Technische Universität Berlin
Chair of Geoinformation Science

GIS Project
A GIS based structural health monitoring system
Summer Term 2015

istSOS Workpackage Report

Authors:
Farooq, Umar [359512] - umar.farooq@campus.tu-berlin.de
Hossan, Md. Nazmul [355039] - hossan@campus.tu-berlin.de
Kouseras, Thomas [360387] - thomas.kouseras@campus.tu-berlin.de

Berlin, April 13, 2015

Contents

List of Figures

List of Tables

1 List of Abbreviations

CSV: Comma Separated Values
DAT: Data file
HTTP: Hypertext Transfer Protocol
JDBC: Java Database Connectivity
GUI: Graphical User Interface
GIS: Geographic Information System
OGC: Open Geospatial Consortium
SensorML: Sensor Model Language
SOS: Sensor Observation Service
SWE: Sensor Web Enablement
SHM: Structural Health Monitoring
SHMS: Structural Health Monitoring System
SQL: Structured Query Language
UML: Unified Modeling Language
XML: Extensible Markup Language

2 Introduction

This projects concerns the development of an automated GIS-based structural health monitoring system for a bridge. As part of this project the istSOS was called to support the transaction of data:

1. provide an OGC SOS conformant web server on top of a compliant database to store data recorded from the sensor network
2. provide a standardized web service, enabling data querying using HTTP 'GET' and 'POST' requests, through an easy to use browser based user interface

The istSOS Observation Service Data Management System was developed by the Institute of Earth Sciences (IST, Istituto Scienze della Terra) of Switzerland [1]. It is an open source application for managing observations according to the SOS standard [2] written in Python.

It was opted for on the following grounds:

- it is solely developed on Open Source software
- it is OGC compliant (SOS standard)
- consequently to the above points, it ensures the standardization of data flow, which itself enables automated procedures, without involving user's intervention at any part of the process
- the project provides also a Graphical user Interface that allows for easing the daily operations and a RESTFull Web api for automatizing administration procedures
- its transactional profile includes the registerSensor functionality

One of the main visions in developing such a project is to automate, or at least minimize the need of a human operator, when it comes to data generation, data format transformation, storage and export. Such an automated process could further support the potential for developing a hybrid SHM system. That is a system which stays in hibernation mode and alerts the stakeholders and starts recording when a user defined event is detected.

istSOS with its OGC standards compliance enables automated or semi-automated registration of sensors, data transformation, import and export routines.

3 Server side setup

This section the server side setup is described, specifically the server setup and the database architecture and setup.

3.1 Server setup

The computer employed in the project is DELL Precision 370 equipped with Inter Pentium (R) CPU with 800MHz clock, 1 GB memory and 160 GB hard drive. The operation system installed on the computer is GUN Linux distribution Ubuntu.

The web server is an Apache 2.2 HTTP Server, hosted in TU Berlin in the department of Geodesy and Geoinformation under the url "quader.igg.tu-berlin.de"

3.2 Database Set Up

The database on top of which the istSOS application functions is a postgreSQL 9.1 with the postGIS extension enabled. The schema of the database is presented in figure 1. The administrator using the web admin interface provided by istSOS walib (web library) is able to create various services (istSOS database instances), which automatically instantiate a database, which implements the schema depicted in figure 1. In our case that means that each monitoring system has its own independent database. According to the schema sensors are mapped as procedures and are grouped in offerings and features of interest. The phenomena being measured are stored as observed properties and data as measures. An extra table called event_time stores the various timestamps. Observations are finally linked with event_time, observed properties, units of measurement and procedures through various foreign keys.

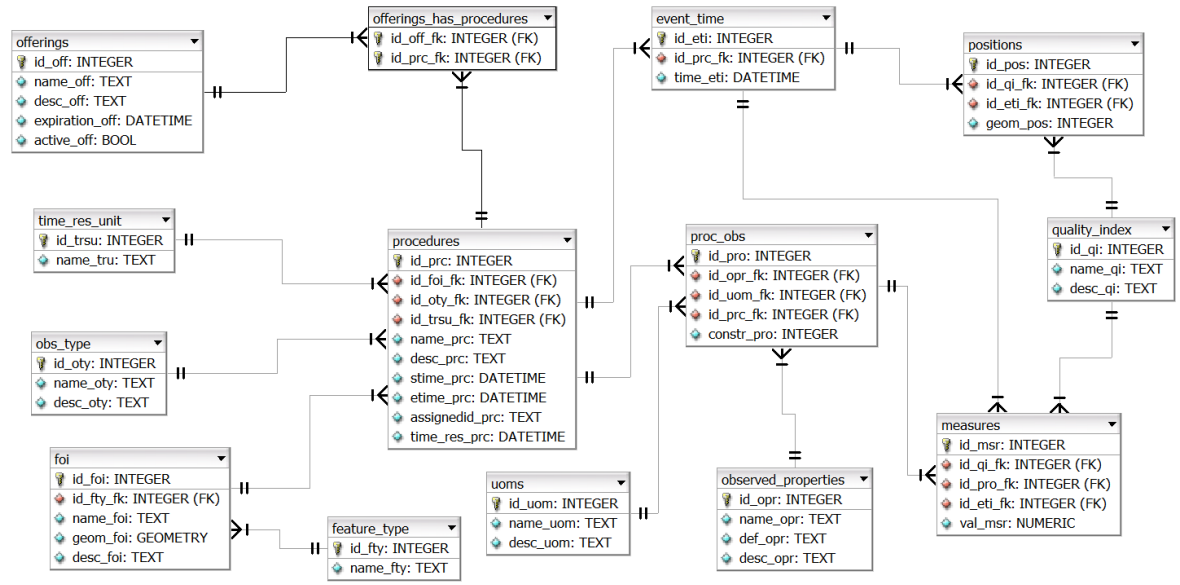

Figure 1: istSOS database schema (source: `https://geoservice.ist.supsi.ch/projects/istsos/index.php/IstSOS`).

The administrator/user using the web interface declares all the procedures (sensor), the observed properties (phenomena being measured), units of measurements and groups of sensors (offerings and

features of interest) involved in his project, without having to know the database architecture. A simplified procedure is followed by typing information in text fields .

The current project concerns a network of sensors deployed on a model bridge. This sensor network consists of four types of sensors:

- triaxial accelerometers
- strain gauges
- displacement sensors
- thermometers

which are connected in four sensor nodes of two types:

- G-Link-LXRS Wireless Accelerometer Node (Nodes 383, 384)
- V-Link-LXRS Wireless 7 Channel Analog Input Sensor Node (Nodes 573, 574)

Nodes 383 and 384 are G-Link-LXRS Wireless Accelerometer Nodes with 4 channels. Each channel is connected to one observed property:

1. channel 1 is recording acceleration in x axis (m/s^2)
2. channel 2 is recording acceleration in y axis (m/s^2)
3. channel 3 is recording acceleration in z axis (m/s^2)
4. channel 4 is recording temperature (^{o}C)

Node 573 is a V-Link-LXRS Wireless Analog Input Sensor, with four active channels connected to 3 strain gauges and one temperature sensor:

1. channel 1 is recording strain (μ Strain)
2. channel 2 is recording strain (μ Strain)
3. channel 3 is recording strain (μ Strain)
4. channel 8 is recording temperature (^{o}C)

Node 574 is a V-Link-LXRS Wireless Analog Input Sensor, with four active channels connected to 2 strain gauges, one displacement sensor and one temperature sensor:

1. channel 1 is recording strain (μ Strain)
2. channel 2 is recording strain (μ Strain)
3. channel 5 is recording displacement (mm)
4. channel 8 is recording temperature (^{o}C)

Table 1 lists the different types of available sensors, with their observed properties and table 2 depicts the sensor node relationship.

Table 1: Sensor types and observed properties involved in the project

Sensor Type	**Observed Property**	**Unit**
G-Link-LXRS Wireless Accelerometer	Acceleration (x,y,z), Temperature	m/s^2,^{o}C
KFG series Foil Strain Gage	Strain	μ Strain
S-DVRT Subminiature Displacement Sensor	Displacement	mm
Temperature sensor on V-Link-LXRS Wireless node	Temperature	^{o}C

Table 2: Node sensor linkage involved in the project

Node	Sensors	Observed Properties
Node 383	G-Link-LXRS Wireless Accelerometer	Acceleration (x,y,z), Temperature
Node 384	G-Link-LXRS Wireless Accelerometer	Acceleration (x,y,z), Temperature
Node 573	KFG series Foil Strain Gage (x3) Temperature sensor on V-Link-LXRS Wireless node	Strain, Temperature
Node 574	KFG series Foil Strain Gage (x3) Temperature sensor on V-Link-LXRS Wireless node S-DVRT Subminiature Displacement Sensor	Strain, Displacement, Temperature

The customized solution chosen for the above described monitoring network is graphically presented in figure 2.

Specifically *Node_383* and *Node_384* were declared as procedures, each of them measuring 4 observed properties, acceleration in x,y,z axis and temperature. It was decided this way, because all those sensors are mounted (fixed) on the node. In addition one needs to record acceleration in all three axis at the same time and all three accelerometers should be treated as one sensor.

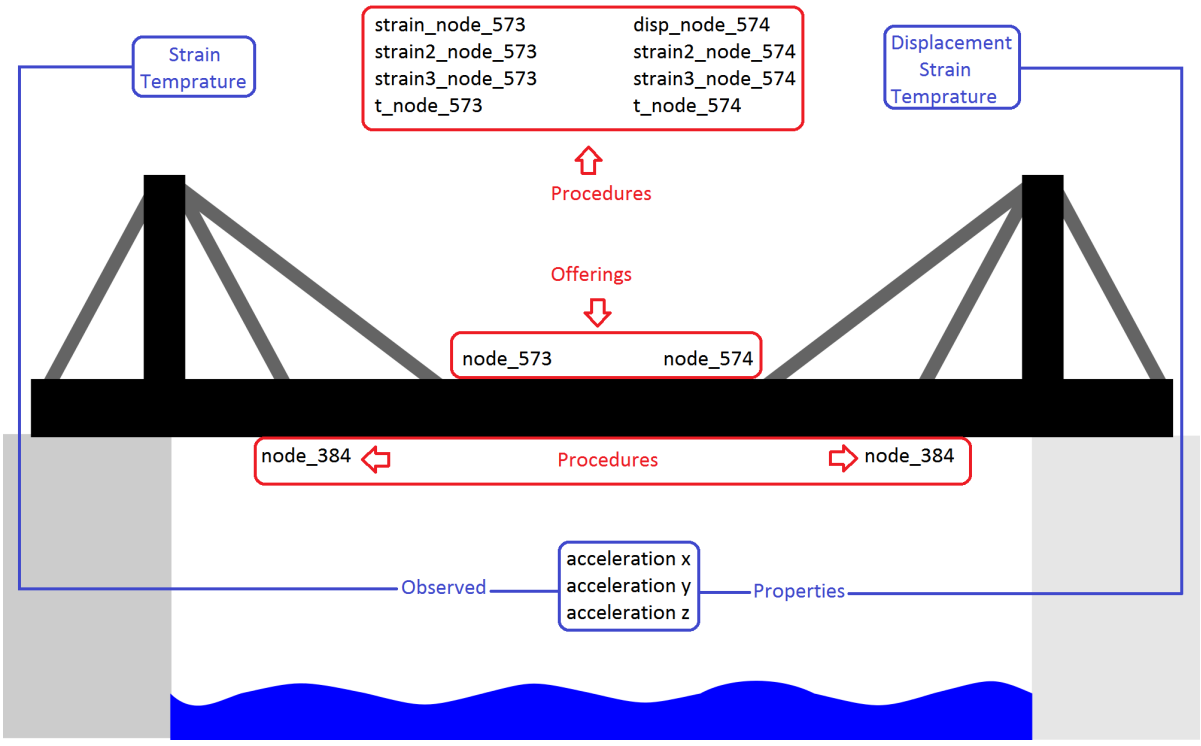

Figure 2: Offerings, procedures and observed properties declared in istSOS database according to the monitoring network chosen.

Node 573 and *Node 574* are connected to a set of different sensors placed in various positions on the structure. They can be plugged in and out of the sensor nodes and their position could be changed. On these grounds sensor nodes 573 and 574 were declared as offerings (*node_573* and *node_574*) and each sensor connected to them as independent procedure. Procedures linked with node 573 are grouped together under the *node_573* offering and the ones linked with node 574 under the *node_574* offering (table 3).

Table 3: Monitoring network as declared as offerings and procedures in istSOS database

Offerings	Procedures	Observed Properties
	Node_383	acceleration-X acceleration-Y acceleration-Z air-temeperature
	Node_384	acceleration-X acceleration-Y acceleration-Z air-temeperature
node_573	strain_node_573 strain2_node_573 strain3_node_573 t_node_573	strain air-temeperature
node_574	strain_node_574 strain2_node_574 displ_node_574 t_node_574	strain displacement air-temeperature

The customized database solution includes an extra offering named *Calibration* which groups 7 "*dummy*" procedures which are used to store the calibration parameters for each of the accelerometer channels of nodes 383 and 384 and one to store the strain gage calibration parameters (figure 3).

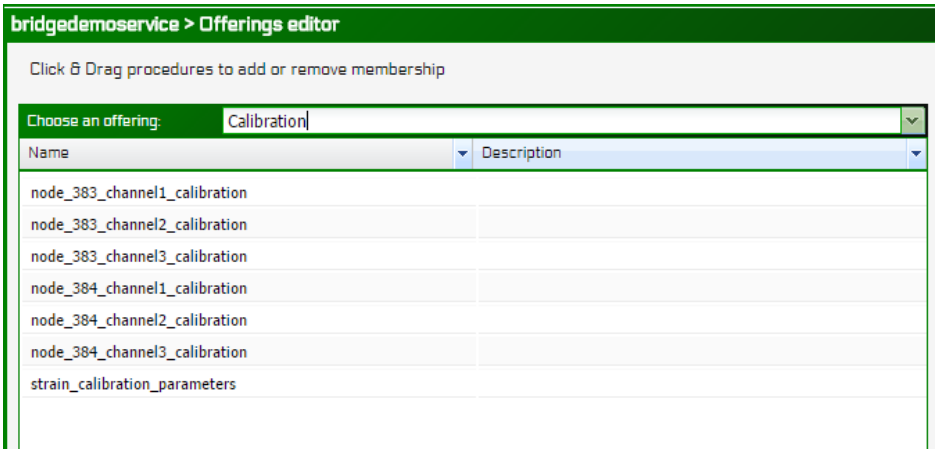

Figure 3: Offering to hold the *dummy* calibration procedures.

4 Client side setup

On the client side and after the system administrator has instantiated and set up the istSOS service and database and created and tuned all offerings, procedures, observed properties and units of measures, the end user using the functionality provided by an OGC compliant Web Service and istSOS Observation Service Data Management System is able to import and export data through HTTP requests. Furthermore through customized solutions user can import data through different routines (HTTP 'POST' request, direct insertion through SQL queries, by employing the cmdimportcsv.py script offer by istSOS) or export them through a simplified, user friendly web interface. Such solutions can be developed and customized according to user needs by the administration and can be handed over to him. Their main benefit is that they are portable and it is not demanded from the user to have specialized knowledge on the database of his service.

Such client side solutions can include a portable Data Insertion Manager and a Web Data Querying Manager (accessed by a URL provided by the administrator).

4.1 Data Insertion Manager

To enable the data insertion in the database, an application in JAVA has been developed and tested. The application is portable, it comes in an executable '*.jar*' file, which can be executed in any computer that has a Java Virtual Machine installed. The user interface is a Graphical User Interface (figure 4). The Graphical User Interface ensures that the user doesn't need any programming or other special technical skills to handle it, just knowledge of his monitoring service.

In the following sections the functionality of the istSOS Data Insertion Manager is explained.

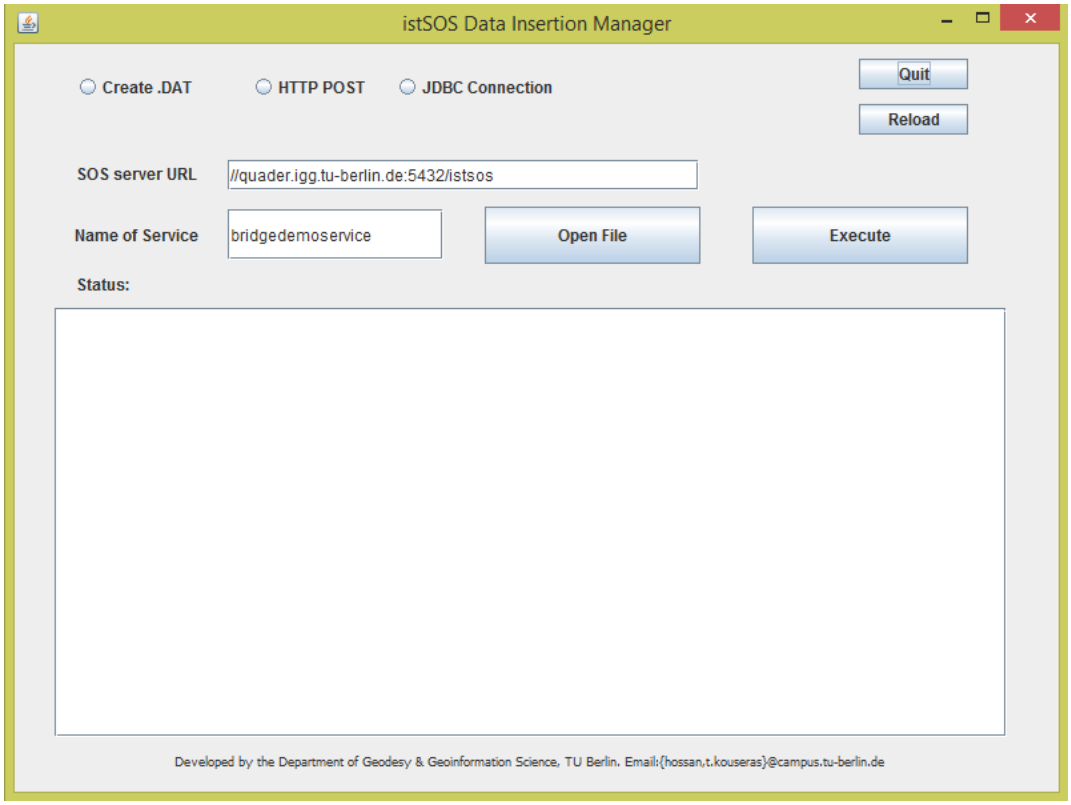

Figure 4: istSOS Data Insertion Manager Graphical User Interface

The aim of designing such a graphical user interface for the whole data insertion process so that an user is able to insert sensor observation easily into the database with a very basic istSOS internal structure knowledge. The GUI (figure 4) will only ask the user to feed the following parameters into the program and the rest of the process are fully automatically done by the developed application.

1. select any of the data insertion method which is displayed as radio buttons
2. write the URL of the server (default is *'//quader.igg.tu-berlin.de:5432/istsos'*)
3. write the name of service (default is *'bridgedemoservice'*)
4. press "Open file" button and select the sensor data file (.csv format)
5. press "Execute" button and the system corresponds with the user request

The **Status** part is an output screen where all the console outputs and error messages are redirected and showed by the application. Reload button is placed to re-initiate the application and Quit is introduced to close the application. Table 4 shortly describe the pseudo algorithm for istSOS data insertion manager GUI application.

Table 4: Pseudo algorithm for GUI interface

Algorithm 1. istSOS data insertion manager GUI

```
1: start the application
2: get the type of method, mode = 1; 2; 3
3: get the SOS server URL, url (e.g. //quader.igg.tu-berlin.de:5432/istsos)
4: get the name of service, nameOfService(e.g. bridgedemoservice)
5: get the input data file location, fileLoc = fileLocator.jfilechooser()
6: execute:
    if mode = 1:
        call importDAT.csv2DAT(fileLoc)
    end if
    else if mode = 2:
        call importXML.importPOSTXML(url, nameOfService, fileLoc)
    end else if
    else if mode = 3:
        call importJDBC.SelectNode(url, nameOfService, fileLoc)
    end else if
    else:
        show some error messages
    end else
7: reload/quit application
```

Algorithm [1] is the simplified version of the implemented Java MainGUI class. The details UML Java classes are depicted in (figure 5) where the dependencies and the association between different classes has been listed.

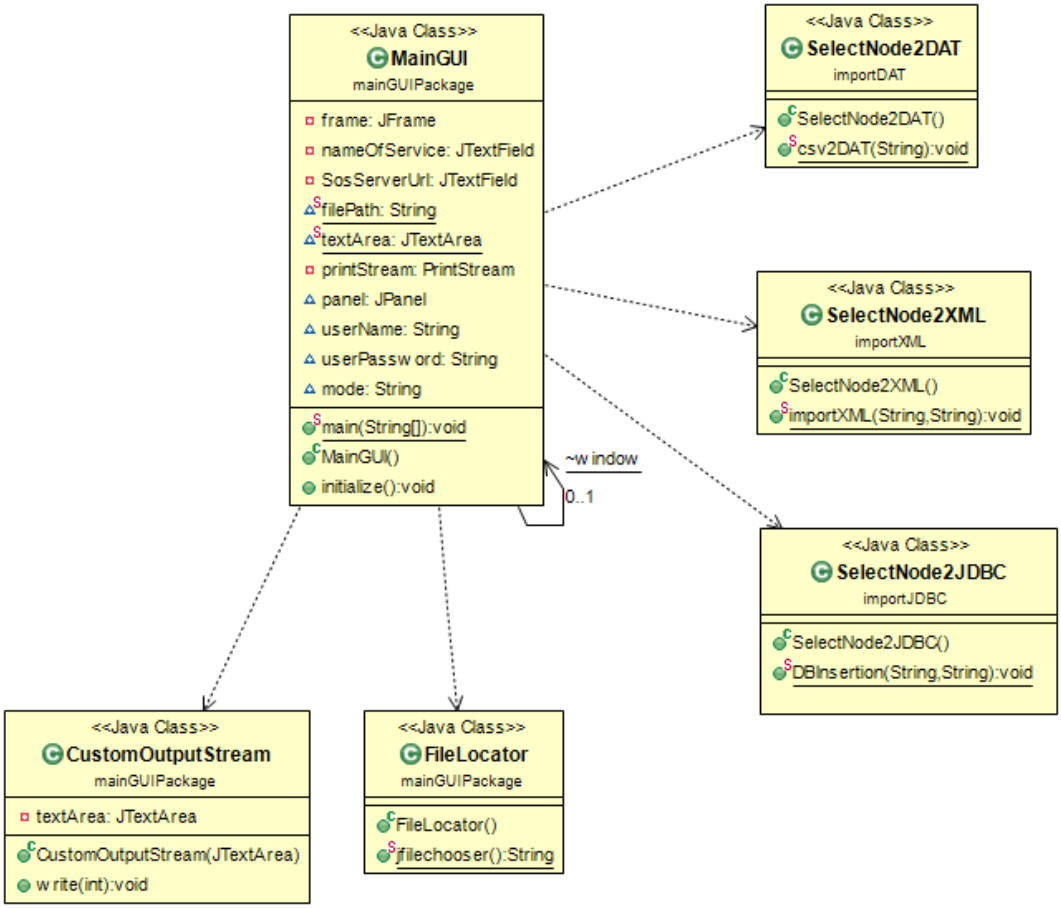

Figure 5: UML << *Java Class* >> dependencies between MainGUI and other methods

The application contains four main packages which are:

1. MainGUIPackage
2. ImportDAT
3. ImportXML
4. ImportJDBC

All of the data insertion packages contains the *'SelectNode2* [1] *'.java* program in order to extract the name of procedure and then invoke the particular function according to the user request. The main philosophy behind the *'SelectNode2* [1] *'* program is explained in table 5.

[1] DAT;XML;JDBC

Table 5: Pseudo algorithm for *SelectNode2* [1]

Algorithm 2. Selection of correct node (*SelectNode2* [1]*.java*)

```
1: inputDataFile=initialize CSVReader and read the input file
2: parse nodeType from inputDataFile(line2)
3: make decision:
    if the nodeType is of 384, or 383:
        call the particular className.specificMethods()
    end if
    else if the nodeType is 573:
        initialize numOfProcedures from position 1 in line4
        For i is 0, i is less than numOfProcedures, i increments by 1
          in each iteration scan the type of procedure and call the corresponding function
          and read the corresponding channel and mark the
          column of the data file which holds the data we want to extract
          (4 channels in node 573, channels 1,2 and 3 are strain gauges and channel 8 temperature sensor)
          Switch on channel
          case 1:
              set procedureName to "strain_node_573"
              call the particular className.specificMethods()
          case 2:
              set procedureName to "strain2_node_573"
              call the particular className.specificMethods()
          case 3:
              set procedureName to "strain3_node_573"
              call the particular className.specificMethods()
          case 8:
              set procedureName to "t_node_573"
              call the particular className.specificMethods()
    end else if
    else if the nodeType is 574:
        initialize numOfProcedures from position 1 in line4
        For i is 0, i is less than numOfProcedures, i increments by 1
          in each iteration scan the type of procedure and call the corresponding function
          and read the corresponding channel and mark the
          column of the data file which holds the data we want to extract
          (4 channels in node 574, channels 1,2 are strain gauges, channel 5 is displacement sensor
          and channel 8 temperature sensor)
          Switch on channel
          case 1:
              set procedureName to "strain_node_574"
              call the particular className.specificMethods()
          case 2:
              set procedureName to "strain2_node_574"
              call the particular className.specificMethods()
          case 5:
              set procedureName to "disp_node_574"
              call the particular className.specificMethods()
          case 8:
              set procedureName to "t_node_574"
              call the particular className.specificMethods()
    end else if
    else:
        show some error messages
    end else
4: return or finish execution
```

4.1.1 Insert observation through command line (cmdimportcsv.py)

Insert observation through the istSOS library file which can be found in the istSOS root folder named *cmdimportcsv.py* and the user have to write necessary commands in the command line environment to use the above mentioned (.py) python file. Before using *cmdimportcsv.py*, the input data file should be formated according to the istSOS developer requirement which are as follows:

- The header must contain the URN names of the data followed in the same column (e.g. urn:ogc:def:parameter:x-istsos:1.0:acceleration-X)
- The time stamp is a mandatory field and must follow ISO8601 time standard (e.g. 2015-03-26T13:16:01.236125000+0100)
- The different observed properties must be separated with comma (,)
- The name of data file must be the same name of the procedure
- The data file format should be a (.DAT) or (.CSV) or other supported file format

The application is designed such a way that it recognizes the different procedure and is able to generate (.DAT) files into the folder where the input data files are located. For example, for Node_383 and Node_573, it generates 1 and 4 different (.DAT) files respectively.

The more details about importDAT Java package are shown in the (figure 6) which is a UML class diagram for the mentioned package.

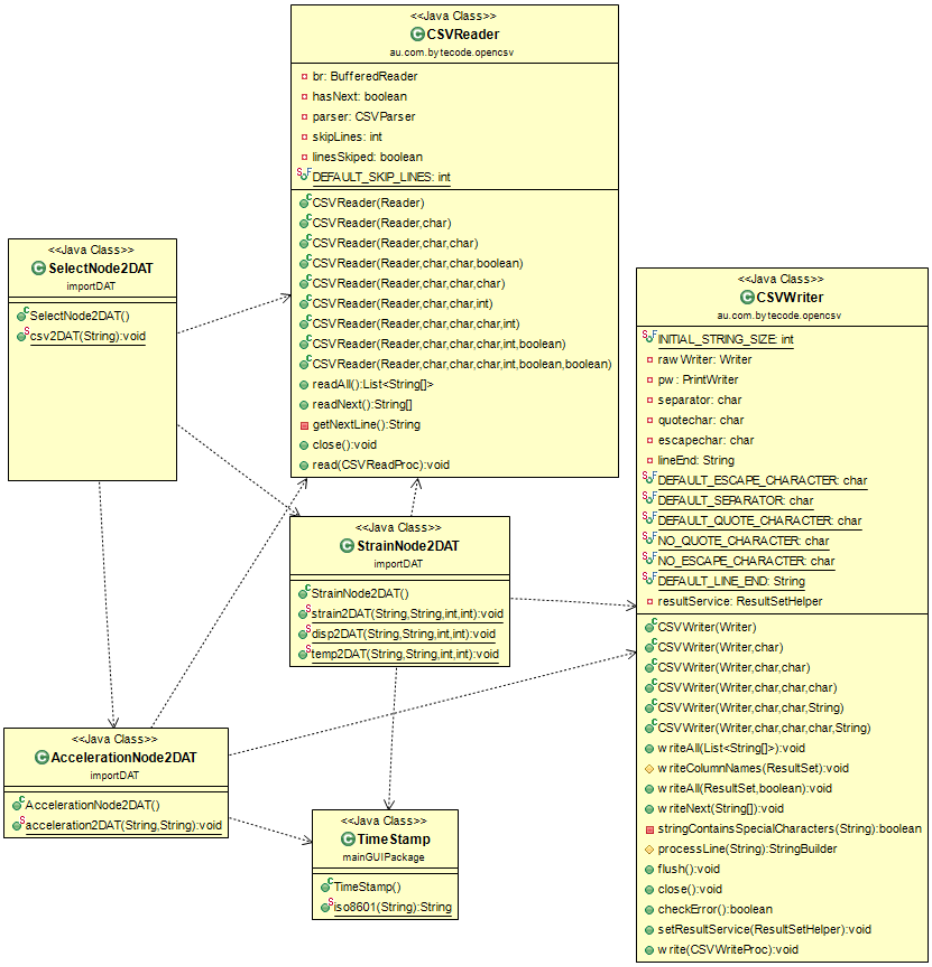

Figure 6: UML << *Java Class* >> diagram for creating .DAT files

According to the algorithm [1] described in table 4, *MainGUI.java* program takes the user request into account and if the user wants to generate (.DAT) files, then the *SelectNode2DAT.java* program has been invoked and it decides the nodeType and call that particular function and create output files is (.DAT) format with all relevant headers. The program always transforms the originated time into the ISO8601 standard time. Now the user is responsible to open command line and insert the observation through command line.

4.1.2 Insert observation with 'POST' XML request

Using a HTTP POST request is another possible solution to insert observation into the istSOS database. For this purpose, the application is designed to write an (.XML) document according to the specific procedure and then send request to the SOS server.

The UML class diagram in (figure 7) shows all the different classes involved in importXML package. *SelectNode2XML.java* program parse the nodeType and call two different java program for two different node type. It calls *AcclerationNodeXML.java* for Node_383 and 384. It calls *WriteXMLBody.java* program for Node573 and 574 and write the XML files where the input data files were located.

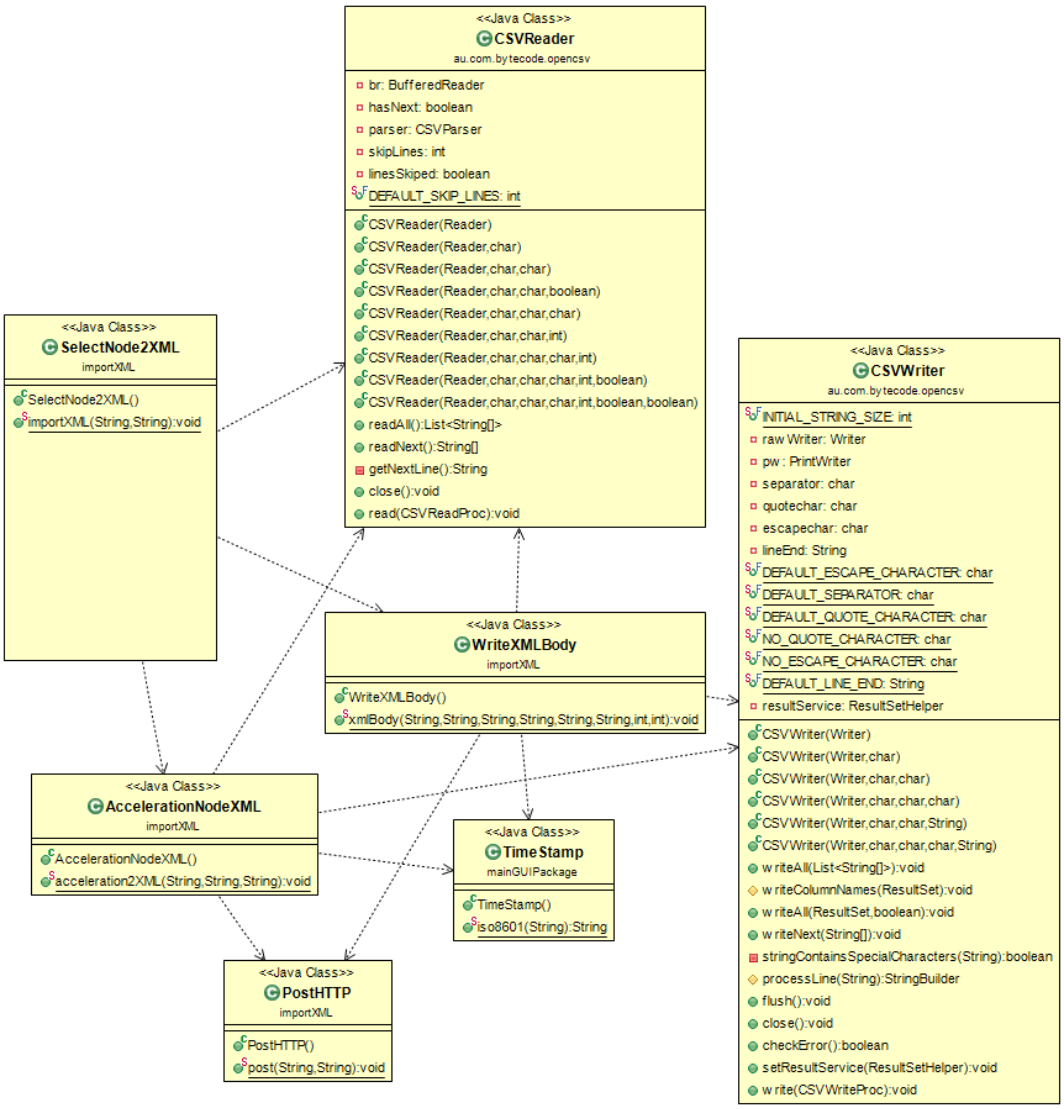

Figure 7: UML << *Java Class* >> diagram for creating XML files and send HTTP POST request to the server

At the moment of writing XML body, the < *gml* : *beginPosition* > of the procedure has to be feed which will be the last observation which is already exists in the database. So, this case we designed our program to connect to the database and perform SQL query to grab the time for < *gml* : *beginPosition* > from the procedures table under the column name etime_prc.

Once the XML file has been created successfully, now time to send that file to the SOS server and complete the data insertion process. *PostHTTP.java* program is designed to send HTTP POST request to the server and catch the response body from the server.

4.1.3 Direct insertion with PostgreSQL JDBC

The most successful method is to insert data directly from the Java environment to PostgreSQL/-PostGIS database through JDBC connection.

The most important part in direct data insertion was to respect all primary and foreign keys and also the relationship between different tables. Once the administrator declares a new service in the istSOS

admin panel, the system automatically creates a new instance of istSOS database schema and all the tables shown in (figure 1) are created inside the new schema (name of service).

In the following section, the pseudo algorithm for data insertion through SQL queries are described.

Table 6: Pseudo algorithm for Database SQL Queries

Algorithm 3. DBInsertion through SQL Queries

1: connect to the istSOS database
2: retrieve id_prc_fk() from procedure where name_prc = nameOfProcedure
3: retrieve id_qi_fk from quality_index (e.g it is 100 for raw data)
4: retrieve id_opr_fk from observed_properties where
name_opr=acceleration-(x,y,z),air-temperature;strain;displacement
5: retrieve id_pro_fk from proc_obs where id_opr_fk=step4 and id_prc_fk = step2
6: insert into event_time (id_prc_fk, timeValue)
7: retrieve id_eti_fk from event_time where time_eti=timeValue and id_prc_fk=step2
8: insert into measures (data, id_eti_fk, id_qi_fk, id_pro_fk)
9: **if** stime_procedure = **null** then update stime into procedure table where name_prc= nameOfProcedure
10: update etime into procedure table where name_prc= nameOfProcedure
11: close database connection

Algorithm [3] in table 6 is the simplified approach for the direct data insertion through SQL queries into the istSOS database. The main challenge of this approach is to get the '*name_opr*' from the data file. In practice the input data file doesn't contain the appropriate name of observed properties but only the name of channel with its units of measure. Till now the '*name_opr*' is hard coded and the program is able to insert data into the particular procedure automatically.

The UML class diagram for the importJDBC package and the sub classes are depicted in the (figure 8).

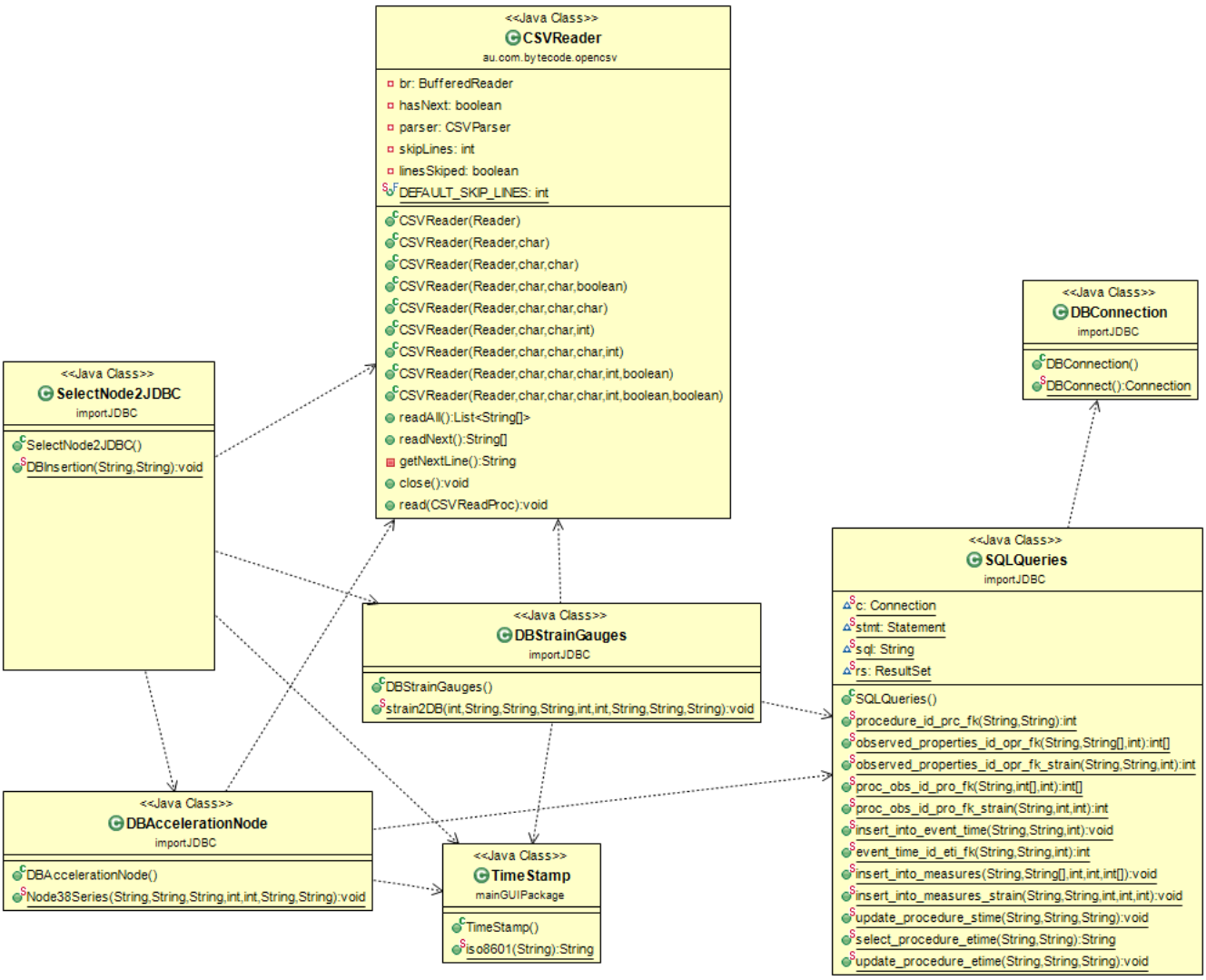

Figure 8: UML << *Java Class* >> diagram for direct data insertion to the database

It has to be mentioned that all the primary keys are generated by the database itself and the user doesn't have to care about the numbers. Through JDBC method the only hard coded part is the name_opr which is fixed as follows:

- for 'Node38' series name_opr = {acceleration-X,acceleration-Y,acceleration-Z,air-temperature}
- strain gauges for 'Node57' series name_opr = {strain}
- displacement sensor for 'Node574' name_opr = {displacement}
- temperature node for 'Node57' series name_opr = {air-temperature}

Another invariants for this approach is the 'name of procedures' and this can also be parsed from the sensor data file coming from Nodemaster for 'Node38' and 'Node57' series.

4.2 Web Data Querying Manager

In such a monitoring system the existence of a tool or interface to query and export OGC standardized datasets, without demanding specialized knowledge from the user, is imperative.

Towards such a solution a Web Data Querying Manager (figure 9) was developed to enable all the potential given by the SOS web service. This Web Data Querying Manager was developed in JavaScript in synergy with some HTML elements. It can be included in the Web Interface of the project, which will reside in the server (’*quader.igg.tu-berlin.de*’) and could be accessed by the user through a url.

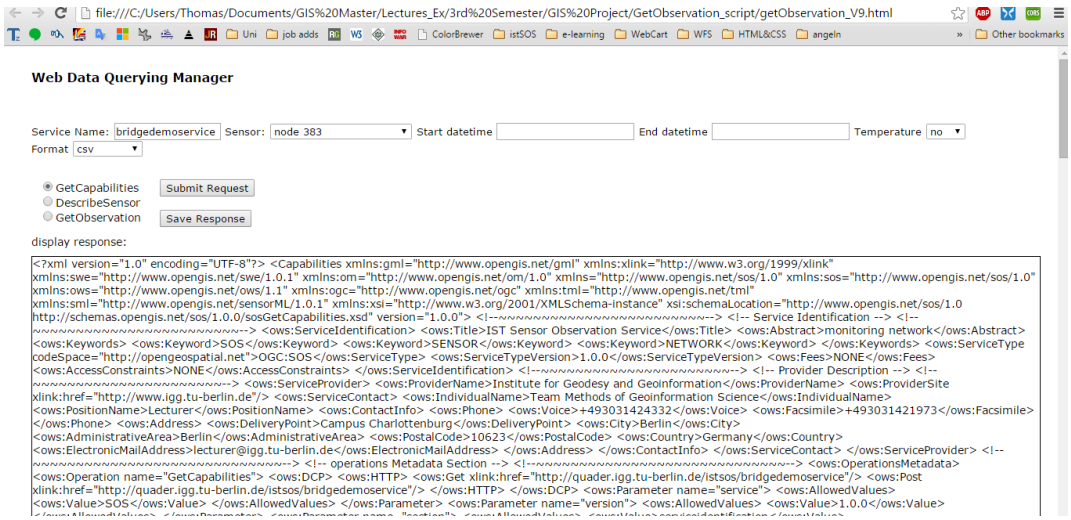

Figure 9: Web Data Query Manager displaying a ’getCapabilities’ responce

Accessing the url through his browser the user, only needing an Internet connection and the name of his service, can employ the Web Data Querying Manager to query dataset for further processing (accomplished through other processes of Web Interface) or even download and export.

The tool was developed to provide automation from querying datasets to processing and plotting them on the fly, without having to download them, export and import them all the time, and to help the user from forming all these complicated HTTP ’GET’ requests. Instead through simple dropdown lists and datetime pickers the user selects the crucial variables determining the requests and the script build them, send them to the server and displays the response.

The tool is utilized with the following course of action:

1. type the name of the service, as was given in istSOS
2. send a ’getCapabilities’ request and inspect the service through the response document
3. decide on sensor dataset, send ’describeSensor’ request and read observed properties included in sensor and time frame of its dataset
4. select sensor of interest and start and end date-time of dataset (figure 10)
5. select desirable output format and whether dataset should be coupled with temperature data (figure 11)
6. send or save response

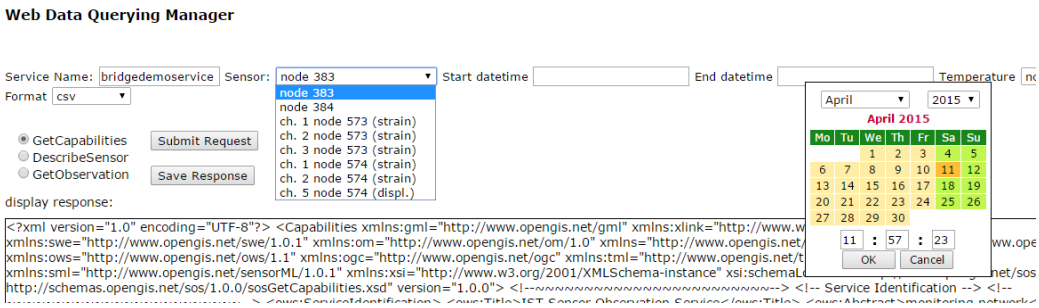

Figure 10: Web Data Query Manager sensor and date selection for a 'getObservation' request

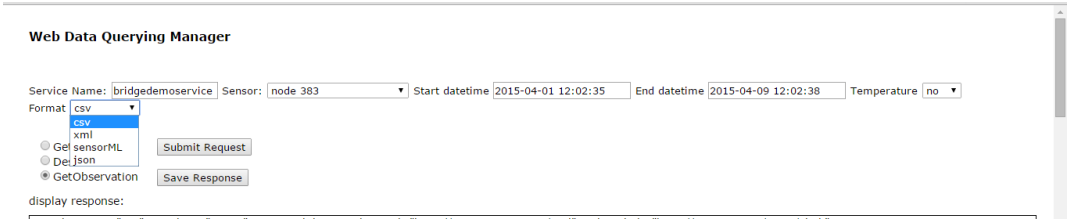

Figure 11: Web Data Query Manager to save 'getObservation' request in different file formats

All requests are built automatically after the user has selected the all the crucial parameters from the selection fields, that is service name, request type, sensor of interest, start and end date, temperature data and output format. According to request type the script attaches the need parameters together to a single url and an AJAX asynchronous request sends the HTTP Requests to server. The request types supported are GetCapabilities, DescribeSensor and GetObservation. The response is displayed on the browser and as a pop-up window. Alternatively can be exported and saved in a file whose format is also decided by the user. All responses are OGC compliant formatted. Output formats supported are 'text/plain', 'text/xml', 'text/xml;subtype='sensorML/1.0.0" and 'text/x-json'.

At this stage of development sensor names are hard coded according to the service procedure. In case new sensors - procedures are registered in the service or procedure names are changed the user has to notify the administrator to adapt the Web Data Querying Manager script, to conserve compliance with the database. In future development when the service name is given a 'getCapabilities' request would be send, parsed on the fly and populate the sensors dropdown list with the actual available procedures in the service.

5 Discussion and future work

The ultimate objective of developing such a monitoring scheme, at least on the data transaction and storage level, is to ensure automated procedures that will record, store and notify the stakeholders in case of a user defined event.

At current state a semi-automated scheme has been developed and the intervention of human operator is imperative in many steps of the procedure.

1. **Sensor Registration:** After an istSOS instance is created and the service is setup for each new sensor being installed the user has to notify the administrator to declare the new sensor as a procedure through the istSOS Web Admin.
2. **Data Storage:** Once a sensor is up and running and recording data and at the same time declare as istSOS procedure, the administrator has to customize all the hard coded parts in the

Data Insertion and Web Data Querying Manager to include the new procedure and possible new observed properties.

3. **Data Insertion:** To insert the data into the database the user has to export a '.csv' file from the Nodemaster and using the Data Insertion Manager manually select the data file and choose the inserting operation.
4. **Data Exporting:** Once the database has been changed, i.e. a service has been tuned, the hard coded parts of the Web Data Querying Manager has to be changed in order to be correctly utilized.

6 Towards a fully automated solution

The developed Java application for istSOS data insertion described in chapter [4.1] is not a generalized way but its automatic. The observed properties and name of the procedure is hard coded inside the program. It can be generalized and also automatic if the application ask only this two parameter from the operator before insertion the sensor data into istSOS database. It would also be possible in future to make it fully automated and more generic. The following section discusses about different possibilities towards a fully automated solution.

1. Till now the registration of a new sensor is done by the istSOS admin panel with the necessary input parameters. The same operation can be done through the HTTP POST request with 'registerSensor'. Alternatively registration of a new sensor through direct database SQL queries and combine with the portable Data Insertion Manager.
2. SQL queries has to be send to change dynamically lists of sensors names, procedure urns and observed properties urns, used by the Data Insertion Manager
3. Once sensors are registered an application has to be developed to insert data from sensors - Nodemasters at the moment they are produced. It would also be a possible future research to discover the sensor data automatically with a time interval from a predefined location.
4. Once the service name is provided by the user in Web Data Query Manager, a 'getObservation' request can be sent, the response parsed on the fly, and the sensor list populated according to the procedure names present in the service. Start and end times could also be given.

7 Appendix

7.1 Structure of the generated (.DAT) file

An example of (.DAT) file formated according to the istSOS application requirement which was generated for strain_node_573 is depicted in the following section.

```
1  urn:ogc:def:parameter:x-istsos:1.0:time:iso8601,urn:ogc:def:parameter:x-istsos:1.0:...
       strain
2  2014-05-13T07:56:50.101750000+0100,-70.3302002
3  2014-05-13T07:57:00.101750000+0100,-61.91453552
4  2014-05-13T07:57:10.101750000+0100,-58.70856857
5  2014-05-13T07:57:20.101750000+0100,-58.10744858
6  2014-05-13T07:57:30.101750000+0100,-52.49700546
7  2014-05-13T07:57:40.101750000+0100,-37.66940308
8  2014-05-13T07:57:50.101750000+0100,-28.45224571
9  2014-05-13T07:58:00.101750000+0100,-74.13729095
10 2014-05-13T07:58:10.101750000+0100,-74.73841095
11 2014-05-13T07:58:20.101750000+0100,-75.53990173
12 2014-05-13T07:58:30.101750000+0100,-75.53990173
13 2014-05-13T07:58:40.101750000+0100,-70.53057098
```

7.2 Structure of the generated (.XML) file

An example of XML encoded file is depicted below which was generated for Node_383 by the importXML Java package.

```
<?xml version="1.0" encoding="UTF-8"?>
<sos:InsertObservation
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.opengis.net/sos/1.0.0/sosAll.xsd"
xmlns:sos="http://www.opengis.net/sos/1.0"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:sa="http://www.opengis.net/sampling/1.0"
xmlns:swe="http://www.opengis.net/swe/1.0.1"
xmlns:gml="http://www.opengis.net/gml/3.2"
xmlns:ogc="http://www.opengis.net/ogc"
xmlns:om="http://www.opengis.net/om/1.0" service="SOS" version="1.0.0" >
<sos:AssignedSensorId>d4920e8252118544b6ae5c08207e90d6</sos:AssignedSensorId>
<om:Observation>
<om:procedure xlink:href="urn:ogc:def:procedure:x-istsos:1.0:Node383"/>
<om:samplingTime>
<gml:TimePeriod>
<gml:beginPosition>2014-06-13T11:56:48.006750000+0100</gml:beginPosition>
<gml:endPosition>2014-06-13T12:56:49.506750000+0100</gml:endPosition>
</gml:TimePeriod>
</om:samplingTime>
<om:observedProperty xlink:href="urn:ogc:def:property:x-istsos:1.0:composit" >
<swe:CompositPhenomenon dimension="5">
<swe:component xlink:href="urn:ogc:def:parameter:x-istsos:1.0:time:iso8601"/>
<swe:component xlink:href="urn:ogc:def:property:x-istsos:1.0:accelerationx"/>
<swe:component xlink:href="urn:ogc:def:property:x-istsos:1.0:accelerationy"/>
<swe:component xlink:href="urn:ogc:def:property:x-istsos:1.0:accelerationz"/>
<swe:component xlink:href="urn:ogc:def:property:x-istsos:1.0:air:temperature"/>
</swe:CompositPhenomenon>
</om:observedProperty>
<om:featureOfInterest xlink:href="urn:ogc:object:feature:x-istsos:1.0:station:test"/>
<om:result>
<swe:DataArray>
<swe:elementCount>
<swe:Count>
<swe:value>6</swe:value>
</swe:Count>
</swe:elementCount>
<swe:elementType name="SimpleDataArray">
<swe:DataRecord definition="urn:ogc:def:dataType:x-istsos:1.0:timeSeries">
<swe:field name="Time">
<swe:Time definition="urn:ogc:def:parameter:x-istsos:1.0:time:iso8601"/>
</swe:field>
<swe:field name="accelerationx">
<swe:Quantity definition="urn:ogc:def:property:x-istsos:1.0:accelerationx">
<swe:uom code="G"/>
</swe:Quantity>
</swe:field>
<swe:field name="accelerationy">
<swe:Quantity definition="urn:ogc:def:property:x-istsos:1.0:accelerationy">
<swe:uom code="G"/>
</swe:Quantity>
</swe:field>
<swe:field name="accelerationz">
<swe:Quantity definition="urn:ogc:def:property:x-istsos:1.0:accelerationz">
<swe:uom code="G"/>
</swe:Quantity>
</swe:field>
<swe:field name="air:temperature">
```

```
<swe:Quantity definition="urn:ogc:def:property:x-istsos:1.0:air:temperature">
<swe:uom code="Degree"/>
</swe:Quantity>
</swe:field>
</swe:DataRecord>
</swe:elementType>
<swe:encoding>
<swe:TextBlock tokenSeparator="," blockSeparator="@" decimalSeparator="."/>
</swe:encoding>
<swe:values>
2014-06-13T11:56:48.006750000+0100,0.01443284005,0.01031082869,-0.9397618771,26.1@
2014-06-13T12:06:48.256750000+0100,0.0226802621,0.01031082869,-0.9437779188,26.2@
2014-06-13T12:16:48.506750000+0100,0.01030912809,0.01855824888,-0.9236976504,26.3@
2014-06-13T12:26:48.756750000+0100,0.01855655015,0.01031082869,-0.9196816087,26.4
</swe:values>
</swe:DataArray>
</om:result>
</om:Observation>
</sos:InsertObservation>
```

7.3 Overview of Communication

The istSOS developers maintains a Google group [2] forum for communication and discussion about the issues, latest development, suggesting and solving bugs and many other possibilities. The group members are very active and accurate about their solution which is really a great help to solve the issues. This section is an overview of the communication between our study group and the istSOS developers will be presented.

Issues [3]: Problem with HTTP POST request

Question (1): I have some problem with XML based HTTP POST request to istSOS server. My insertObservation request looks correct but still I can't insert my observations through HTTP POST request. It says AssignedSensorId parameter is mandatory with multiplicity 1 but I already assigned my sensorId. Could anyone please tell in which part I am wrong or may be my code is wrong. The server error response is as follows:

```
Response status code: 200
Response body:
<?xml version="1.0" encoding="UTF-8" standalone="no"?><ExceptionReport xmlns="http://...
     www.opengis.net/ows/1.1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" ...
     xsi:schemaLocation="http://www.opengis.net/ows/1.1 ../owsExceptionReport.xsd" ...
     version="1.0.0" xml:lang="en">
 <Exception locator="AssignedSensorId" exceptionCode="MissingParameterValue">
  <ExceptionText>
    sos:AssignedSensorId parameter is mandatory with multiplicity 1
  </ExceptionText>
 </Exception>
</ExceptionReport>
```

Solution (1): The error says: $< sos : AssignedSensorId >$ while you submitted $< AssigneSensorId >$. 'sos:' is missing !!!!

Question (2): After fixing the problem, the server response shows another error as follows:

```
<Exception exceptionCode="NoApplicableCode">
  <ExceptionText>
    istSOS internal error
  </ExceptionText>
```

[2] https://groups.google.com/forum/#!forum/istsos
[3] https://groups.google.com/forum/#!topic/istsos/67WM226Y49Q

```
5    <ExceptionText>
6      Please activate debug level for more details
7    </ExceptionText>
8   </Exception>
```

Solution(2): Please take a look in the istsos root folder, there is a file "config.py"

Change the debug value to True

... debug=True ...

Restart Apache and the monitor the standard error log, in linux I use this command: tail -f /var/log/apache/error.log

Then try to execute again the InsertObservation request. If this does not help you to understand what is going on, please cop and paste here the error message, so we can see what is happening, and find a solution.

Question(3): I changed the debug=true and it worked. but another exception shows for ISO8601 date and time. It can not parse my date and time string. I don't know why? My timestamp looks correct. My time stamp look likes: 2014-06-13T07:56:00.006750000+0100 but probably istSOS library can not parse (hyphen) "-" and throws an exception as:

```
1  <ExceptionText>
2      ISO8601Error
3    </ExceptionText>
4    <ExceptionText>
5      Unrecognised ISO 8601 date format: ' 2014-06-13'
6    </ExceptionText>
7    <ExceptionText>
8      ['Traceback (most recent call last):\n', '  File "C:/istsos/application.py", line...
           65, in executeSos\n    response = FR.sosFactoryResponse(req_filter,pgdb)\n',...
           '  File "C:\\istsos\\istsoslib\\responders\\factory_response.py", line 42, ...
          in sosFactoryResponse\n    return IOresponse.InsertObservationResponse(...
          sosFilter,pgdb)\n', '  File "C:\\istsos\\istsoslib\\responders\\IOresponse.py...
          ", line 69, in __init__\n    start = iso.parse_datetime(stime[0])\n', '  File...
           "C:\\istsos\\lib\\isodate\\isodatetime.py", line 49, in parse_datetime\n    ...
          tmpdate = parse_date(datestring)\n', '  File "C:\\istsos\\lib\\isodate\\...
          isodates.py", line 192, in parse_date\n    raise ISO8601Error(\'Unrecognised ...
          ISO 8601 date format: %r\' % datestring)\n', "ISO8601Error: Unrecognised ISO ...
          8601 date format: ' 2014-06-13'\n"]
9    </ExceptionText>
10  </Exception>
11 </ExceptionReport>
```

Solution(3): At the beginning of each date there is a blank character that raises a parsing exception: Unrecognised ISO 8601 date format: ' 2014-06-13'. Remove it and you will solve your problem.

```
1  <swe:values>
2  2015-05-13T07:56:00.006750000+01:00...
       ,0.01443284005,0.01031082869,-0.9397618771,26.37915421@ 2015-05-13T07:58:00...
       .256750000+01:00,0.0226802621,0.01031082869,-0.9437779188,26.37915421@ 2015-05-13...
       T07:59:00.506750000+01:00,0.01030912809,0.01855824888,-0.9236976504,26.37915421
3  </swe:values>
```

8 References

[1] istSOS Project webpage. Link: `https://geoservice.ist.supsi.ch/projects/istsos/index.php/Main_Page`. Last visited: 2015-04-10.

[2] [OGC 2012] Open Geospatial Consortium Inc (2012): OGC Sensor Observation Service Link: `http://www.opengeospatial.org/standards/sos` Last Visited: 2015-04-10

[3] [OGC 2014] Open Geospatial Consortium Inc (2014): OGC Sensor Model Language (SensorML) Link: `http://www.opengeospatial.org/standards/sensorml` Last Visited: 2015-04-10

[4] [OGC 2011] Open Geospatial Consortium Inc (2011): OGC Observation and Measurements Link: `http://www.opengeospatial.org/standards/om` Last Visited: 2015-04-10

9 Attachments

- Portable Data Insertion Manager Solution
- Portable Data Insertion Manager Solution JAVA library
- Web Data Querying Manager source code