istSOS: Sensor Observation Service in Python

Massimiliano Cannata, Milan Antonovic

Scuola universitaria professionale della Svizzera italiana

SUPSI
Institute of Earth sciences

Land Planning

Hydrology

Hydrogeology

Geology

Geomatics
SOS service
(quick intro)
Sensor Observation Service

- It is a standard interface defined by the OGC for the management and distribution of observations. The current version is the 1.0 as defined by the “OGC 06-009r6” document.
SOS v1.0 requests

(core profile)
1. GetCapabilities
2. DescribeSensor
3. GetObservation

(transactional profile)
4. RegisterSensor
5. InsertObservation

(enhanced profile)
6. GetFeatureOfInterest
7. GetResult
8. GetObservationById
9. GetFeatureOfInterestTime
10. DescribeFeatureType
11. DescribeObservationType
12. DescribeResultModel
Data consumer

Consumer

GetCapabilities
Capabilities
DescribeSensor
Sensor description
GetObservation
Observations

SOS service
Data producer

Producer

SOS service

RegisterSensor

SensorID

InsertObservation

ObservationID

InsertObservation

ObservationID
istSOS (technology)
istSOS

is the SOS implementation by the
Istituto scienze della Terra
(Institute of Earth sciences)

http://istgeo.ist.supsi.ch/software/istSOS
• This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
istSOS is entirely developed in Python and rely on Apache/ModPython, PostgreSQL/PostGIS and GDAL/OGR.
SOS has been implemented with a factory method as design pattern. This particular pattern allows the automatic instantiation of the required objects or functions depending on the request type.
1. **SOS request**
   - **istSOS**

2. **filter**
   - Converts SOS requests into Python objects

3. **responder**
   - It gathers required information or performs transactional operations

4. **istSOS DB**
5. **renderer**
   - Converts information in SOS response format

8. **SOS response**
istSOS (package)
package

`sosConfig.py` configuration file

`sos.py` Web interface

`istSOS` SOS library

`sos_schema.sql` PostGIS schema
1. Install dependencies
2. Install istSOS schema
3. Install istSOS libraries
4. Configure apache/mod_python
5. Configure istSOS
1. dependencies

- **Base requirements:**
  - Python (2.6 >)
  - PostgreSQL/PostGIS
  - Apache (2.x >) con mod_python

- **Python packages:**
  - psycopg2
  - isodate
  - GDAL
2. istSOS schema
### Submodules

- **lstSOS.filters**
  - lstSOS.filters.DS_filter
  - lstSOS.filters.GC_filter
  - lstSOS.filters.GF_filter
  - lstSOS.filters.GO_filter
  - lstSOS.filters.IO_filter
  - lstSOS.filters.RS_filter
  - lstSOS.filters.USD_filter
  - lstSOS.filters.factory_filters
  - lstSOS.filters.filter
- **lstSOS.renderers**
  - lstSOS.renderers.DSresponseRender
  - lstSOS.renderers.GCresponseRender
  - lstSOS.renderers.GFresponseRender
  - lstSOS.renderers.GOresponseRender
  - lstSOS.renderers.IOresponseRender
  - lstSOS.renderers.RSresponseRender
  - lstSOS.renderers.USDresponseRender
  - lstSOS.renderers.factory_render
- **lstSOS.responders**
  - lstSOS.responders.DSresponse
  - lstSOS.responders.GCresponse
  - lstSOS.responders.GFresponse
  - lstSOS.responders.GOresponse
  - lstSOS.responders.IOresponse
  - lstSOS.responders.RSresponse
  - lstSOS.responders.USDresponse
  - lstSOS.responders.factory_response
- **lstSOS.sosDatabase**
- **lstSOS.sosException**
Configure `mod_python`

```xml
<Directory "/var/www/sos">
    AddHandler mod_python py
    DirectoryIndex sos.py
    PythonHandler mod_python.publisher
    PythonDebug On
    PythonPath "['/var/www/sos/istSOSconfig']+sys.path"
</Directory>
```
5. configure istSOS

```python
#database properties
connection = {
    "user": "postgres",
    "password": "1234",
    "host": "localhost",
    "dbname": "sos",
    "port": "5432"
}

schema = "istsos"

#define the authority and version of your institution
#x- denote a not registered authority
authority = "x-ist"
version = ""
```
istSOS
(features & characteristics)
Supported requests

- **CoreProfile:**
  - GetCapabilities
  - DescribeSensor
  - GetObservation
- **Transactional Profile:**
  - RegisterSensor
  - InsertObservation
- **Enhanced profile:**
  - GetFeatureOfInterest

Not yet 😊

- **Enhanced profile:**
  - GetResult
  - GetObservationByID
  - GetFeatureOfInterestTime
  - DescribeFeatureType
  - DescribeObservationType
  - DescribeResultModel
For istSOS the sensor == time serie thus it is: “one set of values at one time”
FOI: featureOfInterest

- According the SOS it may be whatever you want but for istSOS it is the observed geometry and not the medium (e.g.: a point, a network, a region)

- istSOS schema allows only one FOI for each Procedure
sensor types

- Data are different for different sensor, so istSOS decided to distinguish sensors based on:
  - **observation type:**
    - Discrete (point, arc, area) – distributed
  - **sensor stationary:**
    - fixed – mobile
  - **sampling mode:**
    - in situ – remote
Up to now two supported sensor types

"fixpoint" = in situ – fixed – discrete point
sampling location is given by the FOI

[2010-09-05T12:10+02:00, 127.044]

"mobilepoint" = in situ – mobile – discrete point
sampling location is given by x,y,z triplet with the SRS adv in field description

[2010-09-05T12:10+02:00, 697812, 78562, 873.23, 12.7]
getObservation

1. In the response the time has **the same timezone** of the first element of the requested eventTime, (if missing it is assumed to be UTC)

2. If no eventTime is requested only the latest available observation is returned
3. Result parameter (filters on values) is not supported yet! 😞

4. featureOfInterest does not yet support spatial constrains! 😞

5. Non standard aggregateInterval and aggregateFunction parameters allow for data aggregation requests
regular time series

- istSOS uses **backward, open lower bound** intervals with frequency defined by <gml:timeInterval>

\[ \text{] } \text{ } \text{ } \text{] } = \text{open lower bound} \]

\(<gml:timeInterval \text{>} \text{PT1H}\text{</gml:timeInterval}>\)
6. responseFormat support also text/csv and application/Json formats

7. Support of irregular time series ("ad eventum" observation) and discrimination of no data and no event!
Irregular time series

- istSOS uses `<om:samplingTime>` to communicate the observed period, not the returned observations interval (min–max).
1. Automatically detect the sensor type (fixpoint or mobilepoint) depending on the presence of fields x, y, z in the provided observation template.

2. All new procedures are registered to a temporary offering.

3. Return the **sensorID**: this is the only time this value is accessible through internet!
1. Allows to **insert multiple values** and returns as a response an identifier that is the concatenation of observation id with the @ symbol. (1@2@3@4@5..)

2. **Time-value constrain**: one procedure has one property with one value for each instant

3. In case of **error no observation is registered** and the service answers with an exception (following the SOS specifications).
4. **Observed period is updated** accounting for submitted `<om:samplingTime>` (time of observation of submitted data)

5. **forceInsert: non standard** parameter for data management; in this case the `insertObservation` substitutes all the observation within the `<om:samplingTime>` with the new observations (if any provided)
• NON standard request: allows for submission of a new sensor description that substitutes the current one

• this is to account for historical changes of instruments or particular maintenance tasks
istSOS allows to define virtual procedures extending the base class `virtual Process` that has a method for retrieving of classic procedure data based on submitted filters parameters.

→ data may reside wherever you want, just read the filter, get the data and return the record in a few lines of code!!
from istSOS.responders.GOResponse import VirtualProcess
import datetime, decimal

class istvp(VirtualProcess):
    def __init__(self, filter, pgdb):
        VirtualProcess.__init__(self, filter, pgdb)

    #SET THE INPUTS
    self.h = self.setSOSSoVariable("A_BRB", "riverheight")

    def execute(self):
        data_out = []
        for rec in self.h:
            newdata = rec[1] * 0.25 + 124
            data_out.append((rec[0], newdata))
        return data_out
…Try it out…

http://istgeo.ist.supsi.ch/software/istsos/

…and join to the development