CODE SPRINT

RENNES 20>22 NOV

2 THÈMES :

SWE, SOS, IOT

et

Documentation

Laboratoire régional d'innovation publique
5 rue Martenot, 35000 Rennes

www.georchestra.org/blog

psc.georchestra@gmail.com

@georchestra

Proposé par geOrchestra

Avec le soutien de AGRO CAMPUS OUEST, METROPOLE Rennes, Région BRETAGNE

Compte rendu Code Sprint SWE/SOS

Données

On utilise les données SQL de comptage routier fournies par Rennes Métropole:
https://github.com/fvanderbiest/mapserver-sos-experiment/blob/master/db.sql

Agrocampus a des données volumineuses sur plusieurs dizaines d'années.

On identifie également une liste de serveurs SOS fonctionnels disponibles sur internet :

GrandLyon Bruit	https://download.data.grandlyon.com/sos/bruit?service=SOS&request=GetCapabilities	istsos
GrandLyon Velov	https://download.data.grandlyon.com/sos/velov?service=SOS&request=GetCapabilities	istsos
ORE agrhys	http://agrhys.fr/istsos/agrhys?service=SOS&request=GetCapabilities	istsos
ORE AMMA-CATCH / SOS AMMA-CATCH DB	http://bd.amma-catch.org/amma-catchWS2/WS/sos/default?service=SOS&request=GetCapabilities	Constellation
Test SOS Service	http://sensiasoft.net:8181/sensorhub/sos?service=SOS&acceptVersions=2.0.0&request=GetCapabilities	OpenSensorHub
Test SOS Service	http://sensorweb.demo.52north.org/52n-sos-webapp/service?service=SOS&request=GetCapabilities	52 North
Test SOS Service	http://sos.geosas.fr/sos?service=SOS&version=2.0&request=GetCapabilities	OpenSensorHub

Voir aussi
http://rdata-grandlyon.readthedocs.io/fr/latest/services.html#services-sos-donnees-de-capteurs

Standards

L'ensemble des normes OGC relatives à SWE (et plus largement) peuvent être téléchargées très simplement en clonant le dépôt suivant: https://github.com/georchestra/ogc-standards

Question sur l'état actuel de la norme SensorThings. Officiel ? Serveurs existants ?
https://pg.sensorup.com/sign-in.html
https://www.gostserver.xyz/ etc

On questionne la pertinence de SOS pour le besoin exprimé de diffusion de données issues de capteurs, avec potentiellement une surcouche (API) permettant de réaliser des traitements, ex agrégations, simplification, etc

On comprend l'intérêt des requêtes getResult et GetResultTemplate, qui permettent de minimiser la taille des flux d'information échangés.

On découvre collectivement les concepts clés de SWE : offering, observation, featureOfInterest, procedure, observedProperty.

A posteriori, on trouve sur https://52north.github.io/sensor-web-tutorial/ des ressources :

Term	Definition
Feature of Interest (FoI)	Georeferenced abstraction of a real world feature carrying the observed property
Observed Property / Phenomenon	The measurable property that is observed by a sensor; Examples are temperature, precipitation quantity, humidity
Procedure	The process that performs the measurement; Can be a physical sensor (a hardware device) or a virtual sensor (e.g. software that produces computations or simulations as measurement)
Offering	groups collections of observations produced by one procedure and provides additional metadata of the observations
Result	The value of the measurement of an observed property
Phenomenon Time	Time associated to the observation's result
Result Time	Time when the result has been produced
Valid time	Time period for which the observations result is valid
Observation	Composition of the above properties; An *observation* is captured by a *procedure* that measures an *observable property* of a *feature of interest* at a certain *phenomenon time* and stores the measurement as the *result*.

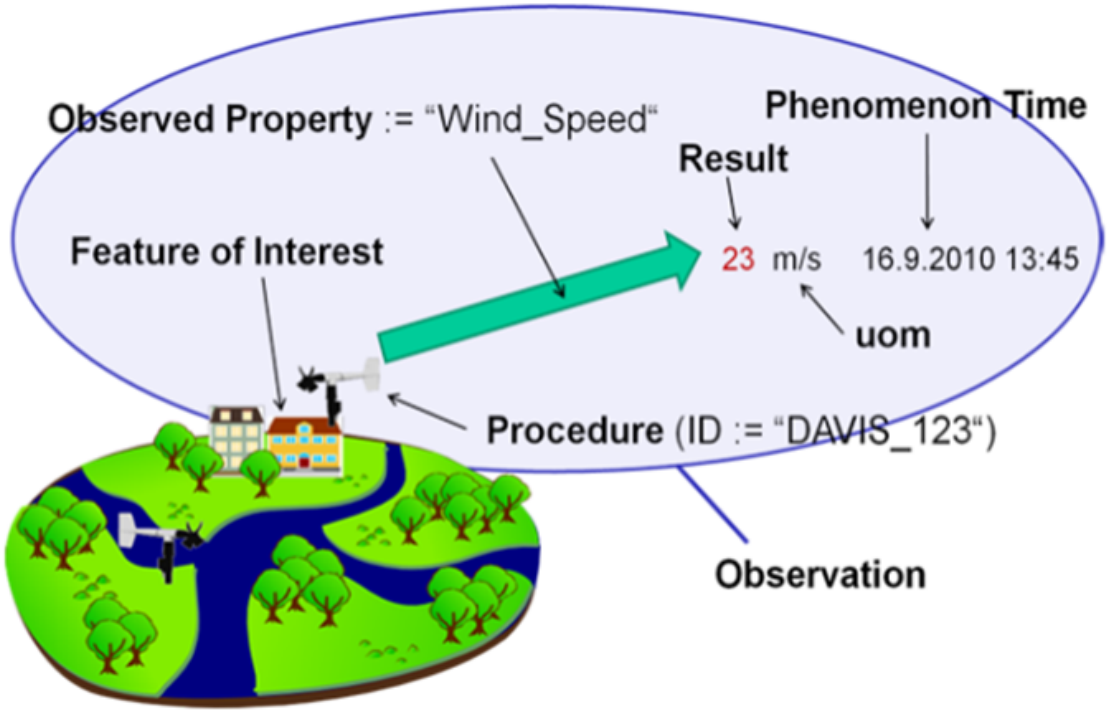
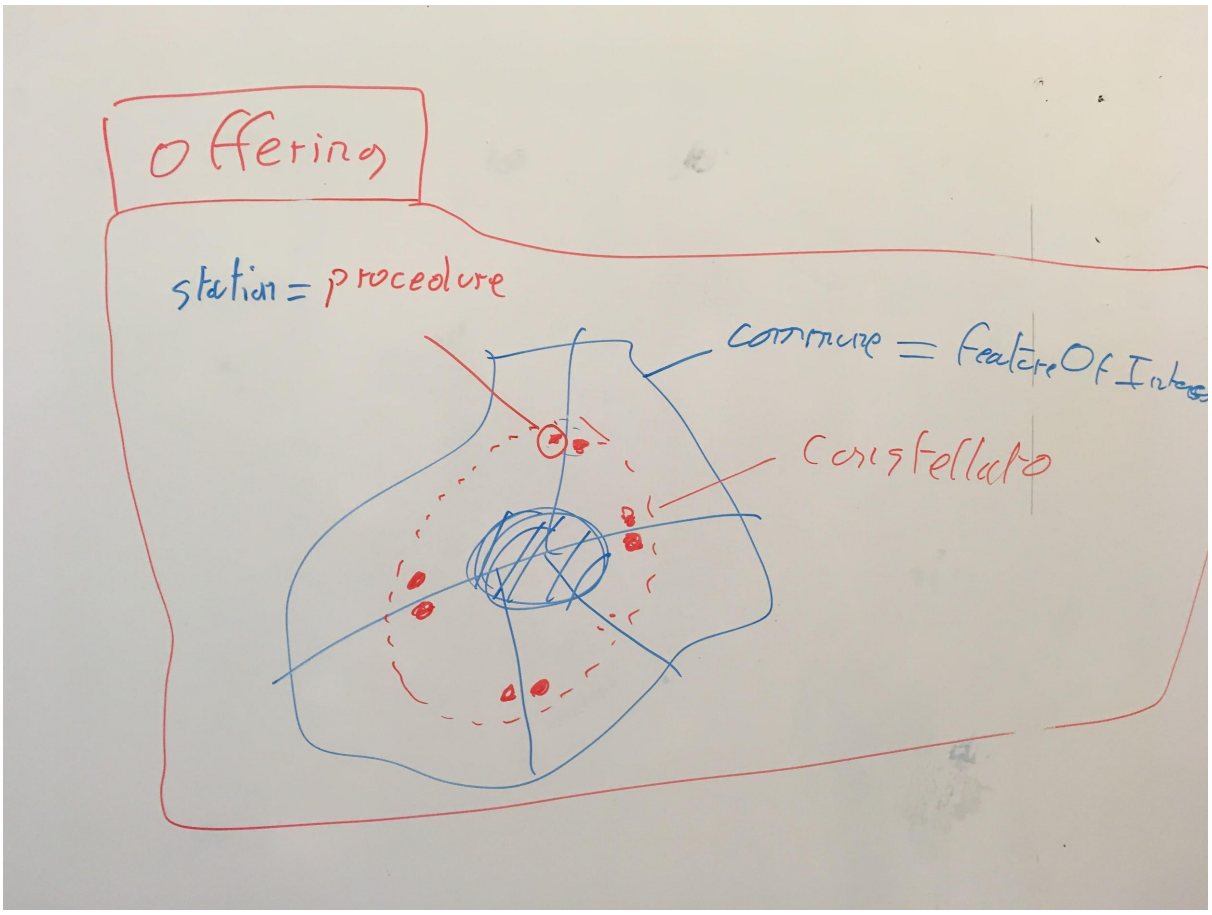
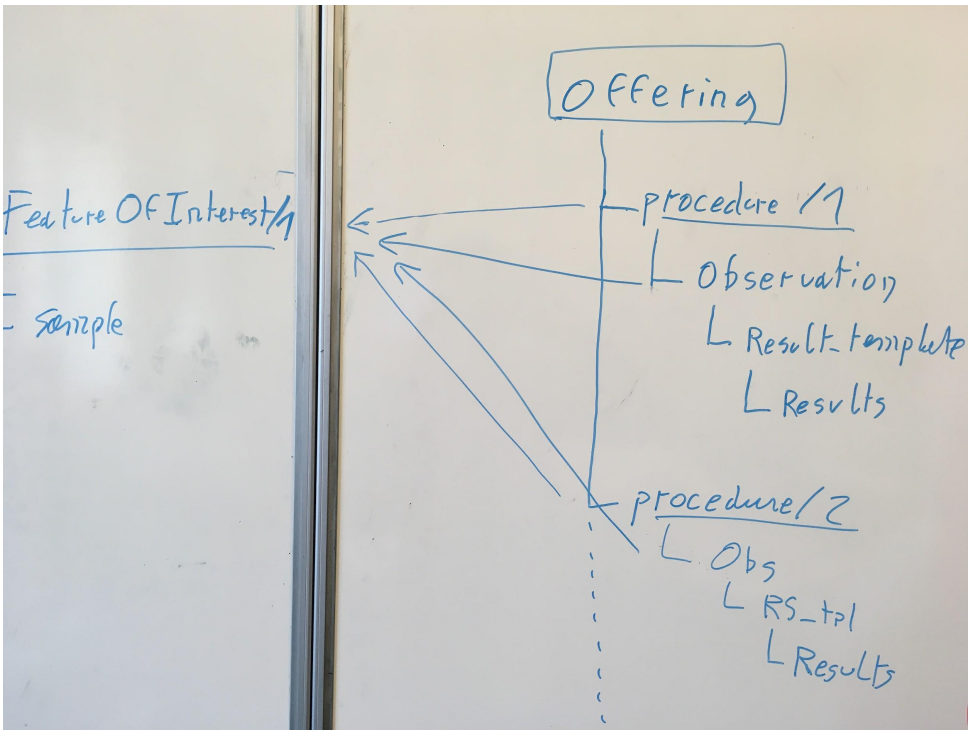
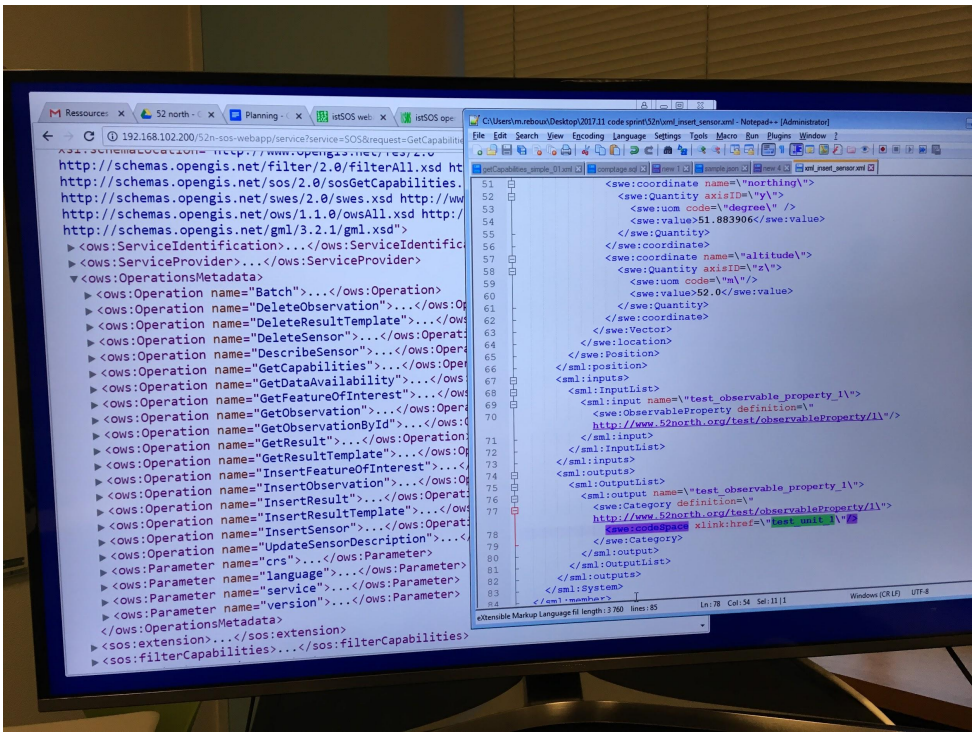


Figure 1 : Les termes principaux du modèle O&M 2.0 (voir http://www.ogcnetwork.net/sos_2_0/tutorial/om)




Offering
FeatureOfInterest/1
sample
procedure/1
Observation
Result_template
Results
procedure/2
Obs
RS_tpl
Results


http://schemas.opengis.net/filter/2.0/filterAll.xsd
http://schemas.opengis.net/sos/2.0/sosGetCapabilities.
http://schemas.opengis.net/swes/2.0/swes.xsd
http://schemas.opengis.net/ows/1.1.0/owsAll.xsd
http://schemas.opengis.net/gml/3.2.1/gml.xsd
<ows:ServiceIdentification>...</ows:ServiceIdentification>
<ows:ServiceProvider>...</ows:ServiceProvider>
<ows:OperationsMetadata>
<ows:Operation name="Batch">...</ows:Operation>
<ows:Operation name="DeleteObservation">...</ows:Operation>
<ows:Operation name="DeleteResultTemplate">...</ows:Operation>
<ows:Operation name="DeleteSensor">...</ows:Operation>
<ows:Operation name="DescribeSensor">...</ows:Operation>
<ows:Operation name="GetCapabilities">...</ows:Operation>
<ows:Operation name="GetDataAvailability">...</ows:Operation>
<ows:Operation name="GetFeatureOfInterest">...</ows:Operation>
<ows:Operation name="GetObservation">...</ows:Operation>
<ows:Operation name="GetObservationById">...</ows:Operation>
<ows:Operation name="GetResult">...</ows:Operation>
<ows:Operation name="GetResultTemplate">...</ows:Operation>
<ows:Operation name="InsertFeatureOfInterest">...</ows:Operation>
<ows:Operation name="InsertObservation">...</ows:Operation>
<ows:Operation name="InsertResult">...</ows:Operation>
<ows:Operation name="InsertResultTemplate">...</ows:Operation>
<ows:Operation name="InsertSensor">...</ows:Operation>
<ows:Operation name="UpdateSensorDescription">...</ows:Operation>
<ows:Parameter name="crs">...</ows:Parameter>
<ows:Parameter name="language">...</ows:Parameter>
<ows:Parameter name="service">...</ows:Parameter>
<ows:Parameter name="version">...</ows:Parameter>
</ows:OperationsMetadata>
<sos:extension>...</sos:extension>
<sos:filterCapabilities>...</sos:filterCapabilities>
<swe:coordinate name="northing">
<swe:Quantity axisID="y">
<swe:uom code="degree" />
<swe:value>51.883906</swe:value>
<swe:coordinate name="altitude">
<swe:Quantity axisID="z">
<swe:uom code="m"/>
<swe:value>52.0</swe:value>
<sml:inputs>
<sml:input name="test_observable_property_1">
<swe:ObservableProperty definition="
http://www.52north.org/test/observableProperty/1"/>
<sml:outputs>
<sml:output name="test_observable_property_1">
<swe:Category definition="
http://www.52north.org/test/observableProperty/1">
<swe:codeSpace xlink:href="test_unit_1"/>
<sml:System>

Logiciels / Outils

Côté serveur

On identifie rapidement plusieurs outils logiciels permettant de servir les données en SOS:

- MapServer
- OpenSensorHub aka OSH
- 52North SOS
- ISTSOS
- Deegree (?)

La question de la connection aux sources de données (capteurs ou bases) est abordée. Il semblerait que la connection directe aux capteurs soit assez peu répandue à l'heure actuelle (même si mise en avant par OSH).

Par ailleurs, on constate dans certains cas (istsos, 52North), que l'application vient avec son propre schéma applicatif (52North créée une 50aine de tables), qu'il faut alimenter via divers scripts d'insertion ou ETL, de prise en main par forcément facile.

MapServer

Une expérience est menée autour de MapServer, pour servir les données de RM en SOS, et les visualiser sur une carte.

Celle-ci est poussée sur https://github.com/fvanderbiest/mapserver-sos-experiment et utilise docker afin de garantir la reproductibilité (voir le README).

Liens intéressants:

- http://mapserver.org/fr/ogc/sos_server.html
- http://mapserver.org/development/rfc/ms-rfc-13.html#rfc13

Conclusions de l'expérience:

- On a monté un serveur SOS 1.0.0 fonctionnel à partir des données fournies, assez rapidement (quelques heures), qui supporte GetObservation.
- Le service est très limité :
    - uniquement SOS 1.0.0
    - ne supporte que GetCapabilities, DescribeSensor, DescribeObservationType, GetObservation. Il manque cruellement getFeatureOfInterest.
    - Une couche = un capteur, même si un patch (cf https://github.com/fvanderbiest/mapserver-sos-experiment/blob/2c0a6bf7b25884a885c228182c9738ebb3bd93e4/sos.map#L33 et https://github.com/mapserver/mapserver/issues/2050) permettrait en théorie de servir les données de tous les capteurs d'un coup. Malheureusement nous n'avons pas réussi à le faire fonctionner.
- Plus généralement, le support de SOS dans MapServer semble ancien, peu maintenu, assez anecdotique.

OpenSensorHub

On installe très rapidement OSH à partir du binaire fourni sur https://github.com/opensensorhub/osh-core/releases, et on accède facilement à une démonstration et une interface d'administration classieuse.

La documentation fournie par http://docs.opensensorhub.org/ est intéressante, mais insuffisante sur certains points, notamment la connection aux données.

En veille sur SensorThings...

Possibilité de développer des connecteurs vers API dédiée (ex: sensorhub-driver-waterdata vers API USGS). Enjeu : pas de stockage propre au service SOS.

Le support PostGreSQL est manquant pour le moment. Seul ElasticSearch semble supporté actuellement. Nous n'irons donc pas plus loin dans le cadre du code sprint.

Liens pertinents :

- https://opensensorhub.org/
- https://github.com/opensensorhub/osh-core

52 North

Logiciel en java a installer dans un container de servlet (tomcat, jetty) avec un postgis pour la BDD.

52 north v 4.4.2
config lors de l'install :

- pas support time series
- pas support multilingual

Sinon on passe de 54 tables à 67 tables.

52 North SOS Data model: https://wiki.52north.org/SensorWeb/SosDataModeling

Liens:

- Un importer: https://github.com/52North/sos-importer à tester ?
- https://github.com/52North/helgoland une appli d'exploration visuelle et d'analyse des données de capteurs. URL de démo ?

La journée du lundi a consisté à confronter les concepts SWE / SOS à l'implémentation par ce logiciel. Ceci afin de dresser une méthodologie de chargement de données en utilisant l'API.
Cela donne :

1. **InsertFeatureOfInterest :** (world) > France > commune
2. **InsertSensor (table procedure)** : The InsertSensor operation allows a client to register a new sensor system at the SOS. The operation is part of the Sensor Insertion requirements class. Sensor observations can only be inserted for sensors that have first been inserted in the SOS.

3. **InsertObservation** : The InsertObservation operation allows clients to insert new observations for a registered sensor system.
4. **InsertResultTemplate**, for inserting a result template into a SOS server that describes the structure of the values of a InsertResult of GetResult request.
5. **InsertResult**, for uploading raw values accordingly to the structure and encoding defined in the InsertResultTemplate request

L'interface client nous a permis de nous approprier les codes / requêtes d'exemple. A ce sujet, nous recommandons l'emploi du XML "pur" (POX) qui est plus facilement lisible. Eviter le JSON.
http://192.168.102.200/52n-sos-webapp/client

Nous avons réussi les 4 premières étapes.
L'apprentissage a été un peu laborieux.
Le hack de la base de données envisagé en premier lieu a été abandonné car trop complexe sur un temps court car la BD est très relationnelle.

Un FeatureOfInterest doit toujours faire référence à un FeatureOfInterest parent (sampledFeature). La base est initialisée avec un un objet "world" par défaut auquel il faut rattacher les FeatureOfInterest de premier niveau suivant.

ISTSOS

Logiciel en python largement utilisé (grandlyon, agrhys, etc), dont les utilisateurs rapportent la simplicité d'alimentation en données, mais également la limitation quand on traite de gros volumes de données.

Etant donné que plusieurs personnes dans la salle ont eu des expériences avec ce logiciel, nous ne l'utiliserons pas dans le cadre du code sprint.

Alimentation de la base du service SOS en 2 temps :
- Création des offering, FOI, procedure … par une interface web admin,
- Import des observations par fichier CSV + script fourni : csv2istsos.py

Communauté de développeurs réduite : 1 ?

Liens:
- http://www.istsos.org/

Deegree

Nous n'avons pas trouvé de support SOS au niveau de deegree. Peut etre avons nous mal cherché ?

Côté client

OpenLayers 2

Une expérimentation a été menée en 2010 et présentée au FOSS4G, il s'agit d'un client SOS avec OpenLayers2: http://2010.foss4g.org/presentations/2891.pdf

Celle-ci a amené au développement d'une suite de classes permettant à OpenLayers2 de travailler avec SOS.
Ceci est de **très bon augure** en vue de l'intégration de SOS dans le coeur de geOrchestra, notamment son visualiseur "**mapfishapp**", via le développement d'un addon dédié.

On trouve notamment dans OpenLayers2 :

- Un protocole pour interagir avec le service - https://github.com/openlayers/ol2/blob/master/lib/OpenLayers/Protocol/SOS/v1_0_0.js
- Un format pour lire les capacités du service (GetCapabilities) https://github.com/openlayers/ol2/blob/master/lib/OpenLayers/Format/SOSCapabilities/v1_0_0.js
- Un format pour obtenir la localisation des stations de mesure (sachant qu'une station peut possèder un ou plusieurs capteurs) -> GetFeatureOfInterest : https://github.com/openlayers/ol2/blob/master/lib/OpenLayers/Format/SOSGetFeatureOfInterest.js
- Un format pour obtenir les valeurs mesurées par le capteur (GetObservations) : https://github.com/openlayers/ol2/blob/master/lib/OpenLayers/Format/SOSGetObservation.js

On observe la même logique, présente dans tous les services OGC pour accéder à la donnée :

1. Le client lance une requête GetCapabilities, de manière à négocier la version du protocole d'échange et connaître les capacités en terme de requêtage mais également en terme de données proposées par le service
2. Le client lance une requête GetFeatureOfInterest pour identifier une station de mesure et ses capteurs
3. Le client lance une requête GetObservations pour obtenir les valeurs mesurées au niveau du capteur

A noter qu'il n'existe pas encore de support pour l'opération **DescribeSensor** dans OpenLayers 2.

Pendant le code sprint, on réutilise l'exemple OpenLayers traitant SOS :
http://dev.openlayers.org/releases/OpenLayers-2.13.1/examples/sos.html
… en essayant de le connecter soit au service MapServer dockerisé avec les données RM, soit au service SOS fourni par l'observatoire AMMA-CATCH (cf supra).

Dans le premier cas, l'échec intervient au moment où OL essaie de lancer une requête GetFeatureOfInterest, non supportée par MapServer.

Dans le second cas, l'échec intervient un peu plus tard, … .
Pour le lien avec Amma-catch, on arrive à récupérer les Observations, mais pas les mesures contenues dans les flux.

En essayant de se connecter au serveur 52North, on reste bloqué sur la version 2.0.0 qui n'est pas compatible avec cette partie d'OpenLayer. Même en forçant l'url pour utiliser la version 1.0.0 de SOS dans l'url d'appel, la réponse se fait en 2.0.0. Il faudrait essayer de rajouter un proxy devant 52N pour rerouter tous les appels en sos 1.0.0

Cette solution ne fonctionne que pour des services SOS en version 1.0.0

OpenLayers 3

- Un prototype ex ici par geomatys:

https://vimeo.com/161031565

Mais rien de bien solide pour OL3 à l'heure actuelle.

- Un document type tutoriel pour afficher des données istSOS :

http://istsos.org/en/trunk/doc/ws_mapping.html

52North/sos-js

https://github.com/52north/sos-js built on top of the **OpenLayers 2** SOS support
Assez léger.

OpenSensorHub

Client très complet pour interagir avec des services SOS et visualiser les données
https://github.com/opensensorhub/osh-js

On note surtout la possibilité

- d'utiliser des websockets pour garder la communication ouverte entre client et serveur.
- de créer des dashboards pour visualiser l'état des capteurs
- d'interagir avec des cartes.

Exemple de réalisation:
http://opensensorhub.github.io/osh-js/Demos/VideoDisplay/
http://opensensorhub.github.io/osh-js/Demos-old/
http://opensensorhub.github.io/osh-js/Showcase/

Affiche les données streamées par:
http://sensiasoft.net:8181/sensorhub/sos?service=SOS&version=2.0&request=GetResult&offering=urn:android:device:060693280a28e015-sos&observedProperty=http://sensorml.com/ont/swe/property/Location&temporalFilter=phenomenonTime,2015-02-16T07:58:00Z/2015-02-16T08:09:00Z&replaySpeed=3&responseFormat=application/json

Exemple de code source permettant de consommer ce WS:
https://github.com/opensensorhub/osh-js/blob/f8bdc55966a6c810b44dd486088ced33c24e8db4/Demos/VideoDisplay/js/config-3.js#L24

Est en capacité d’interagir avec des cartes Cesium, Leaflet, OL3
Cf https://github.com/opensensorhub/osh-js/tree/master/Toolkit/src/osh/ui/view/map

52North/helgoland-js

Pas une lib mais un Viewer permettant de visualiser rapidement les données issues du server 52N. Viewer Open-Source, la partie Carto est basé sur Leaflet. L’ajout de provider est rapide et simple un peu comme l’ajout de couche.
https://github.com/52North/helgoland

Avec une démo disponible ici :
http://sensorweb.demo.52north.org/client/#/map

Et SensorThings ?

Le cœur du sujet du sprint était focalisé sur les standards SWE et SOS. Si du temps disponible s'était dégagé sur les 4 demi-journées nous aurions étudié ce nouveau standard de l'OGC (février 2016).

Un participant a toutefois réalisé quelques recherches et a fait une restitution rapide synthétisée ci-dessous.

De la page sur le site de l'OGC : http://www.opengeospatial.org/projects/groups/sweiotswg nous retenos surtout ce paragraphe :

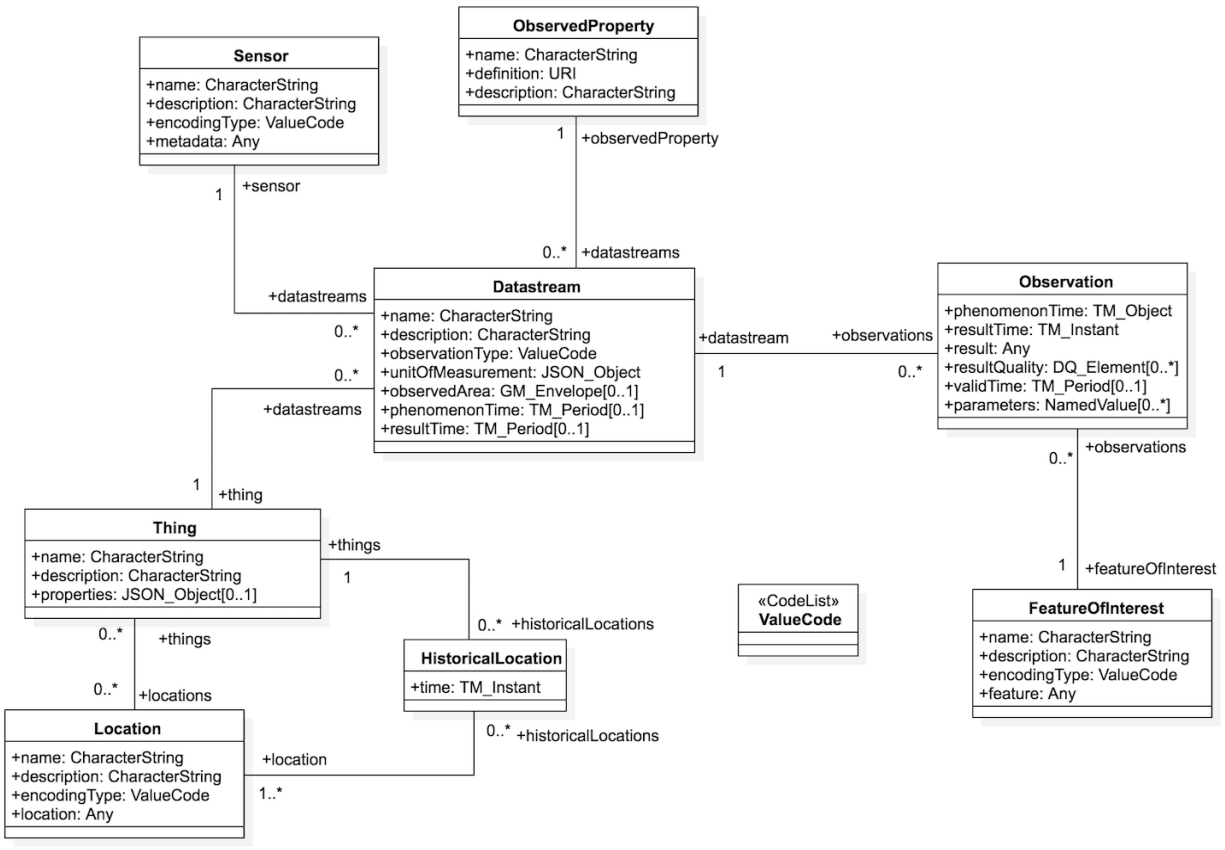
> *The Sensing Profile (Part I) provides the functions similar to the OGC Sensor Observation Service and the Tasking Profile (currently in progress) provides the functions similar to the OGC Sensor Planning Service.* ***The main difference between the SensorThings API and OGC SOS and SPS is that the SensorThings API is designed specifically for the resource-constrained IoT devices and the Web developers****. As a result the SensorThings API adopts the REST principle, the efficient JSON encoding, and the flexible OData protocol and URL conventions. The following UML diagram shows the entities of the SensorThings API.*

Et cette page https://github.com/opengeospatial/sensorthings apporte un complément :

> *The OGC SensorThings API can be considered as a lightweight SWE profile suited particularly for IoT applications. As a result,* ***the OGC SensorThings API is a new and efficient API based on the proven and widely implemented SWE standard framework.***

Tout le chemin d'apprentissage de SWE-SOS réalisé au cours du sprint n'a donc pas été un vain travail.

On voit au diagramme UML que ce standard SensorThings va à l'essentiel là où SWE-SOS a une démarche, disons le rapidement, plus scientifique.



Son attrait est que cela a été pensé d'emblée pour une utilisation web et plus *légère* que SOS.

Un essai d'installation d'un serveur SensorThings a été fait. Il s'agit du serveur de l'institut Franhofer : https://github.com/FraunhoferIOSB/SensorThingsServer
Il s'agit d'une application J2EE en java servie par un conteneur de servlet comme Tomcat ou Jetty.

Une fois le serveur installé, ce tutoriel a été suivi pour charger des données dans la base applicative :
https://sensorup.atlassian.net/wiki/spaces/SPS/blog/2015/11/07/17301576/Tutorial+SensorThings+API+101+-+How+to+upload+observation+data+to+SensorThings+API

Comme pour SWE, il faut d'abord créer un *sensor* puis des *observations*.

Un site - tutoriel intéressant a été trouvé : https://pg.sensorup.com Il permet de tester des requêtes REST vers une API pour tester sa syntaxe. Très visuel et didactique.