.2 CSW clients.

As part of the publishing process, Compusult's CSW creates an ISO 19119 or ISO 19115 document from each OGC service or document it processes and associates it with the item being published. This helps the registry to support querying records using the ISO core queryables. CSW clients can then choose to return the matching record or its associated ISO document.

Another feature of Compusult's CSW is its ability to return various output formats. Using the outputSchema parameter, 2.0.2 CSW clients can choose between the following metadata formats:

- ebRIM (urn:oasis:names:tc:ebxml-regrep:xsd:rim:3.0);
- 2.0.2 CSW Core (http://www.opengis.net/cat/csw/2..2);
- ISO (http://www.isotc211.org/2005/gmd);
- FGDC (http://www.fgdc.gov);
- MARC21 (http://www.loc.gov/MARC21); and
- DIF (http://gcmd.gsfc.nasa.gov/Aboutus/xml/dif/).

To dynamically register a SensorThings service, Compusult used guidelines similar to those outlined in part 1 and 2 of the ebRIM basic extension package (see Section 7.0).

The sections below detail how each SensorThings service is currently represented in the Compusult CSW registry. The terms used are part of the OASIS ebXML Registry Information Model (ebRIM 3.0).

6.1.1.1 The Service Object

Each SensorThing service is represented by an ebRIM Service object. This Service object is the top level object for each registered SensorThing. It is associated with the Thing objects (see Section 6.1.1.2) that are accessible through the SensorThing service Uniform Resource Locator (URL) (i.e., http://TheSensorThingURL/Things).

Each Service object has the following attributes:

- Name = “http://TheSensorThingURL”
- Description = "OGC Sensor Things"

Its associated ebRIM Slots include:

- Slot [Name = "Service URL", Value = “http://TheSensorThingURL”]

To support spatial searching for the service, we create a spatial slot using the maximum bounding area of all the Thing objects.

- Slot [Name = http://purl.org/dc/terms/spatial, Value = “maximum bounding area of all the Thing objects.”]

Each Service object is then classified as: urn:ogc:serviceType:SensorThing

6.1.1.2 The Thing Object

Each Thing object is represented by an ebRIM ExtrinsicObject. It is associated with the DataStream objects (see Section 6.1.1.3) that are accessible through each Thing URL (i.e., http://TheSensorThingURL/Things(11)/Datastreams).

Each Thing object has the following attributes:

- Name = value of @iot.selfLink
- Description = value of description
- Type = urn:ogc:def:ebRIM-ObjectType:OGC:Dataset

Its associated ebRIM Slots include:

- Slot [Name = "@iot.id", Value = (value of @iot.id)]

- Slot [Name = "@iot.selfLink", Value = (value of @iot.selfLink)]
- Slot [Name = "Locations@iot.navigationLink", Value = (value of Locations@iot.navigationLink)]
- Slot [Name = "Datastreams@iot.navigationLink", Value = (value of Datastreams@iot.navigationLink)]

To support spatial searching for each Thing object, we store its last location in a spatial slot:

Slot [Name = "http://purl.org/dc/terms/spatial", Value = gml:Envelope info]

To get this information, we use the last entry in the Thing's Locations link using the top parameter on the REST URL.

Sample URL: http://TheSensorThingURL/Things(11)/Locations?$top=1

Each Thing object is then classified as:

- urn:ogc:def:ebRIM-ObjectType:OGC:Dataset
- urn:ogc:def:ebRIM-ObjectType:OGC:Dataset:SensorThing

To associate the Thing with its parent Service object, an ebRIM Association object is used with the following attributes:

- associationType = urn:ogc:def:ebRIM-AssociationType:OGC:OperatesOn
- sourceObject = id of Service object.
- targetObject = id of Thing object.

6.1.1.3 The Datastream Object

Each Datastream object is represented by an ebRIM ExtrinsicObject with the following attributes:

- Name = value of unitOfMeasurement -> name
- Description = value of description
- Type = urn:ogc:def:ebRIM-ObjectType:OGC:Dataset:OGC-OM:2_0:OM_Measurement

Its associated ebRIM Slots include:

- Slot [Name = "@iot.id", Value = (value of @iot.id)]
- Slot [Name = "@iot.selfLink", Value = (value of @iot.selfLink)]
- Slot [Name = "Thing@iot.navigationLink", Value = (value of Thing@iot.navigationLink]
- Slot [Name = "Sensor@iot.navigationLink", Value = (value of Sensor@iot.navigationLink]
- Slot [Name = "Observations@iot.navigationLink", Value = (value of Observations@iot.navigationLink]
- Slot [Name = "ObservedProperty@iot.navigationLink", Value = (value of ObservedProperty@iot.navigationLink]

To support the ability to search by date and time, we use the Datastream's Observations link to get the last phenomenonTime entry to create the phenomenonTime slot:

- Slot [Name = "phenomenonDate", Value = $phenomenonTime]

This is accomplished using the orderby and top attributes on the REST URL.

Sample URL:
http://TheServiceThingURL/Things(11)/Datastreams(12)/Observations?$orderby=phenomenonTime desc&$top=1

Each Datastream object is then classified as:

- urn:ogc:def:ebRIM-ObjectType:OGC:Dataset
- urn:ogc:def:ebRIM-ObjectType:OGC:Dataset:OGC-OM:2_0:OM_Measurement

To associate each Datastream object with its parent Thing object, an ebRIM Association object is used with the following attributes:

- associationType = "urn:ogc:def:ST-AssociationType:OGC:HasDataStream"
- sourceObject = id of Thing object.
- targetObject = id of Datastream object.

6.2 Pros/Cons

6.2.1 Compusult

6.2.1.1 Pros

- Flexibility: One of the real benefits of using Compusult's CSW is that it is flexible enough to store just about any type of information. This feature allowed us to consume SensorThing services relatively easily.

6.2.1.2 Cons

- No Profile: The specification is flexible and therefore, it allows us to name and associate objects however we want. Without an official profile to follow, continuing with this approach would leave us unable to be semantically interoperable.

- Complicated Queries: Another downfall is that sometimes the data that need to be represented in the ebRIM information model can have a multiple layers of association. For example, the SensorThings service is associated with multiple Thing objects and each Thing object is associated with multiple Datastream objects. Unfortunately, this can sometimes lead to fairly lengthy and complicated CSW queries that are hard to implement.

6.2.2 Envitia

6.2.2.1 Comments on Using the Compusult Registry (Registry Client)

Envitia were, to the knowledge of the authors, the only client provider to directly access the HubCat. The following are comments from the perspective of a provider of CSW-ebRIM clients and servers provider as well as a developer of registry information models and extension packages.

The Compusult CSW-ebRIM implementation exhibits a high degree of compliance with the standard; this is not always true for OGC standards and therefore should be applauded. In that respect, there were no particular interoperability issues and the standard seems sufficiently tight that the Envitia client was able to interact with the Compusult HubCat with little difficulty. Interoperability issues do arise through the choice of HubCat configuration, or ebRIM Registry Extension Package (eREP), since these packages define specializations of the general record types that a generic client may not deal with efficiently.

6.3 Recommended Changes

6.3.1 CSW SensorThings API Profile

Although ebRIM Slots, Classifications and Associations were selected with names and IDs that seemed appropriate, the most important recommendation at this point would be to implement a CSW STA profile so that official guidelines can be followed to ensure proper interoperability with other 2.0.2 CSW clients.

6.3.2 Service Object Model

The current HubCaT implementation focuses to a large degree on cataloging service instances, with service being the primary record type. This is contrary to the approach used in general by the OGC in cataloguing other data. The core OGC model and the model implemented in ISO 19115/19119 as well as in ISO 19139 treat dataset and service as two separate but linked artifacts. This has been embodied in the 115 extension package for the CSW-ebRIM standard that declares datasets and then associate services with it. This would allow a client to present a user with available data and then allow them to discover relevant services that could deliver it. This latter negotiation could go on automatically in the client. With the current model, the Envitia client could list all services available but would represent STA, SOS, and WMS services visualizing an SOS or SensorThings endpoint as separate artifacts even if they serve the same data.

A model closer to that used in I15 would be helpful in resolving this. This is represented in Figure 1 below.

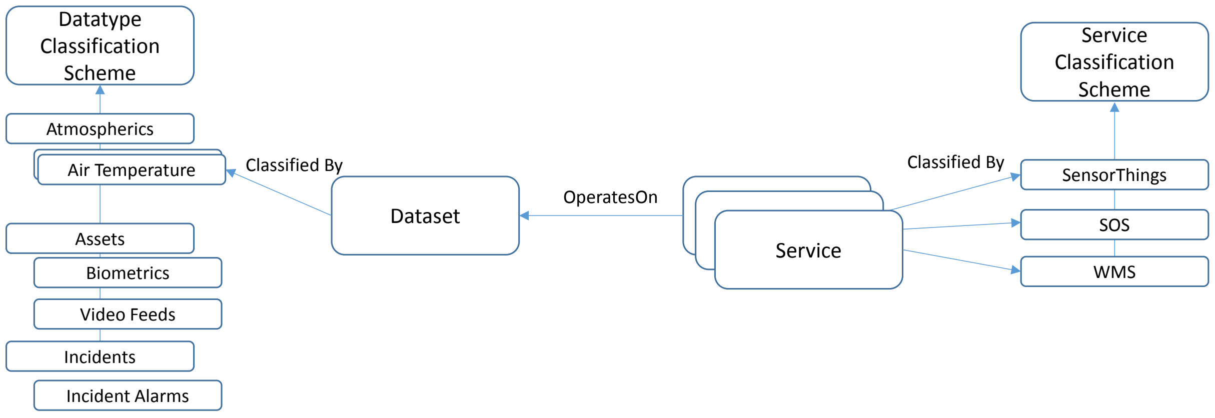


Figure 1: Possible Model

6.3.3 Catalog WMS Service

Using ebRIM classification schemes to classify objects would also be helpful, but as pointed out by Compusult there is a need to standardize on the taxonomies. To some degree, the above model was played out in the Compusult CSW-WMS which implemented form of the classification scheme shown on the left, but in a somewhat limited way, for example linking all sorts of temperature together. This made it useful for discovery but less useful for visualization as it mixed concepts.

In general terms, the availability of the CSW-WMS did allow the discovery of classes of sensors in the Envitia Horizon Geo-portal and also the transition from this to accessing SOS services, which could be accomplished through GetCapabilities requests to the CSW-WMS. But the recommendation is still that a more formal route within the registry would be valuable.

6.3.3.1 Sensor Harvesting WPS

The authors see real value in the Sensor Registration Processing Service (WRPS) provided by Compusult. Evitia provides a very similar interface in practice, although it relies on a separate REST invocation rather than the WPS interface. There is also value in formalizing the rules for mapping metadata from specific sources such as SOS capabilities documents into eREP elements so that the mappings can be implemented the same way in different technologies.

6.3.3.2 OWS Context Document Alignment

The model described here not only maps to CSW-ISO and the CSW-ebRIM I15 profile, but also to OWS Context document which allows for a given (‘Layer’ or ‘Resource’ as it is called) to be offered in various forms. Therefore, an OWS Document could define a ‘Layer’ of ‘Body Temperature’ and offer an SOS and a WMS endpoint to clients so they can access the most appropriate form of this content.

6.3.3.3 Overall Recommendation with Regards to Data Modelling

Overall, the recommendation is that significantly more work is needed to formalize both the SWE-IoT eREP’s and the mapping rules from S-Hub service metadata into the HubCat in order to ensure consistent sensor discovery and exploitation. Envitia consider it to be worth developing a standard profile specification for this. It would have been impossible to effectively perform such work in the first IMIS IoT Pilot, but it would be valuable to carry out in the near future.

6.3.3.4 Stored Queries in CSW-ebXML

If such a standard profile is developed, there may be real value in developing CSW-ebRIM stored queries. These would allow specific questions (such as find human-deployed temperature sensors) that might require fairly complex queries to be executed by relatively simple clients. The value of this feature of CSW-ebRIM in making clients easier to implement is underestimated by many. Envitia has used it extensively in the past and suggest it would fit here well too.

6.3.3.5 Catalog/Registry for Sensor Parameter Classification

An issue in using the various sensors was the lack of metadata to allow them to be discovered without a-priori knowledge. Sensors were in many cases characterized by ‘Sensor_1’ and ‘Parameter_1’ rather than anything identifiable. If there was a sensor issue in providing this, the Catalog would provide a route for administrators to register dictionaries to translate ‘Parameter_1’ to ‘Temperature’ and also add critical metadata such as ‘Deg F or Deg C’ as this is obviously critical.

6.3.3.6 Catalog Federation

Consideration should be given to demonstrating a HubCat federation. Envitia’s client itself actually made use of two HubCat’s for the Pilot demonstration. It accessed the Compusult HubCaT as well as Envitia’s cloud-deployed HubCat2 which had maps and implemented an extension package to store OWS Context documents against communities of interest.

The authors would have preferred to access a single federated registry which issued queries to the HubCat and to the authoritative data. Envitia’s CSW-ebRIM service is capable of supporting this (and most likely that of Compusult, too), and in most urban incident situations there will be more than one HubCat. Envitia’s view is that it might be best to deploy a separate federating catalog service that would not have its own holdings, but simply federate out queries to the HubCat’s serving a particular incident and aggregate the results for the client. Such models are common and efficient if a high degree of interoperability between HubCat’s can be maintained.

6.3.3.7 Dynamic Registration of Sensors

The practice adopted for the IMIS IoT Pilot was to register sensors when they came online and drop them out when they went offline. This caused problems in client implementation; it was hard to obtain a view of the potentially versus actually available sensors. In some cases, sensor placements were ad hoc but in others the sensors were predictably positioned, for example on fire trucks. Two approaches may address this. The first is to catalog every potential sensor, but provide an ‘online/offline’ flag. The other is to model ‘Sensor Class’ so that a sensor’s interface is clear when it comes online (i.e., Is it going to be sensor things or SOS? If it is SOS what profile will it support?).

7 Web Feature Service (WFS)

A WFS was deployed for the Pilot that provided access to the features of interest (FoI's) of observations that had been published through SOS.

7.1 Implemented Solution

7.1.1 52°North

An overview on the components of the implemented solution for WFS is given in Figure 2. 52°North has implemented a WFS based on its Web Service framework Iceland[4] that acts as a proxy to a SOS. The component offers the mandatory operations of a WFS, i.e., the GetCapabilities operation, the DescribeFeatureType and the GetFeature operation. Besides general operations metadata, the Capabilities document lists the FeatureTypes that are served by the WFS (the document is listed in Annex A.2). As the WFS is serving both the observations as well as the sampling features, the types OM_Observation and SF_SamplingFeature are listed in the Capabilities and the corresponding XML schemas can be retrieved using the DescribeFeatureType operation.

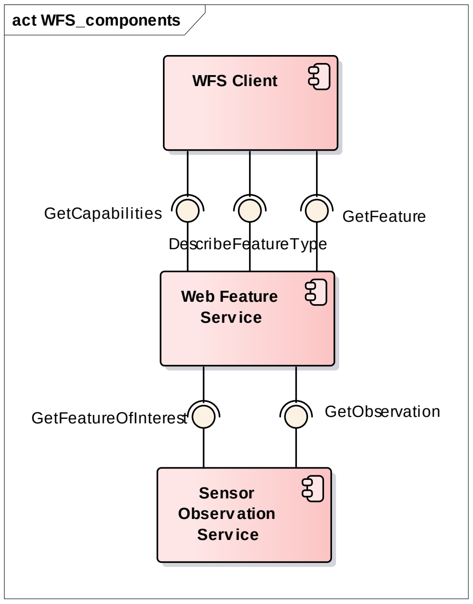


Figure 2: UML Component Diagram the Implementation of the WFS

A sequence diagram of interactions between WFS and SOS is given in Figure 3. First, a client can query the Capabilities and available feature types from the WFS using the GetCapabilities and DescribeFeatureType operations. As core functionality, the client can retrieve the observations and features of interest from the WFS using the GetFeature

[4] More information the 52°North Iceland Framework is available at https://wiki.52north.org/bin/view/SensorWeb/Iceland

operation. As the WFS implementation acts as a proxy for SOS servers, it needs to map the GetFeature requests to GetObservation and/or GetFeatureOfInterest requests and can then forward these requests to the SOS. Once the WFS has received the features, it can then forward them to the client. In case of observations that are requested, the observation features need to be extracted from the GetObservationResponse and put into a FeatureCollection.

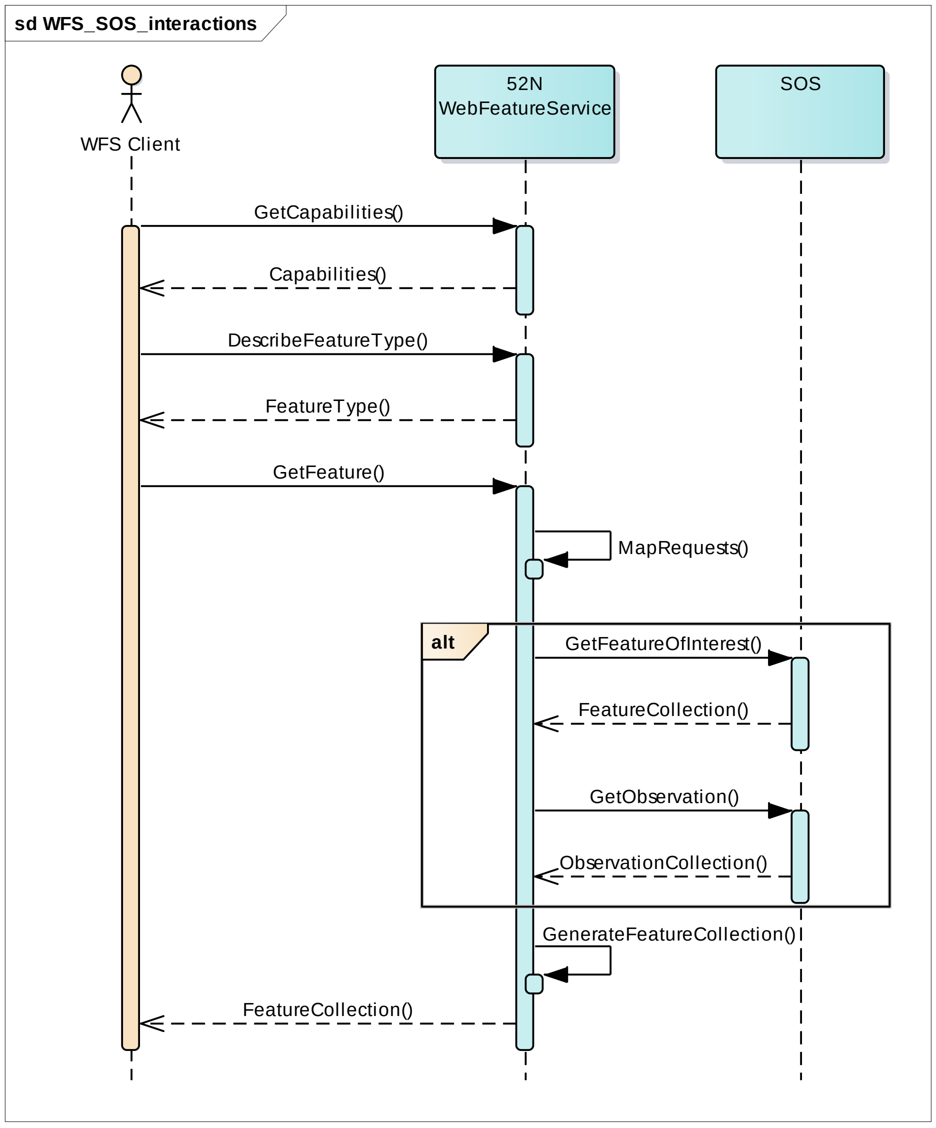


Figure 3: Sequence Diagram of the Interactions Between WFS and SOS

7.2 Pros/Cons

7.2.1 52°North

Serving observations and features of interest through a WFS server allows WFS clients to retrieve this information without having to support SOS servers. The WFS lacks metadata about the observations and sensors available in the Capabilities document, however. For example, it is not possible to obtain information for which time period, observed properties and from which sensor observations are available. Furthermore, the WFS also lacks pre-defined filters for temporal attributes as well as other observation properties, e.g., for the observed property that points to a description of the observed phenomenon or for the procedure that points to a description of how the result of an observation has been taken (usually a sensor description).

Due to these gaps, the SOS may be seen as a specialization of WFS. The SOS supports one basic feature schema for observations (the O&M model) and provides dedicated operations for retrieving sensor metadata (DescribeSensor), features of interest (GetFeatureOfInterest), observations (GetObservation). For each operation, pre-defined filters are available. For example, the Request schema for GetObservation defines filters for procedures, observed properties, samplingTime and resultTime.

7.3 Recommended Changes

No specific changes to the specification are recommended. The Pilot implementation experience suggests, however, that there is some value in devoting SOS to providing sensor metadata and observations, and otherwise using WFS to provide information about spatial features such as the features of interest linked to the SOS observations. This practice is illustrated in Figure 4. The SOS provides, in essence, dynamic property values for the features served by the WFS.

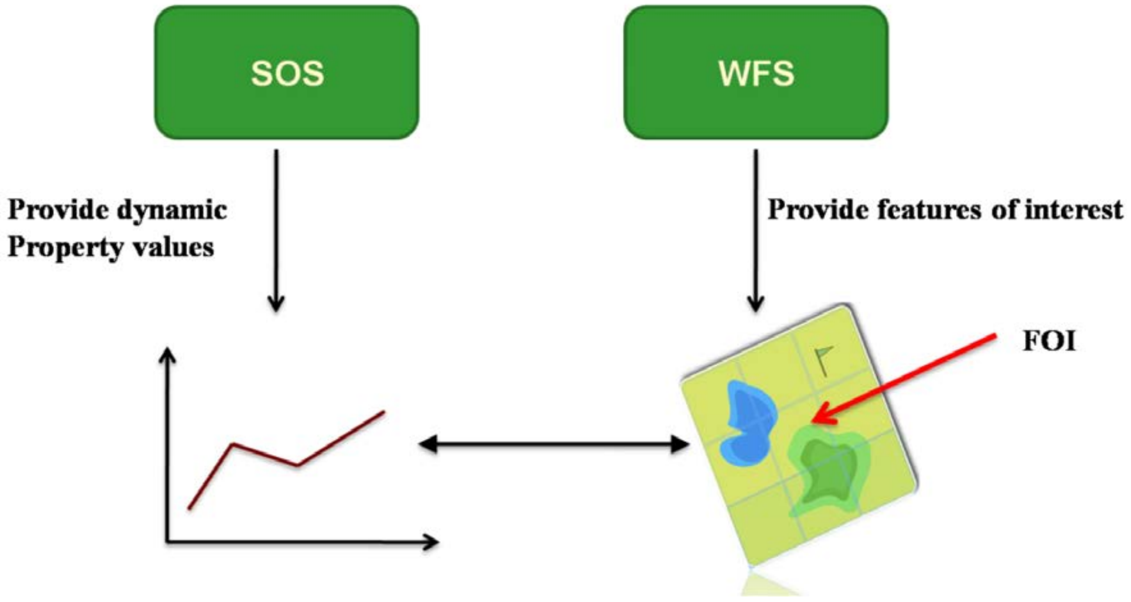


Figure 4: Coupling of SOS and WFS (Source: OGC 12-006; p. 147)

8 Web Maps Service (WMS)

WMS have been developed to provide applicable basemap data for the incident response area along with IoT features as a layer. The WMS functions both as an integral map server and as a Feature Portrayal Server (FPS) to render map images from remote WFS feature collections and SOS observations.

8.1 Implemented Solution

8.1.1 52°North

52°North has implemented a WMS that can harvest features of interest from SOS servers and visualize information about the features and related observations in map layers. Figure 5 provides an overview of the components.

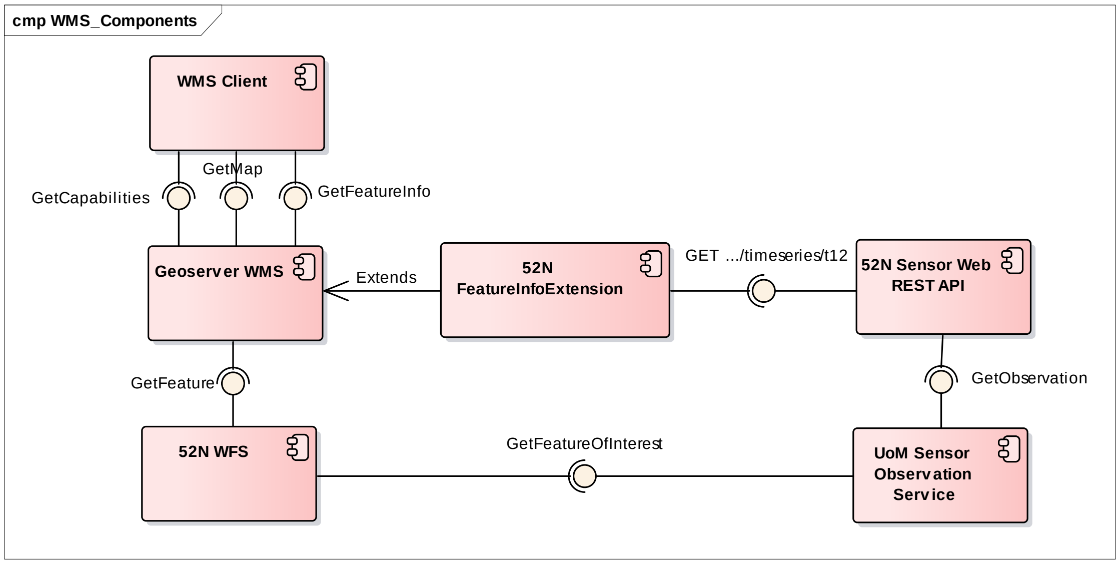


Figure 5: UML Component Diagram of 52°North's WMS Implementation

The GeoServer WMS is used to provide the basic WMS operations GetCapabilities, GetMap and GetFeatureInfo. The GeoServer software can be configured to serve feature layers from a WFS server as map layers in a WMS. Hence, the 52°North WFS described in Section 7.1 is utilized to provide the base data used for rendering the map layer displaying the features of interest. The GeoServer WMS then renders a map layer for these features of interest using pre-defined symbols. Without supporting information about the sensors that are observing the features and observations about the features, however, the WMS layer is of limited use.

52°North thus implemented the FeatureInfo extension. GeoServer utilizes Apache FreeMarker[5], a Java-based template engine, to generate HyperText Markup Language (HTML) templates. 52°North's FeatureInfo extension configures these templates by injecting into them URLs that link to the 52°North's Sensor Web REST API. The API encapsulates the business logic for accessing SOS servers as a client, provides RESTful access to observations and sensor descriptions, and returns those observations encoded in JSON that can easily be integrated into Web sites. Figure 6 shows a sequence diagram of typical interactions between clients, the WMS implementation and the components utilized for the implementation. Once a GetFeatureInfo request is sent to the GeoServer WMS, the WMS searches for a FeatureInfo template using the ID of the feature for which information has been requested. The FeatureInfo extension generates the HTML template by injecting relevant URLs to resources such as observations, sensor description, etc., served by the Sensor Web REST API. The prepared HTML template is then used by the GeoServer WMS and returned to the client. Once the HTML template is loaded on the client side, the URLs are resolved and the information is displayed in the FeatureInfo HTML representation.

[5] http://freemarker.org/

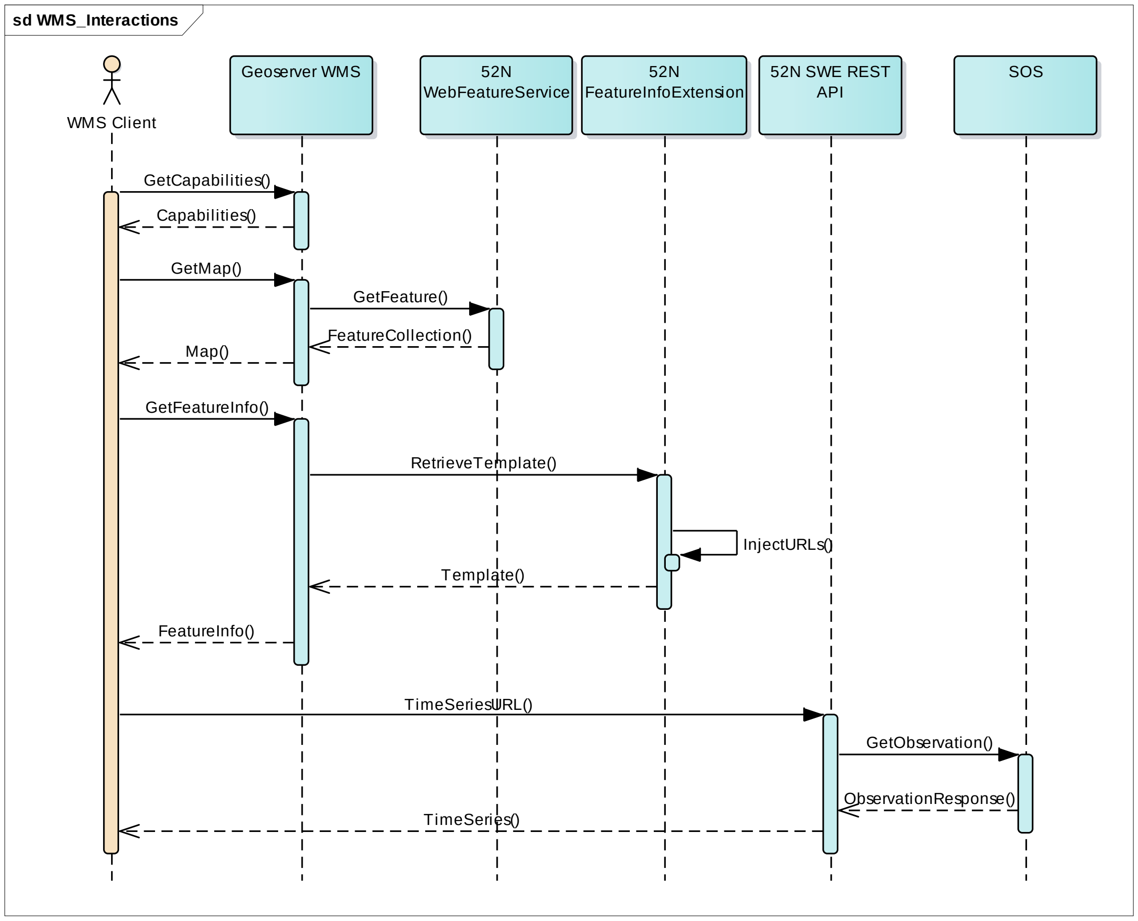
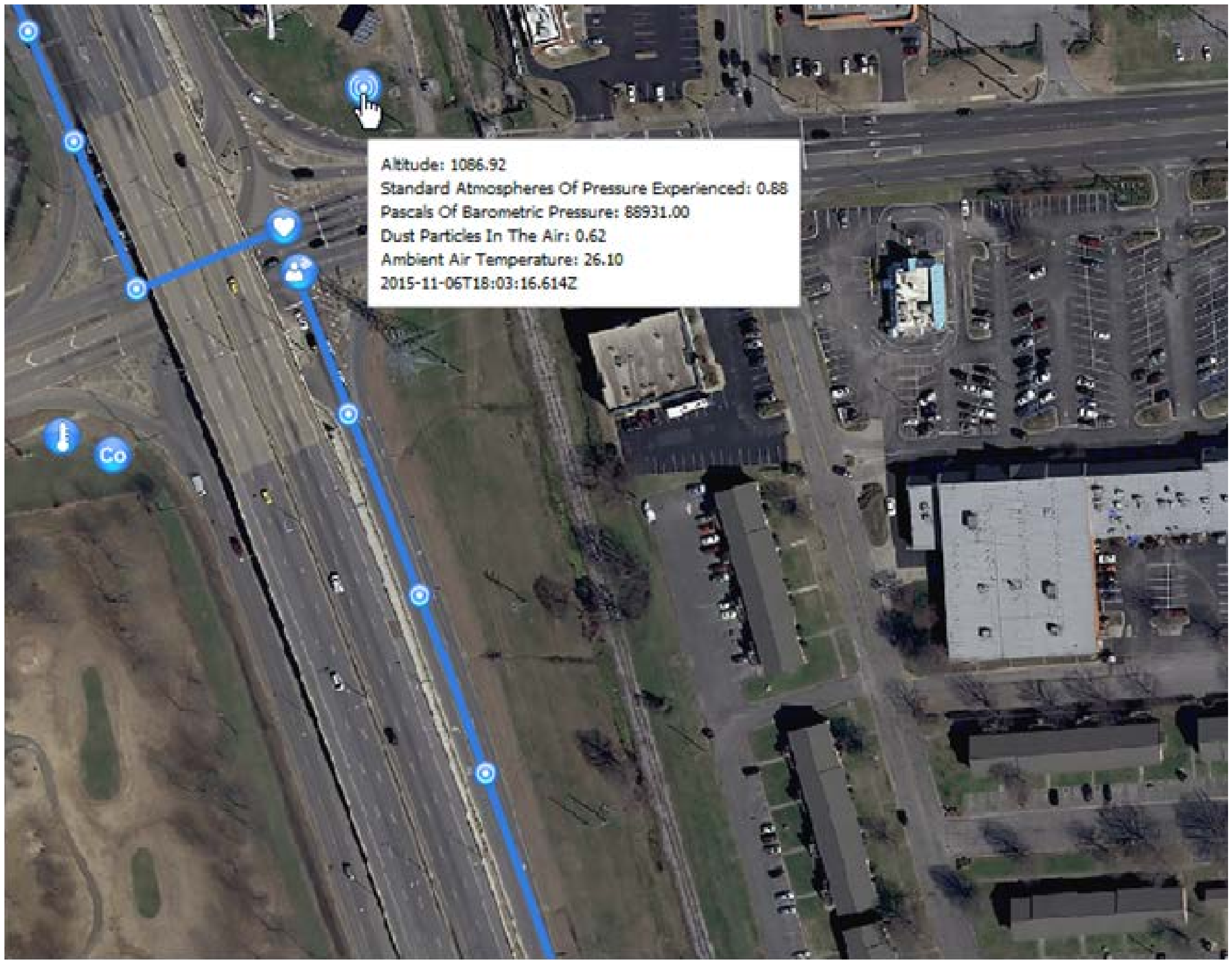


Figure 6: Sequence Diagram of Interactions Between Clients, WMS, WFS and SOS

8.1.2 Compusult

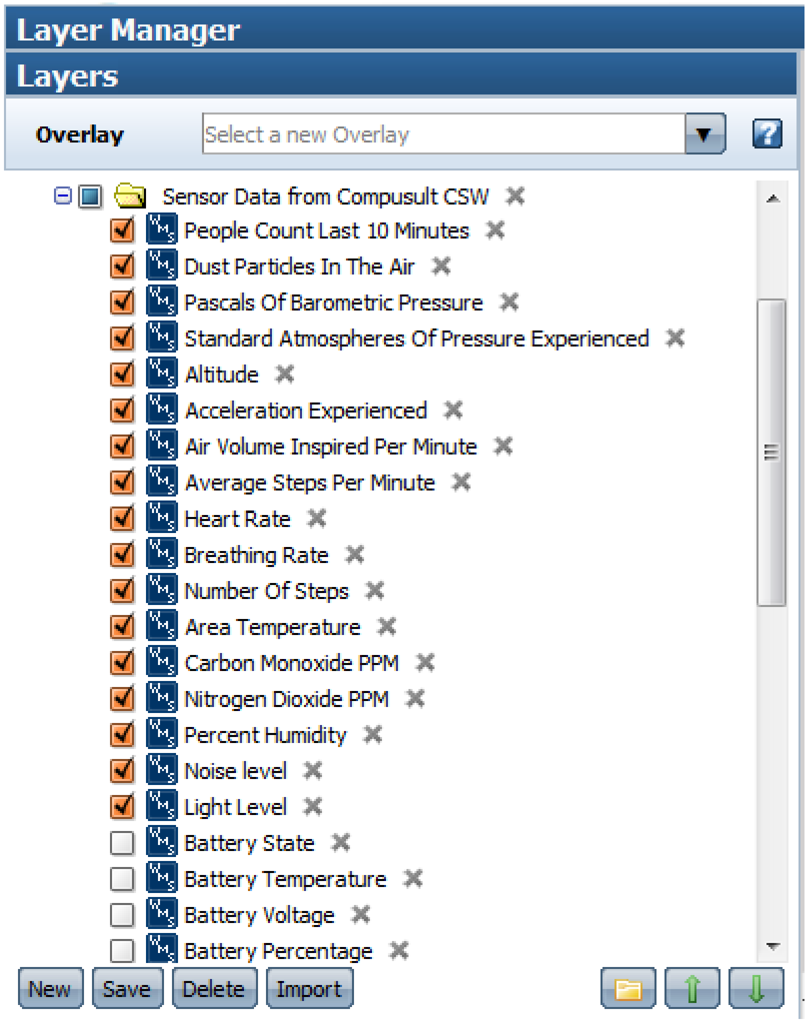
The Compusult WMS was implemented to provide visualization of all the sensor data available in the Compusult HubCat. Sensor data is currently available from both SOS and STA services, however the WMS consolidates the data such that a user does not need to know what type of service the data came through, only the type of the sensor the data comes from. The data are then grouped by type into separate map layers. If more than one layer is active and a device or Thing has sensors in multiple layers, the symbol is changed to represent a device instead of a sensor type. The user can hover over a sensor symbol to see the current values. This is visualized in Figure 7.



Altitude: 1086.92
Standard Atmospheres Of Pressure Experienced: 0.88
Pascals Of Barometric Pressure: 88931.00
Dust Particles In The Air: 0.62
Ambient Air Temperature: 26.10
2015-11-06T18:03:16.614Z

Figure 7: Compusult WMS Output

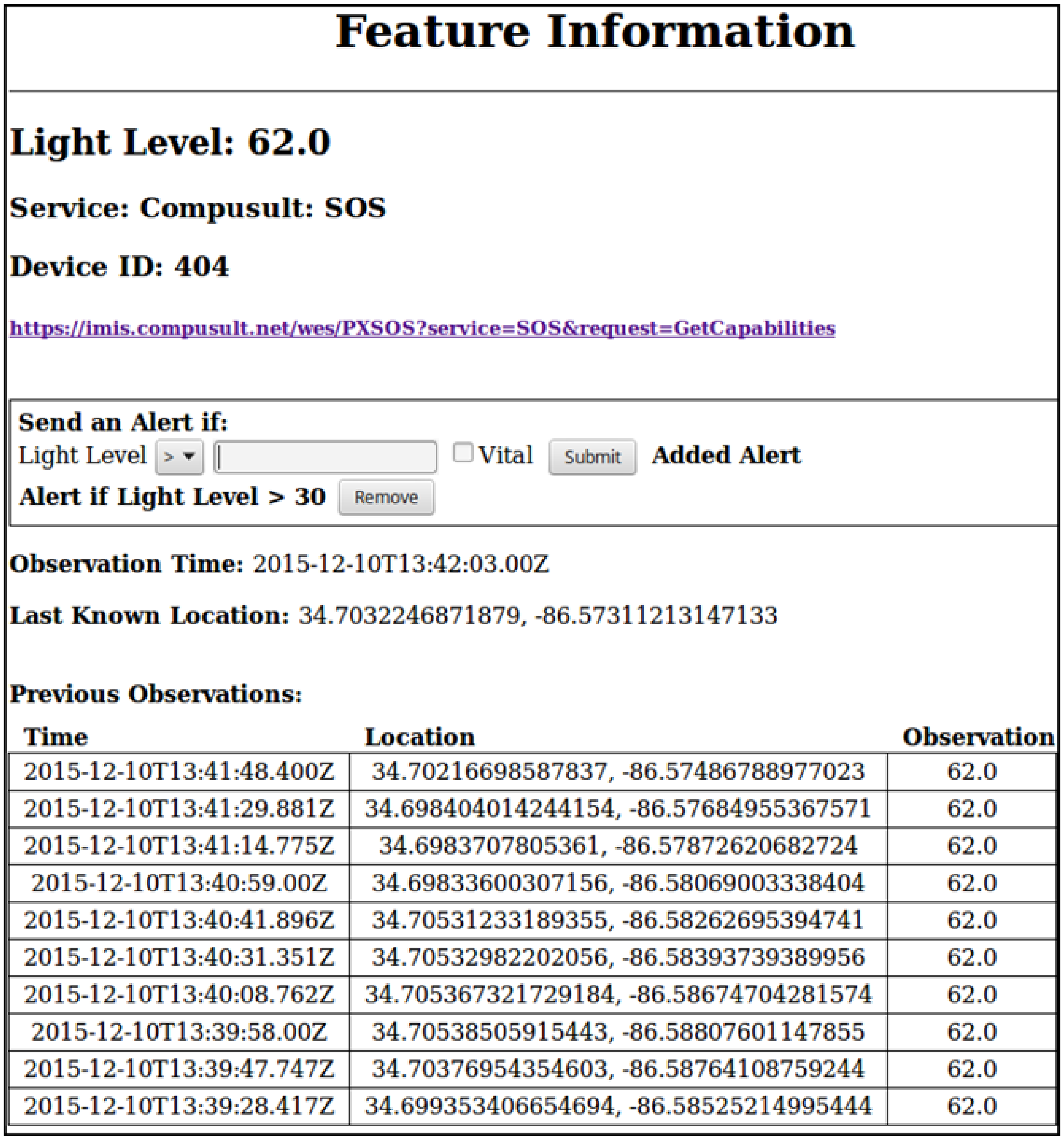
The layers in the GetCapabilities request are organized by sensor data types, allowing a user to see all data of the same type within a single layer. A configurable mapping of data types was used to perform semantic mediation on the data because, typically, services use different names for the same data. A user can simply add layers for the data he or she is interested in, as illustrated in Figure 8 below.



Sensor Data from Compusult CSW
- [✓] People Count Last 10 Minutes
- [✓] Dust Particles In The Air
- [✓] Pascals Of Barometric Pressure
- [✓] Standard Atmospheres Of Pressure Experienced
- [✓] Altitude
- [✓] Acceleration Experienced
- [✓] Air Volume Inspired Per Minute
- [✓] Average Steps Per Minute
- [✓] Heart Rate
- [✓] Breathing Rate
- [✓] Number Of Steps
- [✓] Area Temperature
- [✓] Carbon Monoxide PPM
- [✓] Nitrogen Dioxide PPM
- [✓] Percent Humidity
- [✓] Noise level
- [✓] Light Level
- [ ] Battery State
- [ ] Battery Temperature
- [ ] Battery Voltage
- [ ] Battery Percentage

Figure 8: Layer Manager

Data is retrieved and cached from external services using a system of multiple threads to ensure that slow or problematic services do not slow down the WMS. The WMS stores a configurable amount of historic data for each service as well. Historic locations for a device are displayed as dots with a path connecting them to the current reading. The GetFeatureInfo operation is also available from the Compusult WMS. The feature information page shown in Figure 9 visualizes the recent observations with the associated locations and timestamps. It also shows the data type of the observations and the service the observations were retrieved from.



Feature Information

Light Level: 62.0

Service: Compusult: SOS

Device ID: 404

https://imis.compusult.net/wes/PXSOS?service=SOS&request=GetCapabilities

Send an Alert if:

Alert if Light Level > 30

Observation Time: 2015-12-10T13:42:03.00Z

Last Known Location: 34.7032246871879, -86.57311213147133

Previous Observations:

Time	Location	Observation
2015-12-10T13:41:48.400Z	34.70216698587837, -86.57486788977023	62.0
2015-12-10T13:41:29.881Z	34.698404014244154, -86.57684955367571	62.0
2015-12-10T13:41:14.775Z	34.6983707805361, -86.57872620682724	62.0
2015-12-10T13:40:59.00Z	34.69833600307156, -86.58069003338404	62.0
2015-12-10T13:40:41.896Z	34.70531233189355, -86.58262695394741	62.0
2015-12-10T13:40:31.351Z	34.70532982202056, -86.58393739389956	62.0
2015-12-10T13:40:08.762Z	34.705367321729184, -86.58674704281574	62.0
2015-12-10T13:39:58.00Z	34.70538505915443, -86.58807601147855	62.0
2015-12-10T13:39:47.747Z	34.70376954354603, -86.58764108759244	62.0
2015-12-10T13:39:28.417Z	34.699353406654694, -86.58525214995444	62.0

Figure 9: Feature Information Page

The feature information page also allows the user to create simple alerts on the incoming observations. When a new observation is retrieved, it is checked against any user-defined alerts and notifications are fired when matches occur. These notifications will appear as a dialog that must be dismissed if the alert is set to vital, or a short-lived toast message if not.

8.2 Pros/Cons

8.2.1 52°North

The solution based on GeoServer was largely implemented using default configuration properties of the GeoServer WMS implementation. For this reason, the feature layers of the WFS implementation described in Section 7.1 were used as input for the WMS map layers. Although this reduces implementation efforts, it also comes with a communication overhead in the implementation, as the WMS queries a WFS that in turn queries an SOS. In case of a large number of features, a better solution may directly use the GetFeatureOfInterest operation of SOS. The current solution offers the advantage that features of interest can be served by WFS, whereas the observations for dynamic properties are provided by SOS servers (see Section 7.3).

8.2.2 Compusult

8.2.2.1 Pros

- Consolidated Data: Data from different services is consolidated into a single view. A user can simply pick the layer for the data they are interested in.

8.2.2.2 Cons

- Data Type Naming: Different services use different names for each type of data. With no semantic rules for naming data types, users must implement specific mappings for each service they are using.

- Capabilities Updates: If another service requests all of the available layers from a Capabilities document, it will be unaware of new layers added to the service until it requests the Capabilities document again.

8.3 Recommended Changes

To easily embed information provided by SOS servers into thin clients or, as done for the WMS GetFeatureInfo implementation, in HTML templates, a REST binding for SOS together with a Response Encoding in JSON is recommended. The discussion paper on O&M JSON Encoding (OGC 15-100r1) may serve as a good basis for defining the JSON response encoding.

The GetFeatureInfo operation is well suited to provide common information about the feature as well as observations for this feature. As the feature info can be provided in HTML, the observation information can be encoded in flexible ways, e.g., as tables or as images showing diagrams. In the event that this becomes common practice, however, it would be beneficial to agree upon a minimal set of information that should be provided in the feature info and a common structure for providing this info.

9 Web Processing Service (WPS)

9.1 Implemented Solution

9.1.1 52°North

Within the IMIS IoT Pilot, 52°North developed an event processing architecture which relies on a WPS server for event processing. Central element is the Web Processing Service for Event Processing (Web Event Processing Service, WEPS) which controls the overall workflow. Main tasks of this WEPS are:

- Handling and managing client event subscriptions through WPS Execute requests; and
- Controlling the event processing module which performs the analysis and pattern matching of incoming sensor data streams against the event pattern rules contained in the event subscriptions.

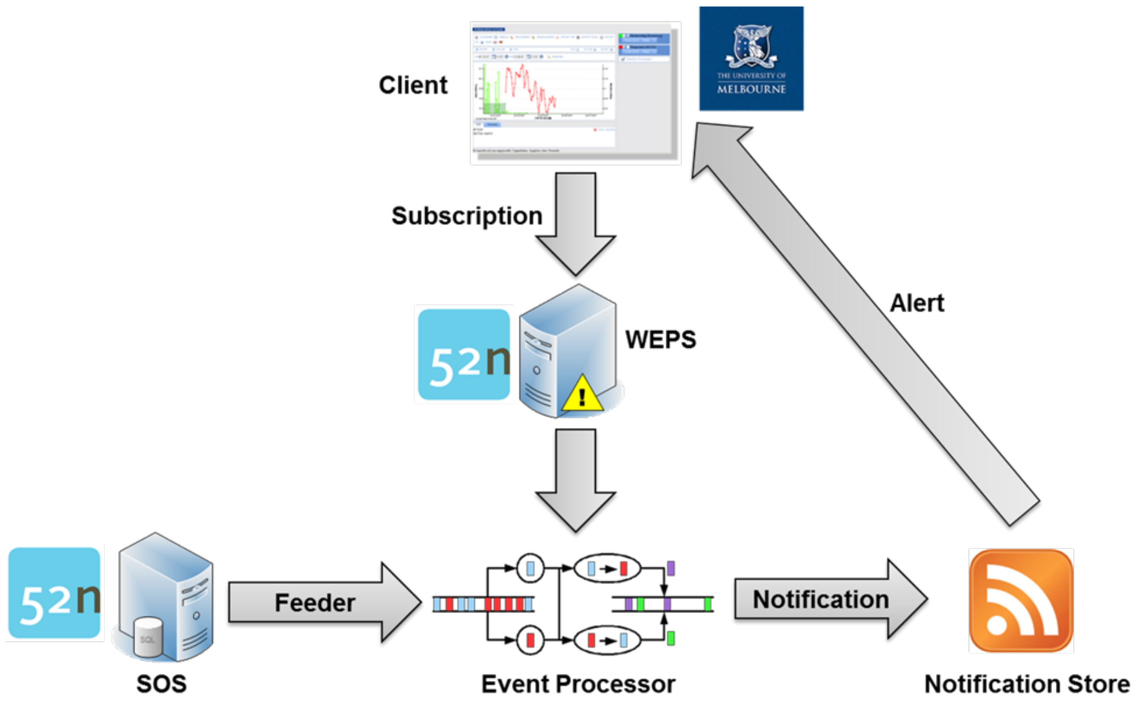


Figure 10: Event Processing Architecture Based on the WEPS

Figure 10 shows the developed architecture with the WEPS at its core. After receiving a subscription, the WEPS tasks the Event Processor with the corresponding rules for detecting events relevant to this subscription. After this, the WEPS initiates a feeding process that regularly checks a data source (in this case a SOS server) for new observations. As soon as a new observation is available, it is pushed into the event processor.

The output of the event processor (i.e., all detected events that match to a subscription) are sent to the Notification Store. This is an RSS-based component that allows clients to consume RSS-feeds containing those notifications that correspond to their subscriptions.

For initiating an event processing task at the WEPS, the Execute operation is used. This request contains the following elements:

- **Rule**: The event filtering rule encoded as specified in the OGC Event Pattern Markup Language (EML) Discussion Paper (OGC 08-132), this rule specifies which events are of interest to the user so that a notification message shall be dispatched if they occur.

- **Sampling Rate**: A value indicating how often new observations are published by the sensor, the sampling rate is used by a feeder to determine how often the data source shall be queried for new observations.

- **Runtime**: The duration that the subscription shall be active.

- **SOS Endpoint**: The URL of the SOS server that shall be used by a feeder to retrieve new observations which are relevant for the subscription.

- **GetObservation Template (KVP)**: A KVP encoded GetObservation request that delivers the observations required for processing the subscription, the feeder automatically adds a temporal filter to this URL. This temporal filter is dynamically generated based on the time stamp of the last observation that was pushed into the event processor.

- **GetObservation Template (POX)**: This includes values for the GetObservation request parameters procedure, observedProperty, featureOfInterest and responseFormat (equivalent to the corresponding parameters in the KVP GetObervation Template).

Furthermore, information about the target to which notifications shall be sent must be included in a WEPS Execute request.

9.1.2 Compusult

The Compusult Web Registration Processing Service (WRPS) was implemented to provide a simpler way for S-Hubs to register their WMS, WMTS, SOS and STA services. Typically, to register a service in a HubCat requires invoking an Insert Transaction that maps input metadata into the schema of the information model supported by the HubCat. The mapping operation can be complex and convoluted. The WRPS simplifies this process because it provides only three operations, taking just a single parameter each. These operations consist of:

<u>Insert</u>

To publish a service, a WPS request which provides the service URL as an input parameter is submitted. The WPS will pass this value to the Compusult Publishing Module, which gathers the required information from the service, creates the required metadata document and performs an insert into the Catalog. A Universally Unique Identifier (UUID) which can be used to perform updates and deletes to this specific record at a later date is returned. This process is illustrated in Figure 11 below:

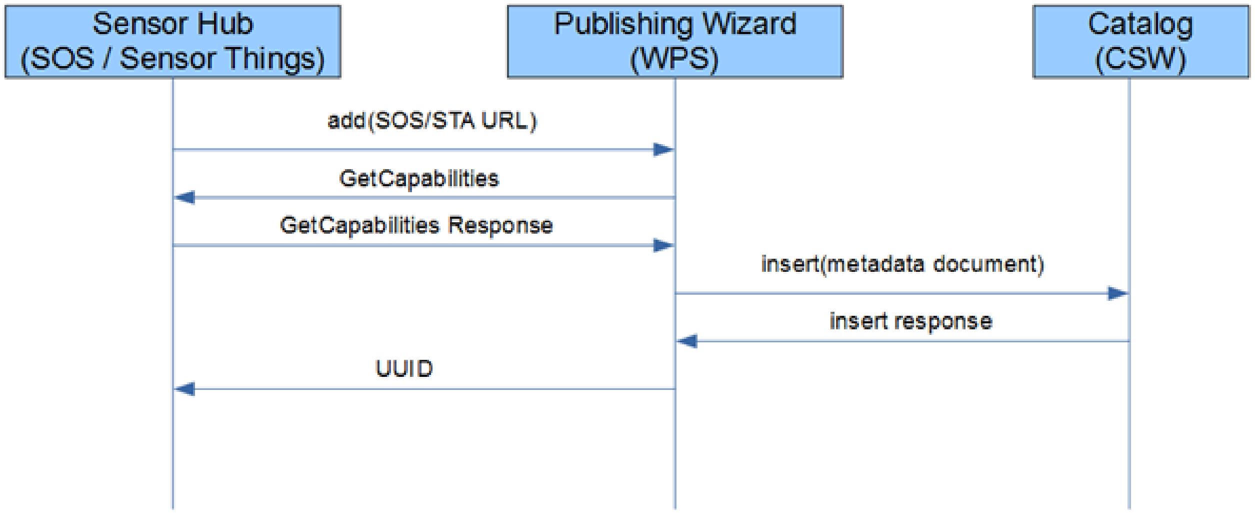


Figure 11: Insert Workflow

<u>Update</u>

To update a service, a WPS request which provides the UUID of the record to be updated is submitted. The process that follows is similar to that of insert and is illustrated in Figure 12 below:

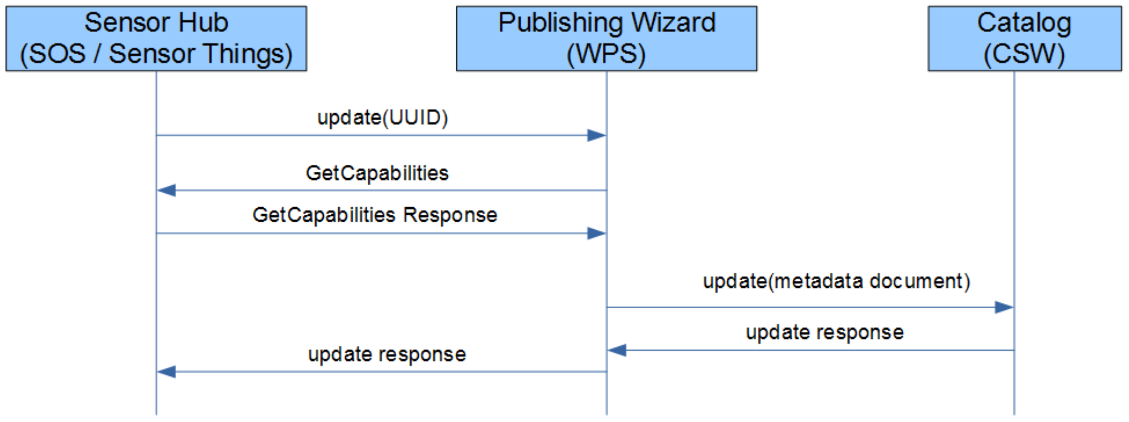


Figure 12: Update Workflow

Delete

To delete a service, a WPS request which provides the UUID of the record to be deleted is submitted. The WPS will send a Delete transaction directly to the Catalog and the record is removed, as shown in Figure 13.

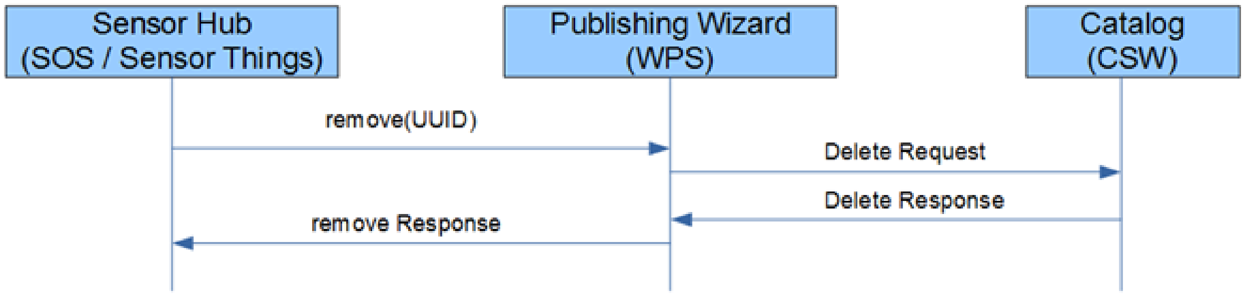


Figure 13: Delete Workflow

9.1.3 University of Melbourne

The University of Melbourne (UM) set up another WPS to enable threshold-based real-time event detection on sensor observations. Although WPS has been widely used for manipulation of static geospatial data, it has been used less often for processing live sensor data. The IMIS IoT Pilot testbed provided an opportunity to try this. GeoServer, an open source, Java-based Web server for editing and sharing geospatial data was selected as the WPS. GeoServer implements a large set of OGC standard services such as WFS, WMS, Web Coverage Service (WCS) and WPS.

9.1.3.1 Event Detection Workflow

Figure 14 shows the workflow of the UM WPS, illustrating the included components and their interactions. These components encompass UM Client, GeoServer WPS Server, UM

Data Store for Notifications, UM WPS Process and SOS Server. The UM client, a GIS Web application, creates and issues WPS requests to the GeoServer WPS, and visualizes the returned processing results. The WPS accesses live observations from the SOS server and processes them to detect threshold-exceeding events. The UM Datastore for Notifications publishes RSS feeds for the alert notifications resulting from the WPS event detections.

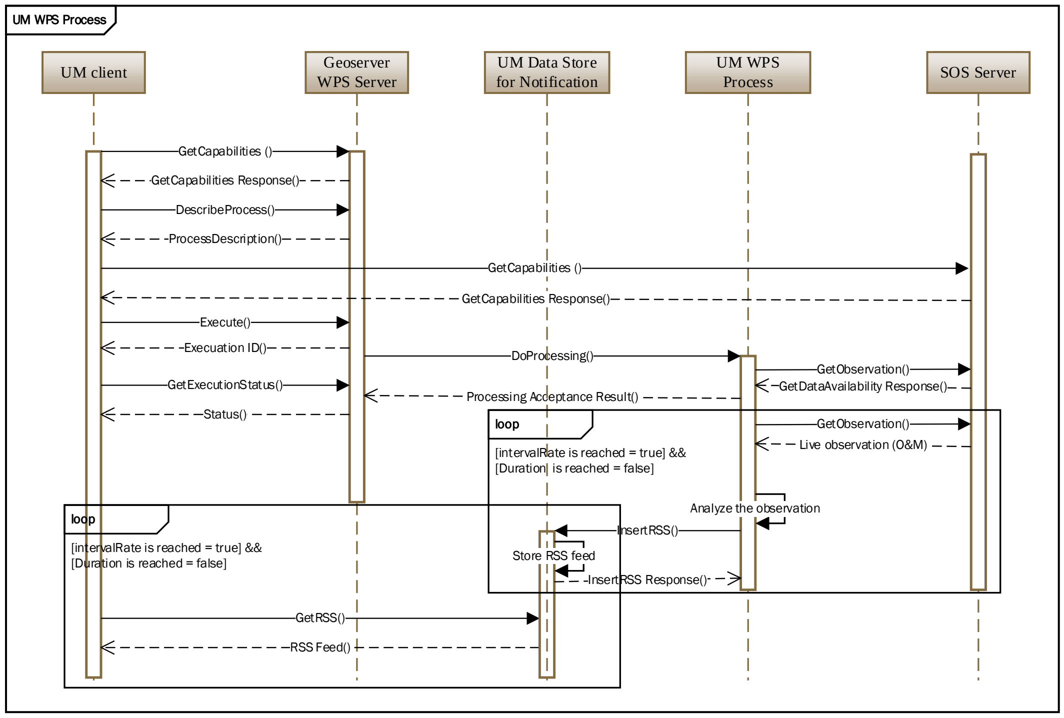
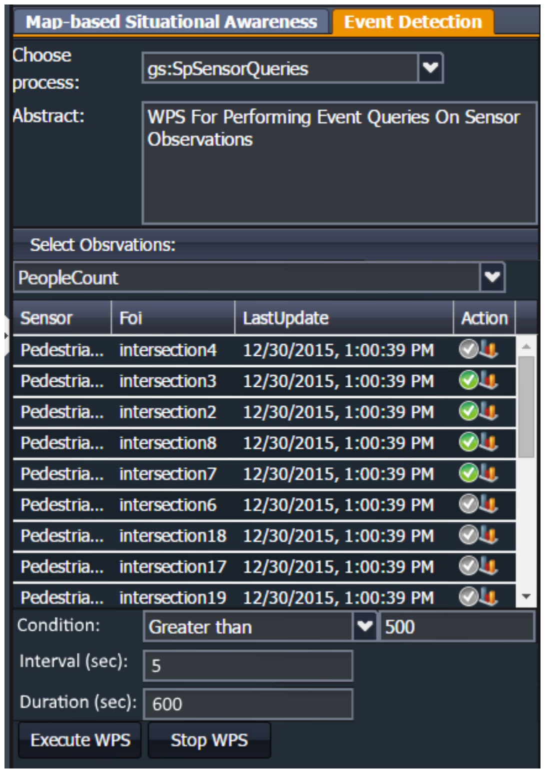


Figure 14: Event Detection Workflow

The workflow begins with the UM client submitting a GetCapabilities request to the WPS server. The GeoServer returns the list of published processes, including “gs:UMEventDetection” which is the identifier for the developed process. In case the user selects this process, the client submits a DescribeProcess request to the WPS (passing the UM WPS process identifier) and a GetCapabilities request to the SOS server. The ProcessDescription and SOS Capabilities document are then parsed and analyzed by the client, whereby the relevant elements are extracted and populated into the client as process input parameters (Figure 15).

Once the values for all the process input parameters are determined, the client uses the values to generate the WPS Execute request and post it to the WPS server. The WPS returns an ExecutionID as a response (since the process is asynchronous). Meanwhile the WPS starts the UMEventDetection process with the requested input parameters including the condition (e.g., greater than or smaller than), threshold value, rate, duration, and a template for GetObservation requests. The process then sends a GetDataAvailability request to the SOS server to check whether there is any recent observation available for the procedure (through analyzing the phenomenonTime included in the GetDataAvailability response).



Choose process: gs:SpSensorQueries

Abstract: WPS For Performing Event Queries On Sensor Observations

Select Obsrvations: PeopleCount

Sensor	Foi	LastUpdate	Action
Pedestria...	intersection4	12/30/2015, 1:00:39 PM	✓
Pedestria...	intersection3	12/30/2015, 1:00:39 PM	✓
Pedestria...	intersection2	12/30/2015, 1:00:39 PM	✓
Pedestria...	intersection8	12/30/2015, 1:00:39 PM	✓
Pedestria...	intersection7	12/30/2015, 1:00:39 PM	✓
Pedestria...	intersection6	12/30/2015, 1:00:39 PM	✓
Pedestria...	intersection18	12/30/2015, 1:00:39 PM	✓
Pedestria...	intersection17	12/30/2015, 1:00:39 PM	✓
Pedestria...	intersection19	12/30/2015, 1:00:39 PM	✓

Condition: Greater than 500

Interval (sec): 5

Duration (sec): 600

Figure 15: Deployed Panel in UM Client for Defining UM WPS Process Input Parameters

In case of availability of recent observations, the process returns “Running” as process output. It then starts repetitive operations including retrieving the latest observations from the SOS server, analyzing each observation against the threshold value, as well as generating and publishing a RSS feed into the Data Store for Notifications in case the observation result meets the threshold condition. Upon receipt of the InsertRSS request form the process, the Data Store for Notifications processes the new RSS feed and stores it in a database. The client can then pull the new RSS feeds from the data store using the GetRSS operation. Figure 16 illustrates the UM client visualizing a notification pulled from the data store.

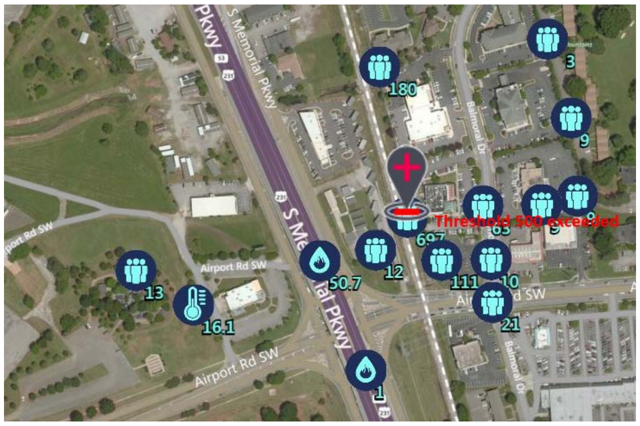


Figure 16: UM Client Visualizes a Threshold Notification that is Pulled from Data Store for Notifications

9.1.3.2 Operations and Request Example

The UM WPS supports the following operations specified by the OGC WPS 1.0 standard:

- GetCapabilities: Requesting details of the service offering, including service metadata and metadata describing the available processes;
- DescribeProcess: Requesting a description of a WPS process available through the service; and
- Execute: Requesting the execution of the process with specified input values.

Listing 1 below shows an example of an UM WPS Process execute request. The process takes a number of LiteralData as input parameters including getObservationTemplate, condition, threshold, intervalRate, duration and dataStoreForNotificationEndpoint. The process then performs the above-mentioned operations using the supplied inputs. It also returns an output indicating whether the process successfully accepted the request or not.

Listing 1: Example of UM WPS Process Execute Request

```
<?xml version="1.0" encoding="UTF-8"?>
<wps:Execute version="1.0.0" service="WPS"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xmlns="http://www.opengis.net/wps/1.0.0"
             xmlns:wfs="http://www.opengis.net/wfs"
             xmlns:wps="http://www.opengis.net/wps/1.0.0"
             xmlns:ows="http://www.opengis.net/ows/1.1"
             xmlns:gml="http://www.opengis.net/gml"
             xmlns:ogc="http://www.opengis.net/ogc"
             xmlns:wcs="http://www.opengis.net/wcs/1.1.1"
             xmlns:xlink="http://www.w3.org/1999/xlink"
             xsi:schemaLocation="http://www.opengis.net/wps/1.0.0
```

```
                                http://schemas.opengis.net/wps/
                                1.0.0/wpsAll.xsd">
   <ows:Identifier>gs:UMEventDetection</ows:Identifier>
   <wps:DataInputs>
      <wps:Input>
         <ows:Identifier>getObservationTemplate</ows:Identifier>
         <wps:Data>
            <wps:LiteralData>http://iddss-sensor.cdmps.org.au:8080/
                            52n-sos-webapp/service?service=SOS
                            &amp;version=2.0.0
                            &amp;request=GetObservation
                            &amp;offering=PedestrianCounting1
                            &amp;observedProperty=PeopleCount
                            &amp;procedure= PedestrianCounting1
</wps:LiteralData>
         </wps:Data>
      </wps:Input>
      <wps:Input>
         <ows:Identifier>condition</ows:Identifier>
         <wps:Data>
            <wps:LiteralData>greaterThan</wps:LiteralData>
         </wps:Data>
      </wps:Input>
      <wps:Input>
         <ows:Identifier>threshold</ows:Identifier>
         <wps:Data>
            <wps:LiteralData>500</wps:LiteralData>
         </wps:Data>
      </wps:Input>
      <wps:Input>
         <ows:Identifier>intervalRate</ows:Identifier>
         <wps:Data>
            <wps:LiteralData>5</wps:LiteralData>
         </wps:Data>
      </wps:Input>
      <wps:Input>
         <ows:Identifier>duration</ows:Identifier>
         <wps:Data>
            <wps:LiteralData>600</wps:LiteralData>
         </wps:Data>
      </wps:Input>
      <wps:Input>
         <ows:Identifier>
            dataStoreForNotificationEndpoint
         </ows:Identifier>
         <wps:Data>
            <wps:LiteralData>
               http://115.146.95.46:8080/iddss-service
            </wps:LiteralData>
         </wps:Data>
      </wps:Input>
   </wps:DataInputs>
   <wps:ResponseForm>
      <wps:RawDataOutput>
         <ows:Identifier>wpsResult</ows:Identifier>
```

```
        </wps:RawDataOutput>
    </wps:ResponseForm>
</wps:Execute>
```

9.2 Pros/Cons

9.2.1 52°North

9.2.1.1 Pros

- The WEPS allows encapsulating event processing functionality in an adopted OGC standard service (WPS 2.0); this has great potential because such a solution is not yet available (previous standardization efforts such as Sensor Alert Service (SAS) and Sensor Event Service (SES) have not resulted in an adopted standard).
- Implementation was possible in a straightforward manner.
- Standardized data access interfaces (i.e., SOS) allow the flexible querying of new observations to push the data into the event processor.

9.2.1.2 Cons

- The pull-based access to the observation data (through the SOS interface) could be optimized by a publish/subscribe pattern; it would be interesting to investigate how the emerging OGC Pub/Sub standard could help in this context.
- There is no common agreement how to structure WPS Execute requests for creating event subscriptions; a corresponding WPS profile (WEPS) would be desirable to cover this functionality.
- The WPS interface would need further functionality for managing subscriptions. This could be covered by a WEPS.
- There are different ways to encode the rules for event pattern detection. At the OGC, an approach has been described in the EML discussion paper. There are further best practices/de-facto standards used in practice, which should be supported by a WEPS profile.

9.2.2 Compusult

9.2.2.1 Pros

- Simple Publishing: Abstracts the complications of creating complicated CSW Insert Transactions.

- Valid Data: Ensures that all services are cataloged correctly and therefore can be discovered by other users.

9.2.3 UM

9.2.3.1 Pros

- Interoperability of Sensor Data Analysis: Encapsulating the algorithms for sensor data analysis using WPS results in interoperable description of processing functions. Consequently, interoperable access to information products derived from analysis and modeling of sensor data is provided.

9.2.3.2 Cons

- Lack of Maturity of Tools for WPS Development: During the recent years, a number of open source toolkits and applications were developed to support OGC WPS specification. Despite of these advancements, it is still too difficult and time-consuming to work with the current tools for developing WPS processes. Inflexibility of GeoServer's implementation of WPS was a main issue that we encountered during WPS development.

9.3 Recommended Changes

9.3.1 WEPS

Based on the experiences gained during the IMIS IoT Pilot, a strong recommendation would be to develop an event processing profile/extension for the OGC WPS 2.0 standard (i.e., WEPS). Such a specification should address the following requirements:

- Specify a basic template for WPS Execute requests that allow the submission of event subscriptions

- Provide guidance on how to encode event pattern detection rules. Besides recommendations on different feasible languages for encoding such rules, a mechanism will be needed to determine which event pattern languages are supported by a WPS server.

- Specify additional operations for managing event subscriptions (e.g., updating and terminating subscriptions); the emerging OGC Pub/Sub standards could be useful for this functionality.

- Provide guidance on how to flexibly couple a WEPS server to different notification publication mechanisms. In the testbed, RSS feeds were used for delivering notifications. Other push based technologies and archives for detected events should be described as well, however. Most likely this will not require additional specification work but guidance to apply existing standards for this purpose.

- Support different observation feeding mechanisms into the event processor (e.g., using the OGC Pub/Sub specification, MQTT or feeder connected to pull-based OGC services).

10 Sensor Observation Service (SOS)

10.1 Implemented Solution

10.1.1 Compusult

The Compusult SOS implementation provides data from mobile devices running Compusult's GoMobile software. These devices provide measurements of battery information, temperature, humidity, light level, speed, bearing, location, etc.

Three operations are currently supported by the service: GetCapabilities, DescribeSensor and GetObservation. The GetObservation operation will allow Temporal, Spatial and ID filters.

When the SOS is started for the first time, it will register itself with the Catalog using the Compusult publishing WPS, by performing the insert operation. The UUID output by the insert is stored in the database and used to perform the WPS update operation when changes have been made to the SOS. If the SOS is shutdown, it will use the stored UUID to perform the WPS delete operation and unregister itself from the Catalog.

10.1.2 UM

The UM SOS implementation aimed to test the capabilities for dealing with live sensor feeds and feeding the live SOS data to other software components that are developed as part of the pilot. For this purpose, the UM SOS was set up based on the 52°North SOS 4.x development line which in turn provides an implementation of the OGC SOS 1.0.0 and 2.0 standards.

Figure 17 shows the UM SOS alongside with other software components that are interacting with the server in real time. These components include UM client, UM SOS Simulating Wizard, UM WPS, 52°North WEPS, UM Data Store for Notifications and Compusult HubCat that are developed and bound during IMIS IoT Pilot testbed to interactively work together.

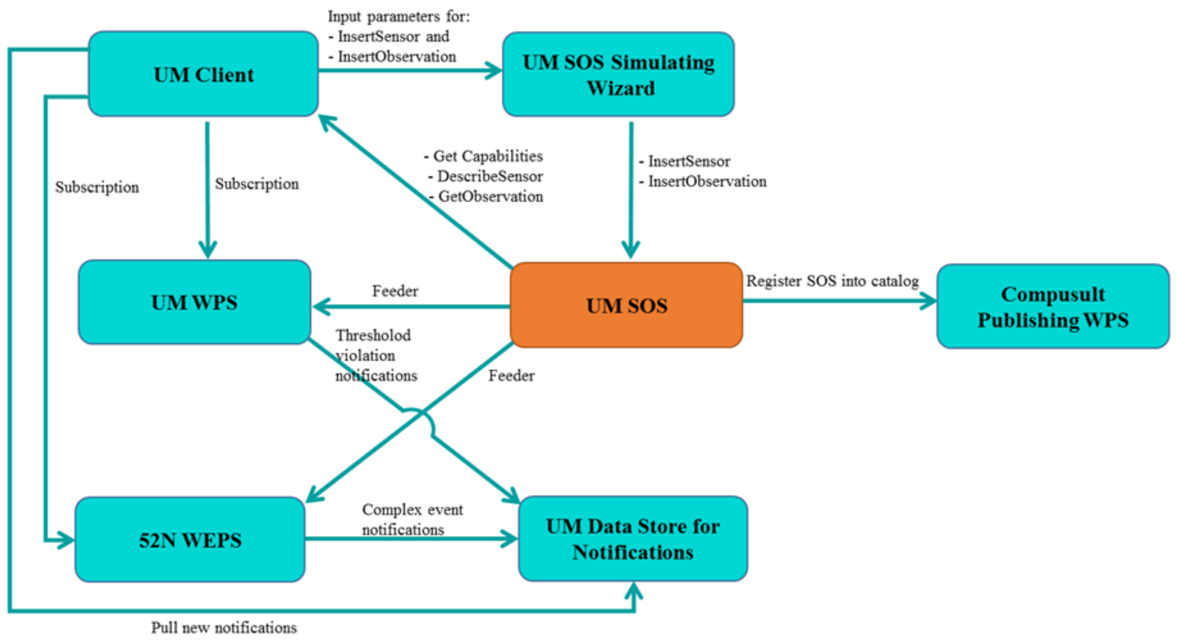


Figure 17: UM SOS Together with the Components Developed for Real Time

The UM Client is a GIS-based Web application that provides an interface for user interaction with the SOS server. Regarding sensor data publication into SOS, the client offers tools and commands for obtaining user inputs. The UM SOS Simulating Wizard component is developed for on-the-fly simulation and publication of sensor feeds into the SOS server. The remainder components are the WPS servers that are developed to interact with the SOS instance and analyze its live sensor data. In this regard, Compusult's Publishing WPS registers the SOS with the Compusult Catalog Service upon performing the insert operation. The UM WPS, 52°North WEPS and UM Data Store for Notifications enable real-time analysis of SOS sensor feeds to detect events. These WPS servers are described in detail in the WPS section of this ER. With an emphasis on UM SOS itself and the developed client capabilities, the remainder of this section describes the supported operations and how these operations are used in practice for real-time publication and retrieval of sensor feeds.

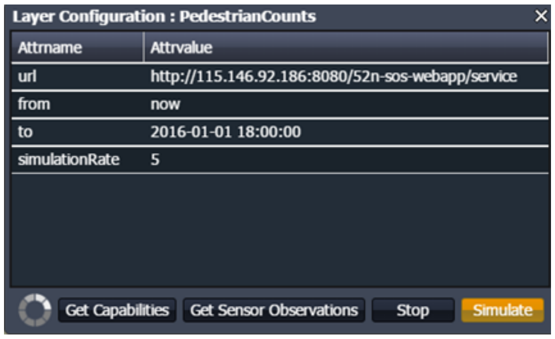
The UM SOS supports the following operations specified by the SOS 2.0 standard:

- GetCapabilities: Retrieving metadata about the SOS server;
- DescribeSensor: Retrieving metadata about the sensors; and
- GetObservation: Retrieving live sensor observations.

In addition, the UM SOS supports the following transactional SOS operations to publish the generated observations:

- InsertSensor: Registration of new sensors; and
- InsertObservation: Inserting new observations published by an already registered sensor.

To enable the flow of live sensor feeds, a number of sensors with different observed properties, including chemical, temperature, pedestrian and traffic count, were described based on SensorML and inserted into the UM SOS. When the sensors are registered, the SOS allows for insertion of observations for the registered sensors using the InsertObservation operation. Given the aim to examine the capabilities of the SOS specification for dealing with real-time sensor feeds, a pragmatic approach was needed for on-the-fly simulation, description and publication observations on the SOS server. This necessity was addressed by developing UM SOS Simulating Wizard which is a mediator software component between an SOS server and the UM client. Figure 18 shows the graphical user interface (GUI) of the SOS simulator that is available in the UM client, whereby the user can define the input parameters (e.g., rate and temporal filter for simulation of sensor observations). The simulation workflow includes time-incremental generation of observations, the encoding of the observations based on O&M and the insertion of the generated O&M files into the SOS server.


Layer Configuration : PedestrianCounts

Attrname	Attrvalue
url	http://115.146.92.186:8080/52n-sos-webapp/service
from	now
to	2016-01-01 18:00:00
simulationRate	5

Figure 18: GUI of UM SOS Simulating Wizard

While the sensor observations are being generated and published into the SOS server, the UM client enables the capability for retrieval of SOS observations in real time. The workflow for getting observations includes time-incremental execution of GetObservation operation and visualization of the retrieved data feeds on a map. Consequently, the user is provided with concurrent access to the multiple layers of live SOS observations (Figure 19). For visualizing time-series sensor observations in the UM client (which is based on Cesium), the CZML[6] format was used. CZML is an open JSON

[6] https://github.com/AnalyticalGraphicsInc/cesium/wiki/CZML-Content

schema for describing properties that change value over time in a Web browser running Cesium[7].

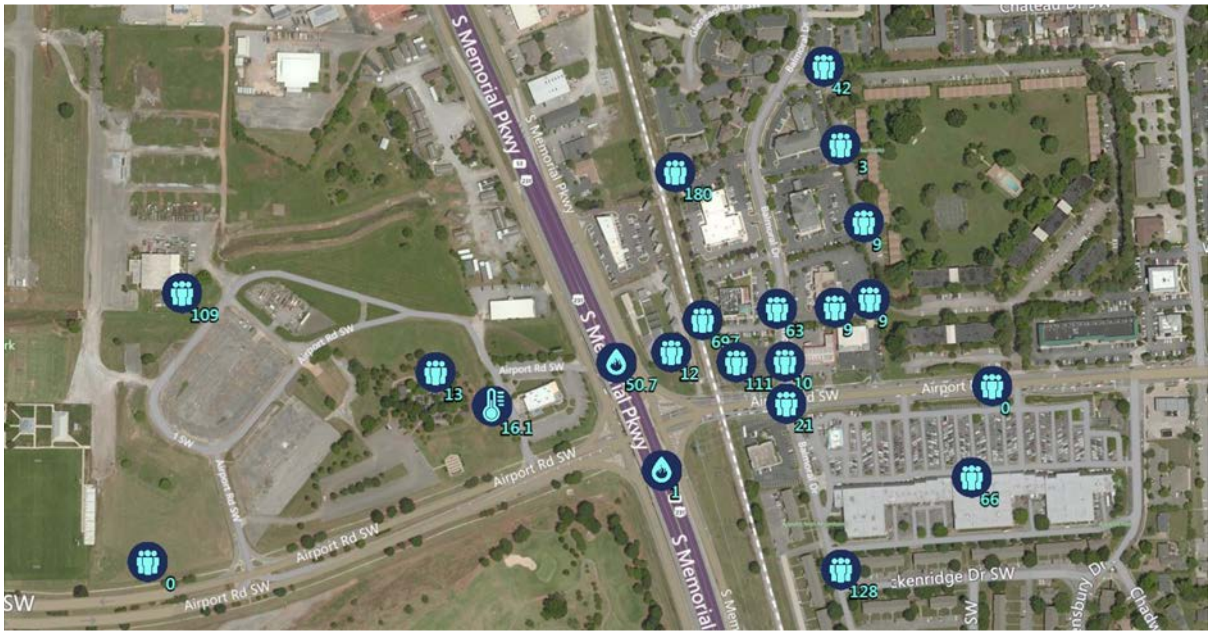


Figure 19: UM Client Provides Concurrent Access to Multiple Layers of Live SOS Observations

In addition to the map-view of observations, the user can use the chart-view analysis to examine the timeline of the measurements made by the selected sensor(s), shown in Figure 20.

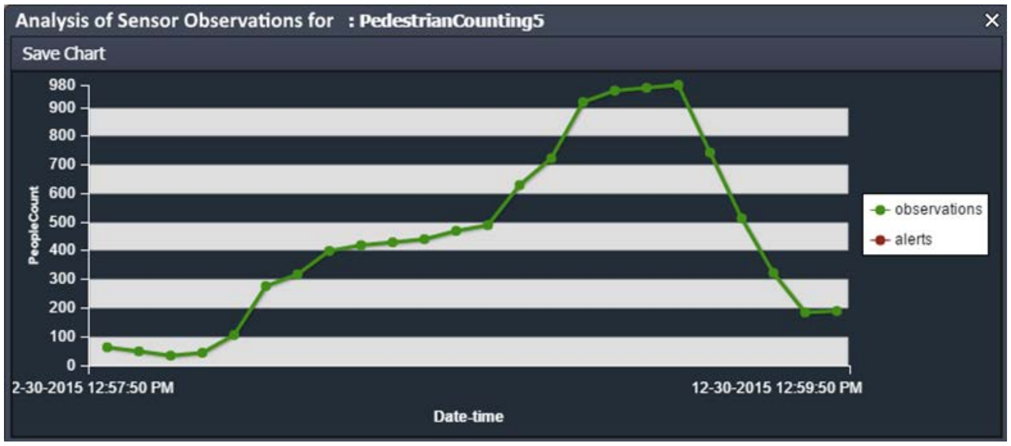


Figure 20: UM Client Provides Chart-view of Live SOS Observations

[7] https://cesiumjs.org/

10.1.3 OpenSensorHub (OSH)

OpenSensorHub (OSH) is an open source, open standard software stack that implements the full vision of the SWE service and encoding standards in an S-Hub component and provides these in an easily deployed package. OSH is highly scalable and configurable, and has been deployed on a variety of platforms from Android cell phones and tablets, to Linux/Windows/IoS devices, ARM boards, Raspberry Pi, various microcontrollers and on the Amazon Web Services (AWS) Cloud. OSH S-Hubs on all of these platforms can be interconnected to provide distributed access to a wide range of types and scales of sensor data (Figure 21).

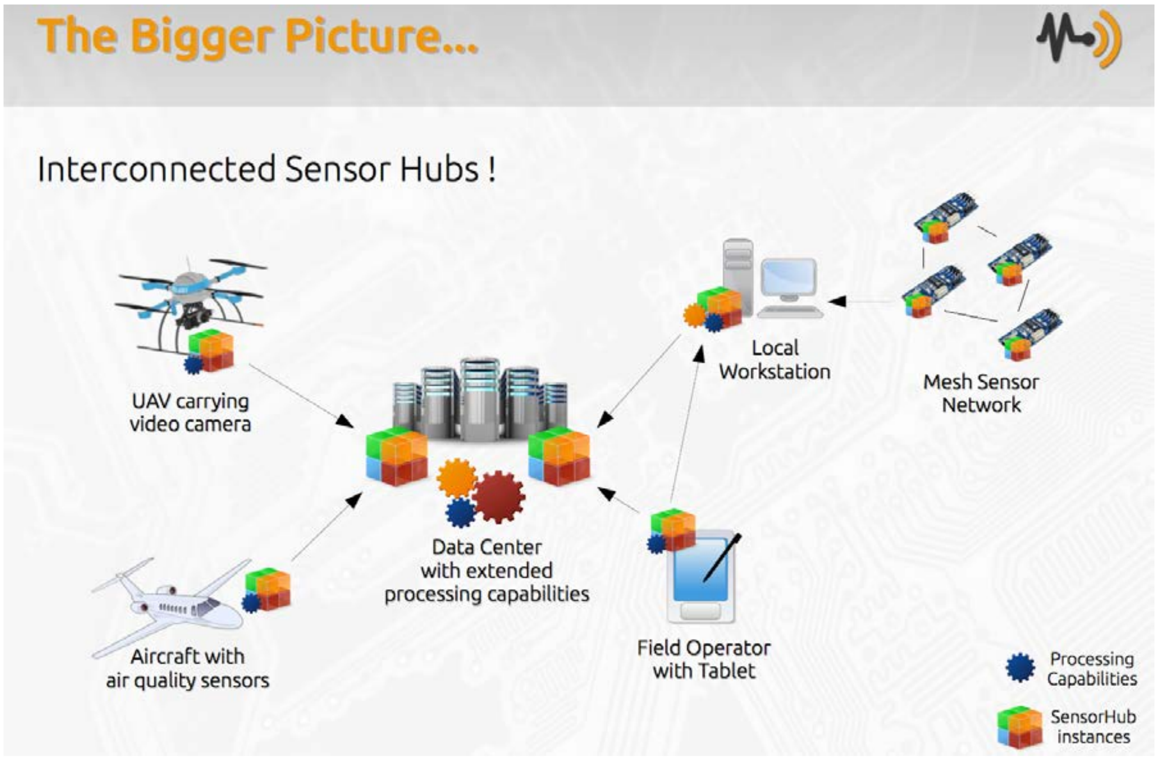

The Bigger Picture...
Interconnected Sensor Hubs !
UAV carrying video camera
Local Workstation
Mesh Sensor Network
Data Center with extended processing capabilities
Aircraft with air quality sensors
Field Operator with Tablet
Processing Capabilities
SensorHub instances

Figure 21: Interconnected Sensor Hubs

In addition to SOS capabilities, OSH supports SOS Transactional Services (SOS-T); Sensor Planning Service (SPS) for tasking sensors, actuators and executable processes; SensorML-encoded on-board processing; and efficient streaming of real-time or archived observation values. OSH was deployed in this pilot project to support:

- Real-time streaming of video and sensor location/orientation for Android phones/tablets to the AWS Cloud;

- Real-time streaming and processing of video and sensor location/orientation for a vehicle-mounted Pan-Tilt-Zoom (PTZ) video camera to the AWS Cloud;

- Real-time streaming of simulated Automated Vehicle Location (AVL) location data from Huntsville Fire and Rescue, Huntsville Police and Huntsville Emergency Medical Service (EMS) vehicles;

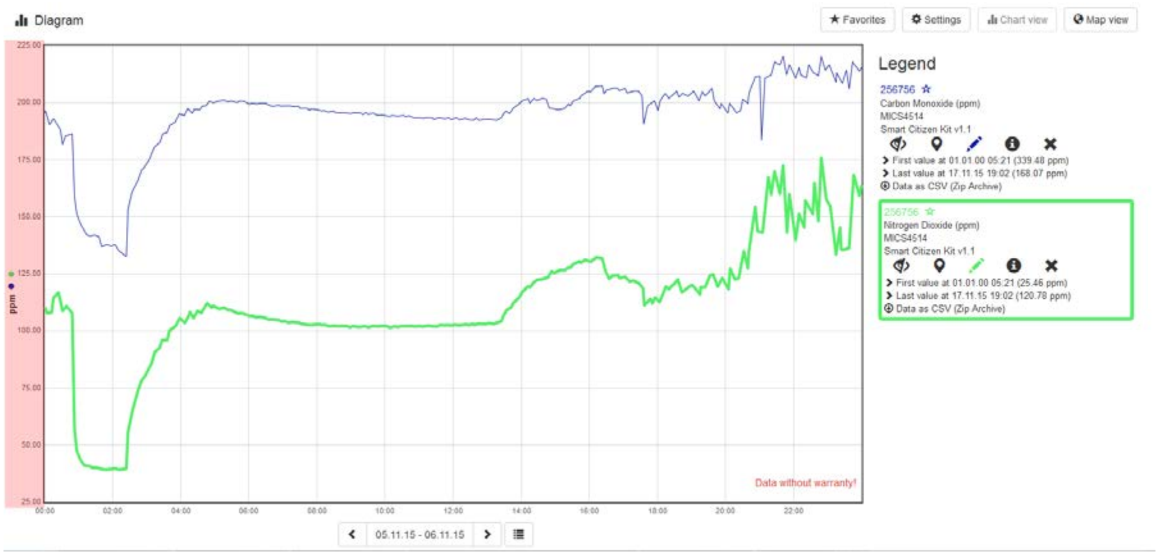
- Real-time processing and streaming of a Laser Rangefinder to support remote location tagging;

- On-demand tasking (through SPS) of a Lagrangian Plume Model and serving (through SOS) the resulting model observations; and

- Storage and streaming of Unmanned Aerial Sensor (i.e., drone) navigation data (location, attitude, gimbal positions and camera settings and HD video, along with on-demand geolocation of the camera footprint).

10.1.4 52°North (Client Only)

Within the IMIS IoT Pilot 52°North contributed its JavaScript Sensor Web Client for allowing users to explore and visualize the available data sets in different ways (e.g., map view showing sensor locations and the latest measured values, diagram view, table view). Within the IMIS IoT Pilot this client was enhanced so that it is not only capable of consuming data from SOS servers but also from STA endpoints (Figure 22). The architecture of this development is illustrated in Figure 23. In this case, the SOS Client does not interact directly with the SOS and STA endpoints. Instead, it relies on an intermediate component, which encapsulates the business logic to interact with SOS and STA endpoint behind a REST/JSON-based interface. Furthermore, this component caches metadata about available data in the different registered SOS and STA endpoints. The reason for this is, that certain client functionality requires more detailed information about the contents of SOS/STA endpoints, which cannot be directly obtained by single service operation calls. Instead, this information is cached and made available to the client through convenience operations (for more details please refer to the IMIS IoT Architecture ER).



Legend

256756 ☆
Carbon Monoxide (ppm)
MICS4514
Smart Citizen Kit v1.1
First value at 01.01.00 05:21 (339.48 ppm)
Last value at 17.11.15 19:02 (168.07 ppm)
Data as CSV (Zip Archive)

256756 ☆
Nitrogen Dioxide (ppm)
MICS4514
Smart Citizen Kit v1.1
First value at 01.01.00 05:21 (25.46 ppm)
Last value at 17.11.15 19:02 (120.78 ppm)
Data as CSV (Zip Archive)

Figure 22: 52°North Client Application for Accessing SOS and STA

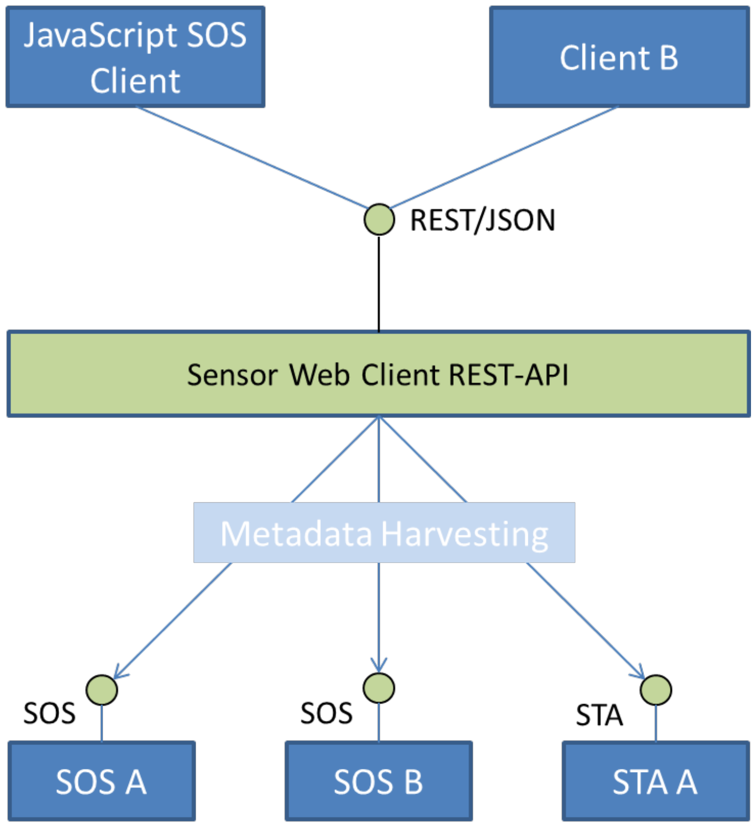


Figure 23: Architecture of the 52°North SOS/STA Client Developments

10.2 Pros/Cons

10.2.1 Compusult

10.2.1.1 Pros

- Automatic Registration: The SOS is published to the Catalog when activated, ensuring that it will be discoverable for all users. The Catalog is also notified of any updates and the SOS is removed when it is no longer active.

10.2.2 UM

10.2.2.1 Pros

- Versatility of SOS Capabilities Document: The Capabilities document is a very useful resource in the process of binding the software components to the SOS server. The Capabilities document provides the answer to the questions such as what sensors are available in the SOS server, what observed properties are monitored, what is the last update time for each sensor, etc. Since the answers to these questions can frequently change during the course of time (e.g., through insertion or deletion of sensors and observations), the existence of the Capabilities document as a centralized reference resource makes it easier for the developers to implement functionalities on-top-off SOS servers.

- Reliability of 52°North 4.x Development Line: We found 52°North Implementation of OGC SOS a well-developed and reliable server for dealing with live sensor feeds. The server never crashed or slowed under extensive data entry and concurrent operation execution that we undertook during the pilot. Also, this implementation of SOS specification additionally supports JSON binding which significantly improves the developer experience while interacting with the server.

10.2.2.2 Cons

- Heavy Weight: A RESTful binding is not specified in the current OGC SOS 2.0 standard. It supports Simple Object Access Protocol (SOAP) and Key-Value Pair (KVP) bindings. Consequently, a lot of bandwidth is devoted to communicating redundant metadata. Also, the SOAP binding is relatively hard to implement and is unpopular among mobile developers. As a result, high communication overhead and heavy power consumption pose a limitation for usage of SOS in the cases when the bandwidth is limited (such as mobile development).

- Scalability of the Capabilities Document: Despite the versatility of the Capabilities document, it poses an issue with relation to scalability. In this regard, when the number of items that are advertised in the Capabilities document increases (e.g., large number of sensors, observed properties and feature of interests) the size of the Capabilities document is directly affected. This is the case for mobile sensor platforms in which all of the sampling features of interest appear in the Capabilities document and make its size too heavy to process.[8]

10.2.3 OSH

10.2.3.1 Pros

- Scalable to a wide range of platforms from microcontrollers and cell phones to the cloud, this allows services to be deployed where they make the most sense.
- Supports both simple and complex sensors.
- Ability to deploy sensors, actuators and processing on the same hub provides flexibility for distributed capabilities.
- Support for GetResult and GetResultTemplate for real-time and archived data provides highly efficient data streaming capabilities.
- Supports both ASCII-comma separated value (CSV) and binary encodings for data blocks and data streams.

10.2.3.2 Cons

- Support is in progress but not yet available for JSON. Note, however, that CSV streaming combined with one-time call to GetResultTemplate for result metadata, structure and encoding is actually more robust and efficient than JSON; still the option for JSON encoding support would be helpful to many.

10.2.4 52°North

10.2.4.1 Pros

- The SOS interface ensures interoperability when integrating observation data from multiple data providers.
- The enhancements of an existing SOS client to cover the STA standard, as well, was possible in a very efficient way.

[8] The GetDataAvailability operation (e.g., described in the OGC SOS 2.0 Hydrology Profile Best Practice document) has the potential to address this issue.

10.2.4.2 Cons

- SOS (and STA) sometimes requires multiple calls to determine the metadata necessary for typical client functionality (e.g., determine which time series with specific characteristics are available. Solutions such as the GetDataAvailability operation (see SOS 2.0 Hydrology Profile Best Practice) are not part of the core SOS standard.

10.3 Recommended Changes

10.3.1 SOS/Sensor Things Profiling

The authors discovered a number of interoperability issues with both standards which were more operational issues rather than failures. Firstly, the benefit of two standards delivering the same information is always questionable. If they are two mandated endpoints then this is fine, but if they are alternatives this simply passes the issue on to the client. Thus, it is necessary to mandate both (i.e., a server must support both) or mandate one. Alternatively, adapter services, again registered with the Catalog, would be another approach. This cannot be left to the client to deal with or everyone will be doing everything twice, however. Secondly, within SOS there is no mandated return format. WMS clients ensure that users will be able to get a Portable Network Graphics (PNG) formatted image even if a client can’t process other formats.

This may be the only way to operate for SOS in general (due to the range of sensors), but for a domain such as IMIS IOT it reduces interoperability. There is a need, per information type, to agree on the return type and thereby agree on a canonical set of result types. This is clearly a domain profiling issue.

10.3.2 Visualization of Sensors

There is clearly a need for a range of visualization techniques for sensors, which are very different in return content even though they have a similar invocation protocol. Discovering and querying an SOS proved relatively easy compared with negotiating the return result semantics.

The Envitia Horizon Client implemented SOS visualization as did the Envitia InSight mobile App. Both used static visualization (displayed geographic position with a symbol and allowed display of attributes by clicking on the object, typically as simple tabular lists).

Clearly a model to define visualization of SOSs using Styled Layer Descriptors (SLD) would be valuable, but there also seems value in taking the next step and supporting a ‘SOS or SensorThings’ portrayal service delivered as a WMS. This is analogous to the Feature Portrayal service (Component WMS) or SLD supporting Integrated SLD-WMS service, both of which are typically used to visualize WFS and WCS. These services provide broader interoperability than SOS as WMS is widely supported.

It is likely that the integrated approach (where the WMS is deployed against an SOS endpoint) is the most interoperable and could support both pre-loaded styles and SLD requests (offering scalability to clients). Compusult's CSW-WMS offers a demonstration of this type of capability, but our proposition is that it needs to be more focused on a specific SOS to be useful for visualization as opposed to discovery (which the Compusult CSW-WMS does well).

It would be possible to set up a WMS service which is orchestrated by the Catalog, which automatically deploys specific WMS endpoints as SOS servers register. This is an evolution of the concept implemented by Compusult and proposed by Envitia in their original proposal.

10.3.3 Determine Available Observation Data

For both the SOS and the STA it was not always possible for clients to determine the availability of data for certain time series or query criteria. To facilitate the development of lightweight client applications, an operation such as the GetDataAvailability operation should be specified as an extension and included in an IMIS SOS Profile.

10.3.4 OWS Context Document Extension Suggestions

The OWS Context document provides a framework to reference both Web Services and local content and embed content. This is all achieved through a concept known as an 'Offering.' As described in the IMIS IoT Architecture ER, the OWS Context document defines a geospatial extent, a temporal extent and an ordered series of layers known as Resources. Resources break down further into 'Offerings.' A given Resource or layer can have multiple 'Offerings.' The concept is that for a given layer the document can offer a client the opportunity to load the offering most appropriate to the function they need. The principle is that offerings within a layer offer the same information and so the client can choose the most appropriate.

10.3.4.1 SOS Offering Extension

The OWS Context 1.0 Standard allows additional offerings to be added, characterized by a Universal Resource Identifier (URI), in the form of a URL, that identifies a defined offering. Within the OWS Context 1.0 standard there are already a number of defined offerings (WMS, WMTS, WFS, CSW, GML etc.) but at present it does not support SOS.

Within the IMIS IoT Pilot, Envitia (co-chair of the OWS Context Standards Working Group) has developed an SOS Extension to OWS Context. Defining an OWS Context extension is a simple process. The capability to use the prototype extension has been added to the Envitia Horizon Web Portal and Envitia InSight Mobile App. The extension is compliant with OWS 1.0 so it could be defined in a profile.

An OWS Context Web service Offering defines two operation requirements, the GetCapabilities and a data request used to define the specific request to be used. These are shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--A Prototype SOS Offering within OWS Context  for SOS -->
<owc:offering code="http://www.opengis.net/spec/owc-atom/1.1/req/sos">
   <owc:operation code="GetCapabilities" method="GET"
                  href="https://imis.compusult.net/wes/PXSOS
                        ?REQUEST=GetCapabilities&amp;SERVICE=SOS"/>
   <owc:operation code="GetObservation" method="POST"
                  xmlns:sos="http://www.opengis.net/sos/2.0"
                  href="https://imis.compusult.net/wes/PXSOS">
      <owc:request type="application/xml">
         <sos:GetObservation version="2.0.0" service="SOS">
            <sos:offering>
               https://imis.compusult.net/wes/PXSOS/403/BATTERY
            </sos:offering>
            <sos:procedure>
               https://imis.compusult.net/wes/PXSOS/403/BATTERY
            </sos:procedure>
            <sos:observedProperty>
               https://imis.compusult.net/wes/PXSOS/403/BATTERY/PERCENT
            </sos:observedProperty>
         </sos:GetObservation>
      </owc:request>
   </owc:operation>
</owc:offering>
```

10.3.4.2 Using OWS Context Documents to Offer Multiple Service Options

Within the OWS Contexts documents used in the IMIS IoT Pilot, layers including an SOS server and related services such as the CSW-WMS were included. They were not created as offerings of one layer because they conceptually do not offer the same information, however; the CSW-WMS layer for Location does visualize the SOS but it also visualizes other SOS servers delivering position so it is not technically the same information.

If a WMS were created to visualize an SOS, it would make sense to offer both so that different client capabilities would be supported. The structure Resource (Layer) definition is shown below. The OWS Context Document is encoded in Atom (a dialect of XML) and resource is translated to an Atom 'entry' tag. The Resource then has a bounding extent and some basic metadata describing what it is, followed by two offerings from which the client can choose.

```
<?xml version="1.0" encoding="UTF-8"?>
<entry>
   <id>OpenLayers_Layer_Vector_1512</id>
   <title>GPS - Utility Crew</title>
   <updated>2016-01-11T23:44:37Z</updated>
   <georss:where>
      <gml:Polygon>
         <gml:exterior>
            <gml:LinearRing>
               <gml:posList srsDimension="2" srsName="EPSG:4326">
                  34.690644924214 -86.583071822499
                  34.696407892743 -86.583071822499
                  34.696407892743 -86.573400869196
                  34.690644924214 -86.573400869196
                  34.690644924214 -86.583071822499
               </gml:posList>
            </gml:LinearRing>
         </gml:exterior>
      </gml:Polygon>
   </georss:where>
   <content type="html">GPS - Utility Crew</content>
   <category term="false" scheme="http://www.opengis.net/owc/active"/>
   <category term="1" scheme="http://www.envitia.com/horizon/
                              layer/opacity"/>
   <!--Standard WMS Offering pointing to The SOS Visualising WMS -->
   <owc:offering code="http://www.opengis.net/spec/
                       owc-atom/1.0/req/sos">
   <!--Content removed for brevity -->
   </owc:offering>
   <!--Extension SOS Offering pointing to the SOS itself -->
   <owc:offering code="http://www.opengis.net/spec/owc-atom/
                       1.1-draft/req/sos">
   <!--Content removed for brevity-->
   </owc:offering>
</entry>
```

10.3.4.3 Sensor Things Implementation in OWS Context

Within the IMIS IoT Pilot, only SOS extensions were used. This involved accessing information from the SensorThings sensor hubs as they also published SOS interfaces. It is relatively easy to define OWS Context offerings which access SensorThings endpoints, and now that the STA has been approved as an OGC Standard, the recommendation is that it is raised as a request on the OWS Context Standards Working Group for inclusion as a standard offering together with a similar request for SOS.

11 SensorThings API (STA)

11.1 Implemented Solution

11.1.1 Compusult

Similar to the Compusult SOS, the Compusult SensorThings implementation provides data from mobile devices running Compusult's GoMobile software. The data from these devices is stored in a database. When a SensorThings request is received it is converted to a database query and the results are transformed to JSON and returned. The database will only keep the data available for a configurable amount of time, ensuring the service is not slowed down by large amounts of data processing. Since the purpose of this service is to provide the data collected from GoMobile, entities within the service cannot be created, updated or deleted with HTTP requests. The system query options $expand and $filter are not implemented; however, all other system query options are available. Like the Compusult SOS, the Compusult SensorThings implementation will automatically register/unregister from the Catalog.

11.1.2 SensorUp

The SensorUp STA is a comprehensive implementation of the OGC SensorThings API. At a high level, the SensorUp STA not only allows clients to retrieve and query IoT data, but also support IoT devices (sensors) to register themselves and upload readings. In addition, to minimize IoT devices' power-consumption and bandwidth-consumption, SensorUp STA supports the data array extension and MQTT extension. The list of conformance classes supported by the SensorUp STA is listed below. A detailed description of the SensorUp STA implementation is presented after the list.

The SensorUp STA implemented the following conformance classes:

- http://www.opengis.net/spec/iot_sensing/1.0/conf/thing
- http://www.opengis.net/spec/iot_sensing/1.0/conf/location
- http://www.opengis.net/spec/iot_sensing/1.0/conf/historical-location
- http://www.opengis.net/spec/iot_sensing/1.0/conf/datastream
- http://www.opengis.net/spec/iot_sensing/1.0/conf/sensor
- http://www.opengis.net/spec/iot_sensing/1.0/conf/observed-property
- http://www.opengis.net/spec/iot_sensing/1.0/conf/observation
- http://www.opengis.net/spec/iot_sensing/1.0/conf/feature-of-interest

- http://www.opengis.net/spec/iot_sensing/1.0/conf/entity-control-information
- http://www.opengis.net/spec/iot_sensing/1.0/conf/resource-path
- http://www.opengis.net/spec/iot_sensing/1.0/conf/request-data
- http://www.opengis.net/spec/iot_sensing/1.0/conf/create-update-delete
- http://www.opengis.net/spec/iot_sensing/1.0/conf/data-array
- http://www.opengis.net/spec/iot_sensing/1.0/conf/create-observations-via-mqtt
- http://www.opengis.net/spec/iot_sensing/1.0/conf/receive-updates-via-mqtt

11.1.2.1 Request Data

SensorUp STA supports all of the system query options, including $filter (for queries), $count/$skip/$top (for both server-side and client-side paginations), $orderby (for sortings), $expand (for saving the number of client requests) and $select (for saving data transmitted over the network). All of the above mentioned query options have been demonstrated useful in the IMIS IoT Pilot demonstration; these query options provide great flexibility for Web clients to navigate and retrieve the desired data with a single RESTful request.

Take a following use case as an example: a user would like to request the Observations from all the Datastreams whose ObservedProperty's name include "temp" (e.g., dew point temperature and air temperature). It can be fulfilled with the following single RESTful request:

> http://api.sensorup.com/OGCSensorThings/v1.0/ObservedProperties?$filter=substringof('temp',name)&$expand=Datastreams/Observations

In this example, the $expand query option plays an important role as it can significantly save the number of the requests for a client sent to the server.

11.1.2.2 Data Array Extension

SensorUp STA data array implementation is very useful to reduce the data transmitted over the network and shorten the server response time. As network bandwidth is a scarce resource when a major disaster strike, the data array feature can be very important.

An example for this IMIS IoT Pilot is for a client to retrieve the heart rate data from a smart shirt. The example request is as follows:

> http://api.sensorup.com/OGCSensorThings/v1.0/Things(<thing_id>)/Datastreams(<datastream_id>)/Observations?$resultFormat=dataArray

The response below shows the very compact encoding of the STA data array:

```
{
  "dataArray@iot.count": 5760,
  "@iot.nextLink": "http:\/\/api.sensorup.com\/OGCSensorThings\
                    /v1.0\/Things(<thing_id>)\/Datastreams
                    (<datastream_id>)\Observations?$resultFormat=
                    dataArray&$top=100&$skip=100",
  "components": [
    "@iot.id",
    "phenomenonTime",
    "result",
    "resultTime"
  ],
  "dataArray": [
    [
      1185124,
      "2016-01-5T05:00:00.000Z",
      "82",
      null
    ],
    [
      1185119,
      "2016-01-25T04:00:00.000Z",
      "83",
      null
    ],
    [
      1185113,
      "2016-01-25T03:00:00.000Z",
      "82",
      null
    ],
    [
      1185105,
      "2016-01-25T02:00:00.000Z",
      "-81",
      null
    ],
    [
      1185095,
      "2016-01-25T01:00:00.000Z",
      "82",
      null
    ]
  ]
}
```

11.2 Pros/Cons

11.2.1 Compusult

11.2.1.1 Pros

- Automatic Registration: The SensorThings service is published to the Catalog when activated, ensuring that it will be discoverable for all users. The Catalog is also notified of any updates and the SensorThings service is removed when it is no longer active.

- Data in JSON Format is Easy to Understand: It is simple for the service to generate and for another machine to parse. It is platform-independent and also very human-readable.

- Powerful Query Options: The service provides system query options, as well as system query functions following the OData Canonical function definitions listed in Section 5.1.1.4 of OData Version 4.0 Part 2: URL Conventions.

- Server-driven Pagination: The STA provides specification for server-driven pagination. This prevents server slowdowns by limiting the amount of data retrieved, as well as the response size.

11.2.1.2 Cons

- Query Options Can Be Very Complex: With many possibilities to combine and process data, queries can become very complicated.

- Information is Dispersed Over Multiple Pages: Data and location for a single device is organized into multiple entities. Multiple requests are often needed to retrieve all necessary information for a single device.

11.2.2 SensorUp

11.2.2.1 Pros

- Great Developer Experiences: STA provides great developer experiences. It is very easy for Web developers to pick up and start coding.

- Efficiency Designed for IoT Devices and Applications: STA considered the high volume and high velocity of the number of the IoT devices and the data collected by them.

- Built-in Publish/Subscribe Support via MQTT: Receiving data as it happens is particularly important for emergency response applications. STA's native support of MQTT provides a consistent approach for both receiving near real-time updates (via MQTT) and accessing historical data (via HTTP).

11.3 Recommended Changes

11.3.1 Discovery of URL and Port Number of MQTT Assoicates

In the specification, it is not mentioned how a client can discover the URL and port number of the MQTT associate with the STA. Location of such information needs to be defined in the specification.

11.3.2 Navigation Links

Navigation links should use non-relative URLs. A "navigationLink" property within the current specification only holds a relative URL. To follow a navigation link, a non-relative URL must be generated from the server endpoint and the relative URL of the navigation link property. The server already must generate non-relative URLs because the "selfLink" property is defined as a non-relative URL. Using non-relative URLs would provide one less processing step to any machine parsing the service as it could read the property directly as a URL. This would also allow simpler navigation for human users of the service.

11.3.3 Relationship to the SOS Standard

See Section 10.3.1 about further recommendations on the relationship between SOS and the STA.

11.3.4 Harmonization with Similar Activities

Besides the STA there are ongoing activities within the OGC community to work on recommendations for applying JSON and REST concepts. To ensure consistency, a close alignment between these activities and future versions of this standard are important.

12 Observations and Measurements (O&M) - XML Encoding

The O&M XML Encoding (OGC 10-025r1) is used in GetObservation responses of SOS servers to encode the observation information.

12.1.1 Implemented Solution

The SOS servers have largely provided their observations using the measurement requirements class from the O&M – XML Encoding specification (OGC 10-025r1, p. 17/18). Two examples for chemical sensors and pedestrian count are given in Annex A3 and Annex A4.

12.2 Pros/Con

12.2.1 Pros

As the SOS servers have used the same result types, implementing interoperable clients becomes easier. As the name suggests, however, the observation type for providing the pedestrian counts may have been an observation with count result type following the countObservation requirements class of the O&M – XML Encoding standard (OGC 10-025r1, p. 18/19).

12.2.2 Cons

The repeating of metadata, as well as the verbosity of the XML structure, is inefficient for supporting highly dynamic observations. Support for time-series tuples (e.g., weather data consisting of time, location, temperature, pressure, wind speed, wind direction and rainfall) is complicated by the O&M model and XML encoding.

12.3 Recommended Changes

The usage of certain observation types for specific sensor/observed property constellations does not seem obvious and straight forward in some cases. We therefore recommend providing tutorials and best practices for choosing and implementing appropriate observation types.

13 Data Streaming with SWE Common Data

13.1 Implemented Solution

The SWE Common Data standard provides a means to robustly describe data records, arrays, vectors and simple components, including the data structure and encoding. For individual components such as quantities, counts, Booleans, categories and time, metadata support includes unit of measurement, semantic definition, constraints, measures of quality, labels, description, etc.

SWE Common Data is utilized within SensorML for describing properties and data components, in SOS for describing results, in SOS-T for defining the incoming data stream, in SPS for describing tasking parameters, and in O&M for providing an optional result metadata and values. The standard is also used in WCS for defining coverage data.

Although GetObservation and O&M is supported in OpenSensorHub (OSH), the demonstrations within the IMIS IoT Pilot make exclusive use of SWE Common Data descriptions and encodings to support data streaming and highly efficient data block transfer, including both ASCII and binary data. SWE Common Data also supported SPS tasking, SensorML descriptions, and SensorML encoded processing. It is the fundamental data protocol used internally within OpenSensorHub.

13.2 Pros/Cons

13.2.1 Pros

- Robust metadata about the results are provided, and is only needed to be retrieved once.
- Unlike JSON and XML encodings in O&M, metadata is not repeated with each measured value allowing for very efficient streams of data.
- SWE Common Data supports both ASCII and binary/compressed data.
- Since the data stream block consist entirely of values, SWE Common Data provides a highly efficient format for supporting a wide range of data.

13.2.2 Cons

- Parsing of SWE Common Data is relatively easy but is not automatically supported by Web browser technologies as is JSON.
- General library for reading/writing SWE Common Data is currently Java only, making it ideal for servers but not necessarily clients.

- As is the case with any data format, whether JSON, O&M, traditional proprietary files or SWE Common Data, it is still very challenging for a generic client to know exactly what to do with a collection of new data without having a priori knowledge of that data. There is a need for at least some common data profiles that are recognizable by the client (see recommendation below).

13.3 Recommended Changes

Recommending not changes, but extensions to SWE Common Data to provide:

- A JSON encoding option based on the models presented in the standard; current encoding of these models is XML only.

- Creation of common profiles for location/orientation, video, imagery, weather, etc. These profiles should be placed in a registry to provide interoperable descriptions of common observation types.

14 Sensor Model Language (SensorML)

14.1 Implemented Solution

SensorML provides a robust description of sensors and actuators, and executable models that can be used for discovery, qualification of results and configuration. Since SensorML models sensors, actuators and executable models as processes, these components can all be included in SensorML-described process chains or workflows to support on-demand tasking, data collection, analysis and reaction.

SensorML 2.0 introduced several improvements to the existing standard including better support for inheritance and more compact description of deployed sensors, actuators and processes. In addition, there is better support for security tagging, configuration descriptions and streaming of disparate messages (e.g., from a Chemical / Biological / Radiological / Nuclear (CBRN) sensing device).

OSH supports SensorML 2.0 through a combined use of predefined SensorML descriptions coming from Original Equipment Manufacturers (OEM) and system deployers, and SensorML descriptions generated in code based on current deployed OSH configurations. Each OSH node can support sensors, actuators and on-demand processing.

Furthermore, as robots are little more than a collection of sensors, actuators and processes, SensorML and OSH are well suited for supporting robotic systems, whether physical entities in specific locations or collections of distributed cooperative virtual components.

Much of the SensorML support was relatively hidden in the initial IMIS IoT Pilot, primarily supporting on-demand processing within OSH nodes (e.g., to support geospatial awareness for video cameras on motor vehicles, unmanned aerial vehicles, and personnel, and for calculation of remote locations tagged by the Laser Rangefinder).

14.2 Pros/Cons

14.2.1 Pros

- Provides a robust description of sensors, actuators and processes.
- Provides the ability to fully describe complex robotic systems or workflows consisting of local or distributed sensing, acting or processing components.
- Can be used for on-demand processing within (or external to) any OSH node.
- Processing within an OSH node can be reconfigured on-the-fly by referencing or sending a new SensorML-encoded process over the Web.

14.2.2 Cons

- A simple SensorML viewer/editor is vital to enable OEM's and sensor deployers to describe their systems.[9]
- As with SWE Common Data, a JSON encoding of SensorML is needed for easier Web-based usage (see recommendation).

14.3 Recommended Changes

As with SWE Common Data, an alternate JSON serialization of SensorML would be helpful to support descriptions of sensors, actuators and processes.

[9] NOTE: Such an editor is currently in the works with initial release planned for 2016.

Annex A

Extensible Markup Language (XML) Example

A.1 General

This annex lists several XML examples of service capabilities, requests or responses.

A.2 Web Feature Service (WFS) Capabilities Example

The document below shows the Capabilities document that is returned by the GetCapabilities operation of the WFS described in Section 7.

```
<?xml version="1.0" encoding="UTF-8"?>
<wfs:WFS_Capabilities xmlns:wfs="http://www.opengis.net/wfs/2.0"
                      xmlns:xsi="http://www.w3.org/2001/XMLSchema-
                                 instance"
                      xmlns:ows="http://www.opengis.net/ows/1.1"
                      xmlns:xlink=http://www.w3.org/1999/
                                  xlink
                      xmlns:fes="http://www.opengis.net/fes/
                                  2.0"
                      version="2.0.0"
                      xsi:schemaLocation="http://www.opengis.net/fes
                                          /2.0
                                          http://schemas.opengis.net/
                                          filter/2.0/filterAll.xsd
                                          http://www.opengis.net/wfs
                                          /2.0
                                          http://schemas.opengis.net/
                                          wfs/2.0/wfs.xsd
                                          http://www.opengis.net/
                                          ows/1.1
                                          http://schemas.opengis.net/
                                          ows/1.1.0/owsAll.xsd">
   <ows:ServiceIdentification>
      <ows:Title xml:lang="eng">52N WFS</ows:Title>
      <ows:Abstract xml:lang="eng">
         52North Sensor Observation Service - Data Access for the
         Sensor Web
      </ows:Abstract>
      <ows:ServiceType>WFS</ows:ServiceType>
      <ows:ServiceTypeVersion>2.0.0</ows:ServiceTypeVersion>
      <ows:Fees>NONE</ows:Fees>
      <ows:AccessConstraints>NONE</ows:AccessConstraints>
   </ows:ServiceIdentification>
   <ows:ServiceProvider>
      <ows:ProviderName>52North</ows:ProviderName>
      <ows:ProviderSite xlink:href="http://52north.org/swe"/>
      <ows:ServiceContact>
```

```
         <ows:IndividualName>TBA</ows:IndividualName>
         <ows:PositionName>TBA</ows:PositionName>
         <ows:ContactInfo>
            <ows:Phone>
               <ows:Voice>+49(0)251/396 371-0</ows:Voice>
            </ows:Phone>
            <ows:Address>
               <ows:DeliveryPoint>
                  Martin-Luther-King-Weg 24<
               </ows:DeliveryPoint>
               <ows:City>Münster</ows:City>
               <ows:AdministrativeArea>
                  North Rhine-Westphalia
               </ows:AdministrativeArea>
               <ows:PostalCode>48155</ows:PostalCode>
               <ows:Country>Germany</ows:Country>
               <ows:ElectronicMailAddress>
                  info@52north.org
               </ows:ElectronicMailAddress>
            </ows:Address>
         </ows:ContactInfo>
      </ows:ServiceContact>
   </ows:ServiceProvider>
   <ows:OperationsMetadata>
      <ows:Operation name="DescribeFeatureType">
         <ows:DCP>
            <ows:HTTP>
               <ows:Get xlink:href="http://pilot.52north.org/52n-wfs-
                                    proxy-webapp/service?">
                  <ows:Constraint name="Content-Type">
                     <ows:AllowedValues>
                        <ows:Value>application/x-kvp</ows:Value>
                     </ows:AllowedValues>
                  </ows:Constraint>
               </ows:Get>
            </ows:HTTP>
         </ows:DCP>
         <ows:Parameter name="outputFormat">
            <ows:AllowedValues>
               <ows:Value>application/gml+xml; version=3.2</ows:Value>
               <ows:Value>application/om+xml; version=2.0</ows:Value>
               <ows:Value>
                  application/samplingspatial+xml; version=2.0
               </ows:Value>
               <ows:Value>text/xml</ows:Value>
               <ows:Value>text/xml; subtype="gml/3.2"</ows:Value>
            </ows:AllowedValues>
         </ows:Parameter>
      </ows:Operation>
      <ows:Operation name="DescribeStoredQueries">
         <ows:DCP>
            <ows:HTTP>
               <ows:Get xlink:href="http://pilot.52north.org/52n-wfs-
                                    proxy-webapp/service?">
                  <ows:Constraint name="Content-Type">
                  <ows:AllowedValues>
                     <ows:Value>application/x-kvp</ows:Value>
                  </ows:AllowedValues>
               </ows:Constraint>
            </ows:Get>
         </ows:HTTP>
      </ows:DCP>
   </ows:Operation>
   <ows:Operation name="GetCapabilities">
      <ows:DCP>
         <ows:HTTP>
            <ows:Get xlink:href="http://pilot.52north.org/52n-wfs-
                                 proxy-webapp/service?">
               <ows:Constraint name="Content-Type">
                  <ows:AllowedValues>
                     <ows:Value>application/x-kvp</ows:Value>
                  </ows:AllowedValues>
               </ows:Constraint>
            </ows:Get>
         </ows:HTTP>
      </ows:DCP>
      <ows:Parameter name="AcceptFormats">
         <ows:AllowedValues>
            <ows:Value>application/xml</ows:Value>
         </ows:AllowedValues>
      </ows:Parameter>
      <ows:Parameter name="AcceptVersions">
         <ows:AllowedValues>
            <ows:Value>2.0.0</ows:Value>
         </ows:AllowedValues>
      </ows:Parameter>
      <ows:Parameter name="Sections">
         <ows:AllowedValues>
            <ows:Value>All</ows:Value>
            <ows:Value>FeatureTypeList</ows:Value>
            <ows:Value>Filter_Capabilities</ows:Value>
            <ows:Value>OperationsMetadata</ows:Value>
            <ows:Value>ServiceIdentification</ows:Value>
            <ows:Value>ServiceProvider</ows:Value>
         </ows:AllowedValues>
      </ows:Parameter>
      <ows:Parameter name="updateSequence">
         <ows:AnyValue/>
      </ows:Parameter>
   </ows:Operation>
   <ows:Operation name="GetFeature">
      <ows:DCP>
         <ows:HTTP>
            <ows:Get xlink:href="http://pilot.52north.org/52n-wfs-
                                 proxy-webapp/service?">
               <ows:Constraint name="Content-Type">
                  <ows:AllowedValues>
                     <ows:Value>application/x-kvp</ows:Value>
                  </ows:AllowedValues>
```

```
                </ows:Constraint>
              </ows:Get>
            </ows:HTTP>
          </ows:DCP>
          <ows:Parameter name="outputFormat">
            <ows:AllowedValues>
              <ows:Value>application/gml+xml; version=3.2</ows:Value>
              <ows:Value>application/om+xml; version=2.0</ows:Value>
              <ows:Value>
                application/samplingspatial+xml; version=2.0
              </ows:Value>
              <ows:Value>text/xml</ows:Value>
              <ows:Value>text/xml; subtype="gml/3.2"</ows:Value>
              <ows:Value>text/xml;charset=UTF-8</ows:Value>
            </ows:AllowedValues>
          </ows:Parameter>
        </ows:Operation>
        <ows:Operation name="GetPropertyValue">
          <ows:DCP>
            <ows:HTTP>
              <ows:Get xlink:href="http://pilot.52north.org/52n-wfs-
                                   proxy-webapp/service?">
                <ows:Constraint name="Content-Type">
                  <ows:AllowedValues>
                    <ows:Value>application/x-kvp</ows:Value>
                  </ows:AllowedValues>
                </ows:Constraint>
              </ows:Get>
            </ows:HTTP>
          </ows:DCP>
        </ows:Operation>
        <ows:Parameter name="service">
          <ows:AllowedValues>
            <ows:Value>WFS</ows:Value>
          </ows:AllowedValues>
        </ows:Parameter>
        <ows:Parameter name="version">
          <ows:AllowedValues>
            <ows:Value>2.0.0</ows:Value>
          </ows:AllowedValues>
        </ows:Parameter>
      </ows:OperationsMetadata>
      <wfs:FeatureTypeList>
        <wfs:FeatureType>
          <wfs:Name xmlns:ns="http://www.opengis.net/om/2.0">
            ns:OM_Observation
          </wfs:Name>
          <wfs:Title>Observations</wfs:Title>
          <wfs:Abstract>OWS-10 observation for VGI</wfs:Abstract>
          <ows:Keywords>
            <ows:Keyword>observations</ows:Keyword>
          </ows:Keywords>
          <wfs:DefaultCRS>urn:ogc:def:crs:EPSG::4326</wfs:DefaultCRS>
          <wfs:OutputFormats>
            <wfs:Format>application/gml+xml; version=3.2</wfs:Format>
        </wfs:OutputFormats>
        <ows:WGS84BoundingBox>
            <ows:LowerCorner>
                34.6890602111816 -86.59235
            </ows:LowerCorner>
            <ows:UpperCorner>
                34.70061 -86.57660675048828
            </ows:UpperCorner>
        </ows:WGS84BoundingBox>
    </wfs:FeatureType>
    <wfs:FeatureType>
        <wfs:Name xmlns:pil="http://pilot.52north.org">
            pil:PilotFeature
        </wfs:Name>
        <wfs:Title>PilotFeatures for IMIS-IoT</wfs:Title>
        <wfs:Abstract/>
        <ows:Keywords>
            <ows:Keyword>pilot features</ows:Keyword>
        </ows:Keywords>
        <wfs:DefaultCRS>urn:ogc:def:crs:EPSG::4326</wfs:DefaultCRS>
        <wfs:OutputFormats>
            <wfs:Format>application/gml+xml; version=3.2</wfs:Format>
        </wfs:OutputFormats>
        <ows:WGS84BoundingBox>
            <ows:LowerCorner>
                34.6890602111816 -86.59235
            </ows:LowerCorner>
            <ows:UpperCorner>
                34.70061 -86.57660675048828
            </ows:UpperCorner>
        </ows:WGS84BoundingBox>
    </wfs:FeatureType>
    <wfs:FeatureType>
        <wfs:Name xmlns:ns="http://www.opengis.net/
                             samplingSpatial/2.0">
            ns:SF_SpatialSamplingFeature
        </wfs:Name>
        <wfs:Title>Features for IMIS-IoT</wfs:Title>
        <wfs:Abstract/>
        <ows:Keywords>
            <ows:Keyword>features</ows:Keyword>
        </ows:Keywords>
        <wfs:DefaultCRS>urn:ogc:def:crs:EPSG::4326</wfs:DefaultCRS>
        <wfs:OutputFormats>
            <wfs:Format>application/gml+xml; version=3.2</wfs:Format>
        </wfs:OutputFormats>
        <ows:WGS84BoundingBox>
            <ows:LowerCorner>
                34.6890602111816 -86.59235
            </ows:LowerCorner>
            <ows:UpperCorner>
                34.70061 -86.57660675048828
            </ows:UpperCorner>
        </ows:WGS84BoundingBox>
```

```
      </wfs:FeatureType>
   </wfs:FeatureTypeList>
   <fes:Filter_Capabilities>
      <fes:Conformance>
         <fes:Constraint name="ImplementsQuery">
            <ows:NoValues/>
            <ows:DefaultValue>false</ows:DefaultValue>
         </fes:Constraint>
         <fes:Constraint name="ImplementsAdHocQuery">
            <ows:NoValues/>
            <ows:DefaultValue>true</ows:DefaultValue>
         </fes:Constraint>
         <fes:Constraint name="ImplementsFunctions">
            <ows:NoValues/>
            <ows:DefaultValue>false</ows:DefaultValue>
         </fes:Constraint>
         <fes:Constraint name="ImplementsResourceld">
            <ows:NoValues/>
            <ows:DefaultValue>false</ows:DefaultValue>
         </fes:Constraint>
         <fes:Constraint name="ImplementsMinStandardFilter">
            <ows:NoValues/>
            <ows:DefaultValue>false</ows:DefaultValue>
         </fes:Constraint>
         <fes:Constraint name="ImplementsStandardFilter">
            <ows:NoValues/>
            <ows:DefaultValue>false</ows:DefaultValue>
         </fes:Constraint>
         <fes:Constraint name="ImplementsMinSpatialFilter">
            <ows:NoValues/>
            <ows:DefaultValue>true</ows:DefaultValue>
         </fes:Constraint>
         <fes:Constraint name="ImplementsSpatialFilter">
            <ows:NoValues/>
            <ows:DefaultValue>false</ows:DefaultValue>
         </fes:Constraint>
         <fes:Constraint name="ImplementsMinTemporalFilter">
            <ows:NoValues/>
            <ows:DefaultValue>true</ows:DefaultValue>
         </fes:Constraint>
         <fes:Constraint name="ImplementsTemporalFilter">
            <ows:NoValues/>
            <ows:DefaultValue>false</ows:DefaultValue>
         </fes:Constraint>
         <fes:Constraint name="ImplementsVersionNav">
            <ows:NoValues/>
            <ows:DefaultValue>false</ows:DefaultValue>
         </fes:Constraint>
         <fes:Constraint name="ImplementsSorting">
            <ows:NoValues/>
            <ows:DefaultValue>false</ows:DefaultValue>
         </fes:Constraint>
         <fes:Constraint name="ImplementsExtendedOperators">
            <ows:NoValues/>
            <ows:DefaultValue>false</ows:DefaultValue>
         </fes:Constraint>
         <fes:Constraint name="ImplementsMinimumXPath">
            <ows:NoValues/>
            <ows:DefaultValue>false</ows:DefaultValue>
         </fes:Constraint>
         <fes:Constraint name="ImplementsSchemaElementFunc">
            <ows:NoValues/>
            <ows:DefaultValue>false</ows:DefaultValue>
         </fes:Constraint>
      </fes:Conformance>
      <fes:Scalar_Capabilities>
         <fes:ComparisonOperators>
            <fes:ComparisonOperator name="PropertyIsEqualTo"/>
         </fes:ComparisonOperators>
      </fes:Scalar_Capabilities>
      <fes:Spatial_Capabilities>
         <fes:GeometryOperands>
            <fes:GeometryOperand xmlns:ns="http://www.opengis.
                                 net/gml/3.2"
                                 name="ns:Envelope"/>
         </fes:GeometryOperands>
         <fes:SpatialOperators>
            <fes:SpatialOperator name="BBOX">
               <fes:GeometryOperands>
                  <fes:GeometryOperand xmlns:ns="http://www.opengis.
                                       net/gml/3.2"
                                       name="ns:Envelope"/>
               </fes:GeometryOperands>
            </fes:SpatialOperator>
         </fes:SpatialOperators>
      </fes:Spatial_Capabilities>
      <fes:Temporal_Capabilities>
         <fes:TemporalOperands>
            <fes:TemporalOperand xmlns:ns="http://www.opengis.
                                 net/gml/3.2"
                                 name="ns:TimeInstant"/>
            <fes:TemporalOperand xmlns:ns="http://www.opengis.
                                 net/gml/3.2"
                                 name="ns:TimePeriod"/>
         </fes:TemporalOperands>
         <fes:TemporalOperators>
            <fes:TemporalOperator name="During">
               <fes:TemporalOperands>
                  <fes:TemporalOperand xmlns:ns="http://www.opengis.
                                       net/gml/3.2"
                                       name="ns:TimePeriod"/>
               </fes:TemporalOperands>
            </fes:TemporalOperator>
            <fes:TemporalOperator name="TEquals">
               <fes:TemporalOperands>
                  <fes:TemporalOperand xmlns:ns="http://www.opengis.
                                       net/gml/3.2"
                                       name="ns:TimeInstant"/>
               </fes:TemporalOperands>
            </fes:TemporalOperator>
         </fes:TemporalOperators>
      </fes:Temporal_Capabilities>
   </fes:Filter_Capabilities>
</wfs:WFS_Capabilities>
```

A.3 Observations and Measurements (O&M) Chemical Measurement Example

```
<om:OM_Observation gml:id="o_AC03F21B313075894C23A471044C896999FE65B7">
   <gml:description>
      test description for this observation
   </gml:description>
   <gml:identifier codeSpace="http://www.opengis.net/def/nil/OGC/0/
                            unknown">
      Chemical1/1452484545014
   </gml:identifier>
   <om:type xlink:href="http://www.opengis.net/def/observationType/OGC-
                        OM/2.0/OM_Measurement"/>
   <om:phenomenonTime>
      <gml:TimeInstant gml:id="phenomenonTime_100607">
         <gml:timePosition>2016-01-11T03:55:45.014Z</gml:timePosition>
      </gml:TimeInstant>
   </om:phenomenonTime>
   <om:resultTime xlink:href="#phenomenonTime_100607"/>
   <om:procedure xlink:href="Chemical1"/>
   <om:observedProperty xlink:href="Gasoline"/>
   <om:featureOfInterest xlink:href="MemorialP" xlink:title="UM"/>
   <om:result xmlns:ns="http://www.opengis.net/gml/3.2" uom="ppm"
              xsi:type="ns:MeasureType">
      50.7
   </om:result>
</om:OM_Observation>
```

A.4 O&M Pedestrian Count Example

```
<om:OM_Observation gml:id="o_8C8E602E4313E1567E6CFA19E507952CBE53B7F5">
   <gml:description>
      test description for this observation
   </gml:description>
   <gml:identifier codeSpace="http://www.opengis.net/def/nil/OGC/0/
                            unknown">
      PedestrianCounting1/1452484647215
   </gml:identifier>
   <om:type xlink:href="http://www.opengis.net/def/observationType/OGC-
                        OM/2.0/OM_Measurement"/>
   <om:phenomenonTime>
      <gml:TimeInstant gml:id="phenomenonTime_101069">
         <gml:timePosition>2016-01-11T03:57:27.215Z</gml:timePosition>
      </gml:TimeInstant>
   </om:phenomenonTime>
   <om:resultTime xlink:href="#phenomenonTime_101069"/>
   <om:procedure xlink:href="PedestrianCounting1"/>
   <om:observedProperty xlink:href="PeopleCount"/>
   <om:featureOfInterest xlink:href="intersection1" xlink:title="UM"/>
   <om:result xmlns:ns="http://www.opengis.net/gml/3.2"
              uom="PeopleCount" xsi:type="ns:MeasureType">
      123.0
   </om:result>
</om:OM_Observation>
```

Bibliography

[1] Botts, M. and A. Robin (2014). OGC Implementation Specification: Sensor Model Language (SensorML) 2.0.0 (12-000). Wayland, MA, USA, Open Geospatial Consortium Inc.

[2] Bröring, A., C. Stasch, et al. (2012). OGC Implementation Specification: Sensor Observation Service (SOS) 2.0 (12-006). Wayland, MA, USA, Open Geospatial Consortium Inc.

[3] Cox, S. (2011). OGC Implementation Specification: Observations and Measurements (O&M) - XML Implementation 2.0 (10-025r1). Wayland, MA, USA, Open Geospatial Consortium Inc.

[4] de la Beaujardiere, J. (2006). OGC Implementation Specification: Web Map Service (WMS) 1.3.0 (06-042). Wayland, MA, USA, Open Geospatial Consortium Inc.

[5] Echterhoff, J. (2011). OGC Implementation Specification: SWE Service Model 2.0.0 (09-001). Wayland, MA, USA, Open Geospatial Consortium Inc.

[6] ISO TC 211 (2011). ISO 19156:2011 - Geographic information -- Observations and measurements - International Standard. Geneva, Switzerland, International Organization for Standardization.

[7] Müller, M. and B. Proß (2015). OGC Implementation Specification: Web Processing Service (WPS) 2.0.0 (14-065). Wayland, MA, USA, Open Geospatial Consortium Inc.

[8] Nebert, D., A. Whiteside, et al. (2007). OGC Implementation Specification: Catalog Services Specification 2.0.2 Corrigendum 2 (07-006r1). Wayland, MA, USA, Open Geospatial Consortium Inc.

[9] Portele, C. (2012). OGC Implementation Specification: Geography Markup Language (GML) - Extended Schemas and Encoding Rules (OGC 10-129r1). Wayland, MA, USA, Open Geospatial Consortium Inc.

[10] Robin, A. (2011). OGC Implementation Specification: SWE Common Data Model 2.0.0 (08-094r1). Wayland, MA, USA, Open Geospatial Consortium Inc.

[11] Vretanos, P. A. (2014). OGC Implementation Specification: Web Feature Service (WFS) 2.0.2 (09-025r2). Wayland, MA, USA, Open Geospatial Consortium Inc.

[12] Whiteside, A. and J. Greenwood (2010). OGC Implementation Specification: Web Services Common 2.0.0 (06-121r9). Wayland, MA, USA, Open Geospatial Consortium Inc.