FATIMA

FArming Tools for external nutrient Inputs and water MAnagement

# D2.3.3 Configuration and installation guidelines for set of sensor hardware and software components for integration

WP2.3 – WSN for soil, plant, water, and yield monitoring

*Karel Charvat, Uģis Grīnbergs, Zbynek Krivanek, Marek Musil, Michal Kepka (BOSC)*

European Commission

Horizon 2020
European Union funding
for Research & Innovation

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 633945.

## Document Information

| | | | | |
|---|---|---|---|---|
| **Grant Agreement Number** | **633945** | **Acronym** | **FATIMA** | |
| **Full Title of Project** | Farming Tools for external nutrient inputs and water Management | | | |
| **Horizon 2020 Call** | SFS-02a-2014: External nutrient inputs (Research and innovation | | | |
| **Start Date** | 1 March 2015 | Duration | 36 months | |
| **Project website** | www.fatima-h2020.eu | | | |
| **Document URL** | (insert URL if document is publicly available online) | | | |
| **REA Project Officer** | Aneta RYNIAK | | | |
| **Project Coordinator** | Anna Osann | | | |
| **Deliverable** | D2.3.2Integration of WSN Nodes with sensor chips | | | |
| **Work Package** | WP5 | | | |
| **Date of Delivery** | Contractual | 01/03/2016 | Actual | 01/03/2016 |
| **Nature** | R - Report | Dissemination Level | PU | |
| **Lead Beneficiary** | BOSC | | | |
| **Lead Author** | Karel Charvat | Email | Charvat@ccss.cz | |
| **Contributions from** | BOSC (Karel Charvat, UģisGrīnbergs, Zbynek Krivanek, Marek Musil, Michal Kepka) | | | |
| **Internal Reviewer 1** | Rosario Napoli | | | |
| **Internal Reviewer 2** | Juan Manuel Moreno | | | |
| **Objective of document** | The objective of this document is to describe agrometeorological sensors - hardware and software deployment for the purpose of in situ monitoring on farms | | | |
| **Readership/Distribution** | All FATIMA Regional Teams; All WP leaders and other FATIMA team members; European Commission / REA | | | |
| **Keywords** | Guidelines, sensors, WSN, conceptual design, integration | | | |

## Document History

| Version | Issue Date | Stage | Changes | Contributor |
|---|---|---|---|---|
| V0.0 | | | | Karel Charvat, Uģis Grīnbergs |
| V0.2 | | | | Karel Charvat, Uģis Grīnbergs |
| | | | | |
| | | | | |

## Disclaimer



## Copyright

# Executive summary

This report describe both hardware and software integration of FATIMA observation system. It starts from integration of Wireless Sensors nodes and integration of sensors. In next part is described integration of WSN and their connection with Gateways. Two option of Gateways are described. Second part of network is focused on software implementation. Last part of report describes full implementation of Senslog as observation management and also available interfaces.

# Table of contents



# List of Tables

# List of Figures



# 1 WSN nodes integration

Integration of nodes depends on application. In open area (environmental monitoring, agriculture) the node must be fully protected against dust and heavy rain, i.e., the housing must meet at least IP67 class. The housing of node has to be fitted with transparent cover, to locate the light sensor and solar panel inside the housing. Gas-permeable membrane filter ventilation system must be installed on the housing to provide the same atmospheric pressure, air temperature and humidity of the outside. It gives a chance to fit environmental monitoring sensors inside the housing. This would allow to place part of sensors within the housing, protecting them from exposure of the environment and reduce the cost of nodes. The junction box is to be used for connecting the external sensors without opening the housing of the node. In this project four channel analog switch is located in the junction box because of large capacity of connected 2-wire air temperature/humidity sensors. The cable between junction box and housing of the node has to be pressurized with cable glands. The node is to be equipped with non-contact switch, for example, permanent magnet steered reed, which can be turned on and off. The mechanical design of node has to provide user-friendly and simple fixation on the field.

Overview of the whole structure of the wireless sensor network is shown on diagram on figure 1 below. Individual parts of the chain are described in following chapters. For the integration of sensors with Wireless Sensor Node was necessary to develop sets of toolkits supporting connection of different types of sensors to node. The Figure 1 describe, how to connect sensors to WSN Hardware

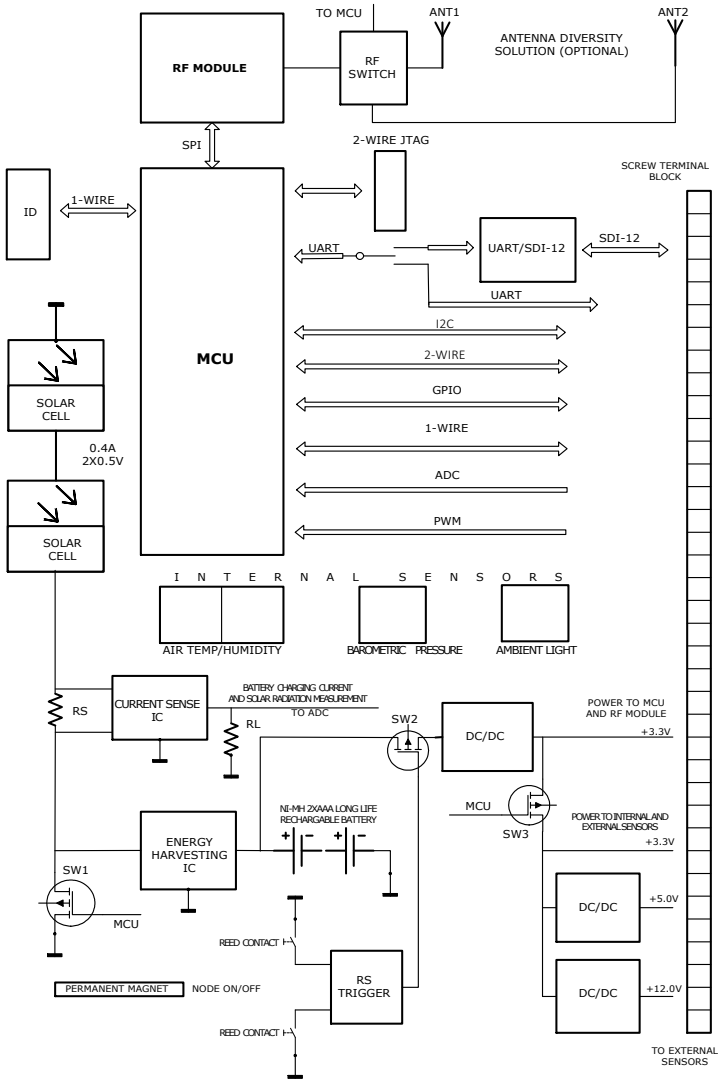


*Figure 1 Connection of sensors with nodes*

Typical application contains processor board, some type of connectivity board, one or more sensor boards and optionally power board.

## 1.1 WSN node description:

*Node consists of some general parts - host MCU with all the necessary interfaces to sensors, radio module, power management circuits with energy harvesting, including solar panel placed inside the housing of node, internal and externals sensors.*

### 1.1.1 Host MCU.

MSP430F2272 from TI with operating system FARM OS was chosen as host MCU for node. Host MCU provides following interfaces for sensors:

- UART;
- SDI-12;
- I2C;
- 1-wire;
- 2-wire Sensirion with multiplexing by 4;
- ADC;
- GPIO;

Host MCU provides SPI interface for radio module, 1-wire interface for ID (MAC address) chip

### 1.1.2 RF module

RF module include

- Radio module is the key element of WSN node. It's a main module determining quality of the network. RF module based on ST Microelectronics SPIRIT1 transceiver IC and Microchip PIC16LF1938 MCU from IQRF company for WSN prototype has been chosen. Radio module has implemented operating system with WSN mesh network protocol stack from IQRF.
- Internal sensors:
- air temperature/humidity measurement with dew point calculation possibility - the Sensirion IC SHT21S with analogue voltage interface;
- barometric pressure – the Freescale Semiconductor IC MPXH6115A6 with analogue voltage interface;
- ambient light – the ROHM IC BH1603FVC with analogue current interface;

### 1.1.3 External sensors

As external sensors are used

- 4 air temperature/humidity measurement probes –_SNS_THE from PAPOUCH with Sensirion 2-wire interface;
- 1 temperature probe – _SNS_TEMP from PAPOUCH with 1-wire interface;
- INSPEED VORTEX WIND SPEED SENSOR with reed contact output interface (frequency of reed contact switching is measured);
- 1 INSPEED E-VANE Hall effect wind direction sensor with analogue voltage interface.

## 1.2 WSN deployment

To complete tasks determined by application it is crucial to choose optimal frequency band, channel bandwidth, data rate, and modulation technology. As it is mentioned above to perform a measurement of one phenomenon 6 B (bytes) of data are necessary; minimal time interval between measurements is considered 60s for meteorological measurements for agriculture use.

According to European table of frequency allocations and applications frequency range of 863-870 MHz is planned for use of Short Range Devices (SRD). According to ERC recommendation 70-03 relating to the use of SRD devices the maximum transmitting power must be not greater than 25 mW (+14dBm), maximum duty cycle for transmitter is 1%. It follows from these specifications- if the time interval between measurements is 60s, the transmitter should not be in active mode more than 600ms. Assume, the sensor node measures 8 phenomena – air temperature, humidity, barometric pressure, solar radiation, soil moisture, soil temperature, battery voltage, battery charging current. The common amount of data sent for one measurement cycle to access point, will be 8X6=48B (bytes) or 48X8=384b (bit). To fit in 600ms necessary data transfer rate will be 384bps/0.6s=640bps. Higher data rate requires larger radio

channel bandwidth for receiver. According to the Nyquist theorem power of thermal noise applied to the receiver input is proportional to the bandwidth of the channel. The way to increase data transfer rate is by reducing link budget. By increasing data transfer rate data transmitting time and duty cycle decreases hence energy consumption of node goes down. In forested areas multi path fading takes place. Implementing of antenna diversity for forested areas is a solution.

WSN deployment should be done according to specifications for necessary data gathering. Before deployment nodes would be programmed in service centers. Supported network topologies are supported by WSN operating system (firmware).

For standard applications on the field configuration wouldn't be necessary - nodes would provide auto configuration together with resources implemented in DW software.

# 2 Gateway configuration and deployment

Main configuration of the system is determined practically based on configuration of the GateWay. Configuration and deployment includes configuration, software and also confection to solar panels:

Hardware configuration

- Power supply board
- STM32F207 processor board
- Connectivity board GPRS (can be varied with ETH, Wi-Fi, Iridium)
- WSN node board (IQRF) with coordinator node

Boards are stacked in sandwich structure interconnected by internal 50pin connector and placed in plastic box. GWL/Power 20W solar panel is chosen for the gateway.

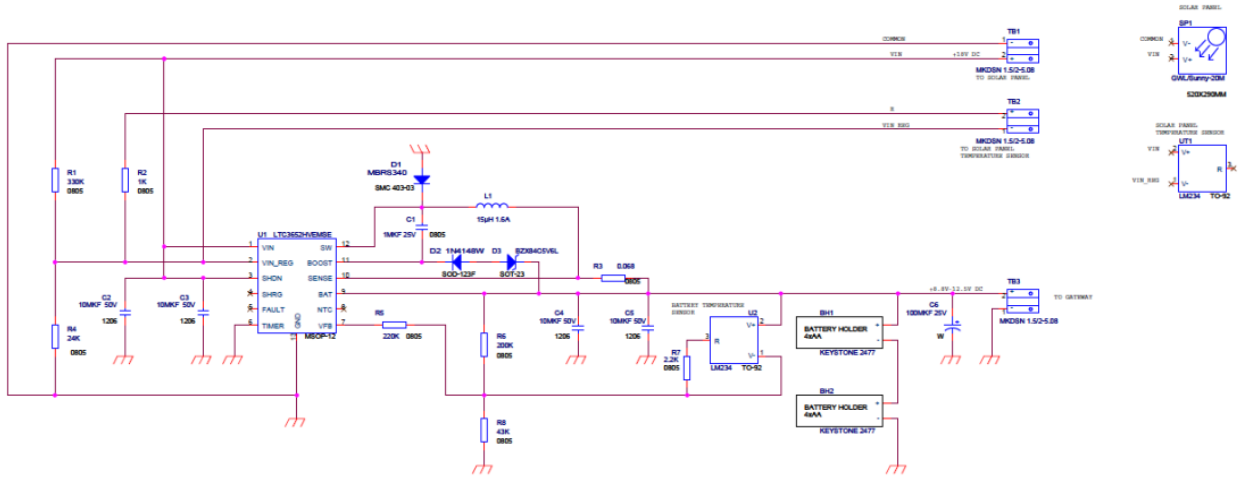

*Figure 2 Gateway Solar power supply schematics*

Configuration of the WSN nodes is very simple and easy. Self configuration is set up for all nodes in the ending stage of manufacturing process. Network topology is determined by necessity gathered by nodes during the first switch on by placement on the field. Each node determines its neighboring nodes and path availability to the Gate Way.

## 2.1 Software

Gateway uses MORT software OS based on ChibiOS with sensor driver for WSN (IQRF modules).

### 2.1.1 Configuration

Unit can be traced and set by asynchronous serial debug port using 9600bits/s, 8N1 no flow control, V24 levels settings.Serial line can run in two modes, debug and command line.Debug mode is set within device start. In this mode unit prints its status and informational messaged to the serial lineCommand line mode is entered once user send space ' ' character to the serial line. Device answers with command prompt 'cmd>' and becomes ready to issue user commands.User can switch device back to debug mode by exit command.If user is inactive for more than 1 minutes, gateway switches automatically to debug mode.System variables are string constants saved in eeprom section of the main processor memory. Constants form is <name_of_the_variable>=<variable_value>. These constants are use to set up different parameters of the system. User can arbitrarily add system variables. Name of the variable is always unique.To list, add or change variables values commands set, get and env are used.

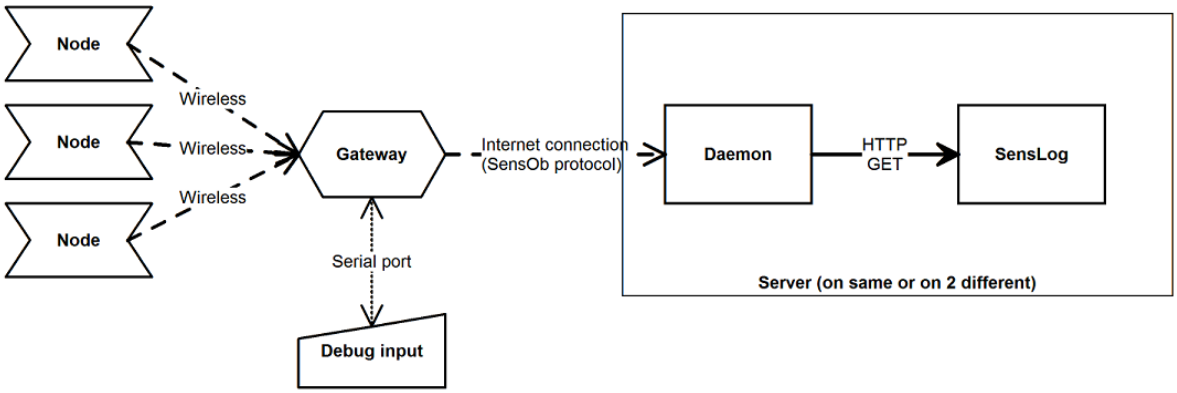


*Figure 3 Integration diagram of the wireless sensor network.*

Mandatory variables:

These variables are required. They are set after facset command is issued.

- server - ip address of data server. To this ip are all measured data send.default: 88.86.113.233
- portport on which receivers run on the server.default: 4252
- timeoutthis is delay in seconds between each attempt to deliver data to server Default: 10
- idid of unit. Herewith number unit identifies itself on the server.default: 1001
- APN used by modem to connect to the provider network. Usually internet.default: internet
- iqrf_debugwhen non zero, gateway prints binary communication with IQRF coordinatorDefault: 0

Additional variables:

- s_<unique_no>_sensdDefines new measurement and inserts it in measurement list.value: <sensor_dev_name>;<sensor_id>;<phenomenon_id>;<logical_address>;<timeout>;<phenomenon_name>;<list> sensor_device is name of device driver which use to communicate with sensor. For
- wsn network use iqrf0
- sensor_id id for this sensorphenomenon is number which defines phenomenon to measure. In SDI12 this number corresponds to position of measurement in D command.
- logical_address address in sensor network. As network can contain subnetwork (SDI12 wired network), address of subnetwork is shifted and combined. See example – SDI12 adress is 0x03 and wsn address is 0x02
- timeout delay between two consequent measurements from this sensor, in seconds. If this sensor just gives its data to filter, timeout must be 0
- phenomenon_name is user name of phenomenon, which will be used for displa
- list defines list in which will be this measurement enlisted.
- example: set s_00_sensd=iqrf0;690020000;20;0x0302;300;5tm_die;send
- s_<unique_no>_sensfDefines new filter and inserts it in measurement list.value:<filter_dev_name>;<filter_id>;<timeout>;<source_sensor_filter_id>;<parameters>;<phenomenon_name>;<list> filter_dev_name is name of filter driver to use filter_id id for this filter
- timeout delay between two consequent measurements from this filter, in seconds. If this filter just gives its data to another filter (chain), timeout must be 0 source_sensor_filter_id is name of variable, which defines data source for this filter
- parameters filter specific parameters, up to 8 comma separated float values
- phenomenon_name is user name of phenomenon, which will be used for display
- list defines list in which will be this measurement enlisted. See chapter Sensors, filters, lists

### 2.1.2 Receiving demon configuration

Daemon is linux native aplication server capable of receiving telemetry data from different type of sources, transform data content to unified form and send them via http get protocol to database feeder or other application

Configuration file has standard Linux shape (see libconfuse library documentation) and must be placed in the same directory as feeder receiver binary or in /etc.

Available parameters:

- logfile

  defines location and name for logging file. In this file all debug, warning and error messages are written. Study this file first if something goes wrong.

- serverdelay

  if http request was not successful, application tries to retransmit request periodically. This is delay between each try. Values is in seconds.

- db

  database feeder parameters, can be defined more than once.

- server

location (ip or address) of database feeder. If on same machine use localhost.

- port

  TCP port on which database feeder receives requests, default is 8180.

- uri

  uri of database feeder, default is /FeederServlet

- type

  measurement unit parameters, can be defined more than once.

- lib

  name and location of parsing library.

- port

  port number on which application should listen.

- proto

  network protocol. Can be PROTO_UDP for UDP transfer, PROTO_TCP_PAS for TCP passive (application acts as server) type of socket, PROTO_TCP_ACT for TCP active (application acts as client) type of socket.

- server

  server address from which application should grab data. Valid only for PROTO_TCP_ACT, otherwise NULL.

- timeout

  delay between each data grab from server. Valid only for PROTO_TCP_ACT, otherwise 0.

Example of configuration file:

```
logfile="receiver_log.txt"
serverdelay=1

db sensors_backup {
   server="localhost"
   port=8180
   uri="/FeederServlet"
}
db sensors {
   server="test.sensors.lesprojekt.cz"
   port=8180
   uri="/FeederServlet"
}
```

```
type sensob {
   lib="./sensob.so"
   port=4252
   proto="PROTO_UDP"
   server=NULL
   timeout=0
}
```

### 2.1.3 User data access

As it explained in paragraph 2 WSN nodes provides self configuration with each other and establish route to the Gate Way (GW) in real Plug and Play mode. End user just need to chose placement where measurements have to be done.

Both GW MORT and Raspberry Pi based could provide data transport to the same data accumulating data base structure. The data “catcher” or listener is daemon located on the dedicated database server usualy cariing name ***SensLog***. (paragraph X)

Thi Linux runned service ***SensLog*** is installed by service company. Its functionality is listening to particular socket for data arrival. When data packet is recognised and received the data is being moved to appropriate data base server (Postgre SQL) and placed in data base tables with predefined structure (Figure 3).

The stored data further can be accessed using one of graphical user interfaces already. The front end application provides data view as a www service. For local or even worldwide data access graphical interface is intended to be located on the internet resource with particular IP address on port 80.


*Figure 4 Graphical representation of sensor nodes on the field.*

As a base for graphical interface Open Google maps service is used different colors of the nodes shows different functionality or status of the nodes. Red means that some service or user interaction for particular node is necessary.

Graphical front end application provides all necessary forms for data representation according to particular user requirements.

Short term historical data could be represented as a chart giving clear view for development of meteorological or other processes on the field.

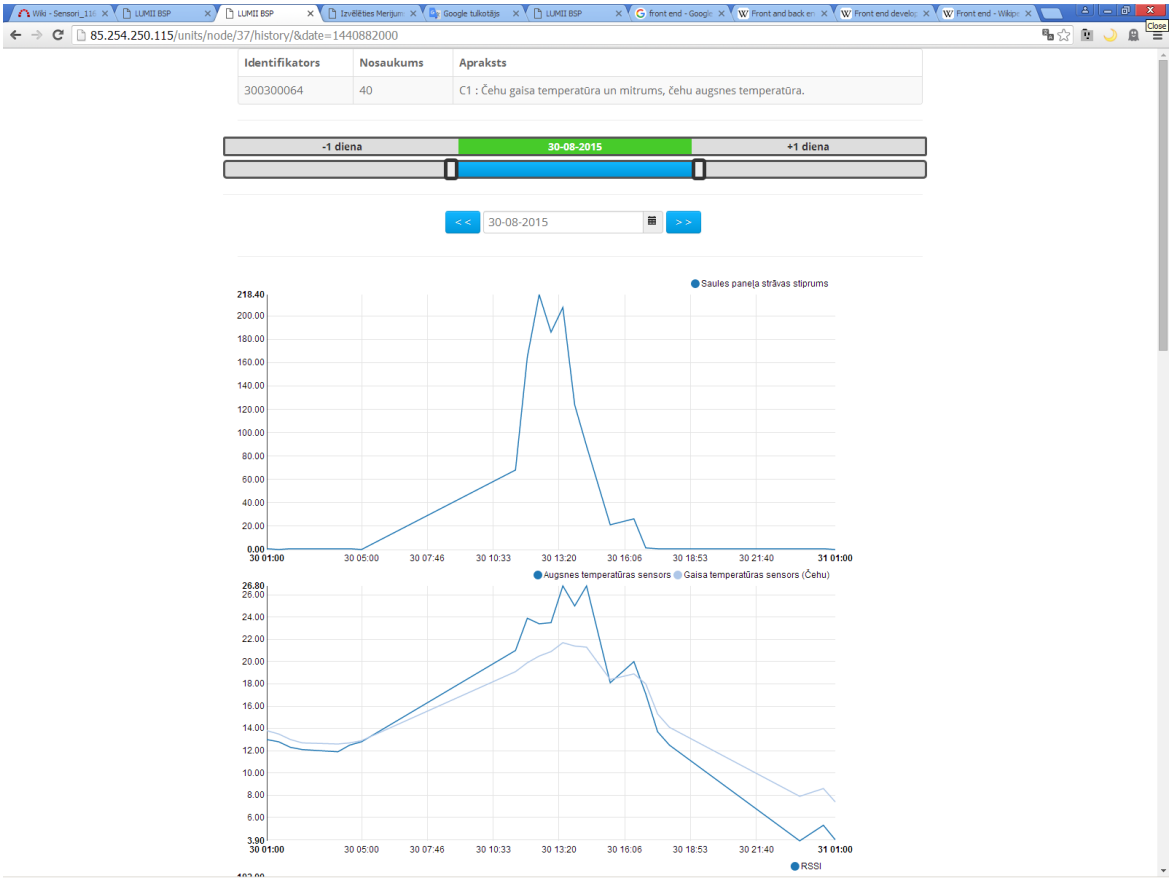


*Figure 5 Graphical representation of measured Solar panel current (char above) and both air and crop foliage temperature during the particular day.*

## 2.2 Alternate GW built using Raspberry PI 2 Model B

Gateway is connected to IQRF WSN[1] network using CK-USB-04A[2] USB programmer and debugger.

Ethernet or GPRS/GSM module could be chosen for communication over the internet. Raspberry Pi 2 Model B specifications:

- 900MHz quad-core ARM Cortex-A7 CPU
- 1GB RAM
- 4 USB ports
- 40 GPIO pins
- Full HDMI port
- Ethernet port
- Combined 3.5mm audio jack and composite video
- Camera interface (CSI)
- Display interface (DSI)
- Micro SD card slot
- VideoCore IV 3D graphics core
- 5.0 V voltage

### 2.2.1 Software

Raspbian operating system based on Debian is used.For data acquisition from WSN and sending to servers Ruby programming language is used. The operating system recognizes CK-USB-04A as a standard serial device. Ruby scripts communicate with the programmer using a protocol provided by IQRF.Because Raspberry GW is intended with use with sensob application instead of MORT GW settings for Raspberry GW is exactly the same as it is described for MORT GW. The gateway software runs as a Linux service.

# 3 SensLog

## 3.1 SensLog description

SensLog is a software component for receiving, storing, analyzing and publishing sensor data in various interfaces. SensLog receives sensor data in form of HTTP GET requests and stores them to PostgreSQL database with PostGIS extension. SensLog publishes data using proprietary RESTful API version 1.0 and using standardized OGC Sensor Observation Service version 1.0.0. Simple diagram is shown on figure 3 below. SensLog contains also simple GUI with several functions to show data, it is supposed to be used in combination with other clients. Next paragraphs are describing structure of the application from database to server-side part.

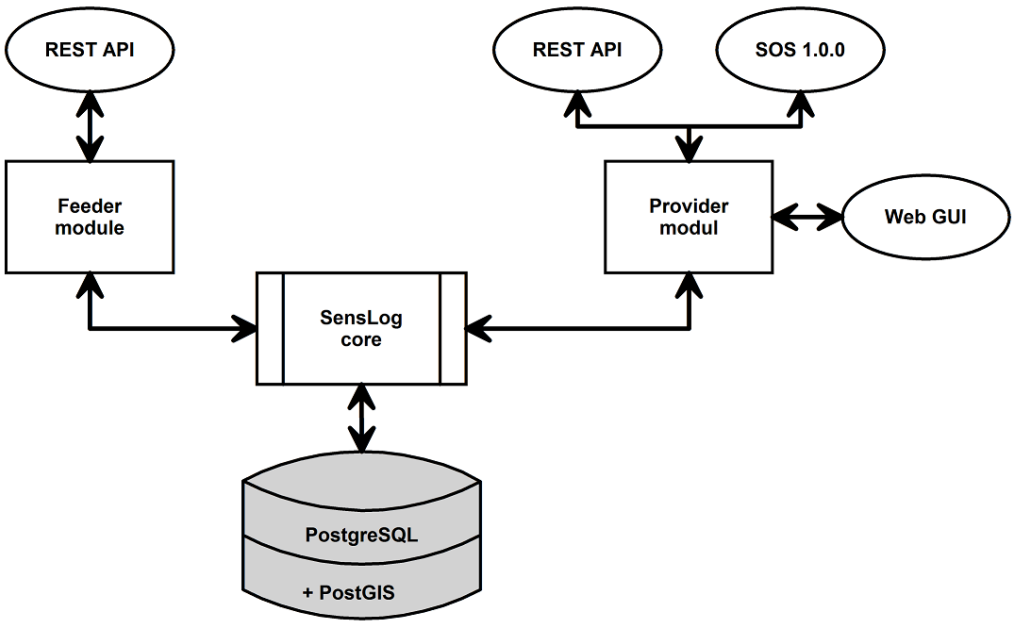


*Figure 6 Overview of SensLog structure*

SensLog consists of database model and server-side application. The core of the database part is a relational data model that was inspired by standardized data model for observations from specification OGC Observations&Measurements version 1.0. SensLog model implements more functionality contrary to O&M model that has been necessary in created sensor networks. Data model is shown on figure 3 below.

There are several core tables that hold structure of the sensor network and user hierarchy. Units are main element of the sensor network and are stored in units table. Unit is physical node with one or more connected sensors. Positions of observations are connected to whole unit. Sensors are physical devices that are measuring phenomena. In the used data model one sensor is measuring one phenomenon. In case of measuring more phenomenon with one sensor device, it is necessary to decompose the device to several virtual sensors. Observations represent values measured by each unit and sensor pair in specified time stamp and at specified position. The unique observation is determined by time stamp, ID of unit and ID of sensor that have produced the observed value. Unit's positions table holds location of units in defined coordinate reference system. A hierarchy of users and units can be created through Groups table. It allows users to access only units that are in same group or subordinate groups.
Database model contains procedures and functions to process and manipulate with data in transactional form. Especially several triggers prevents data integrity.

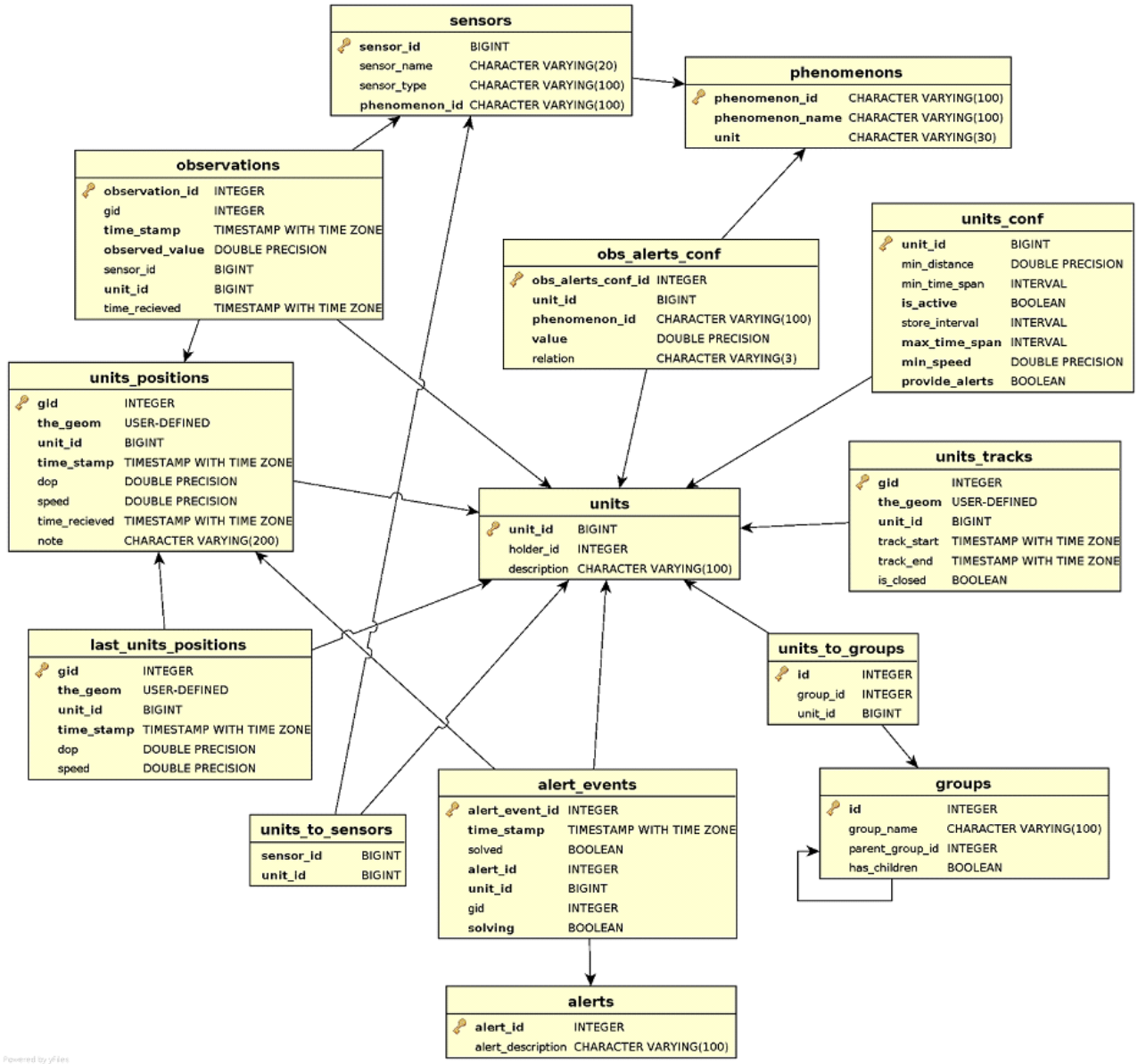


*Figure 7 SensLog database model*

Server-side part of the application contains most of the application logic. Diagram on figure 4 shows simple schema of the server-side part of SensLog. Sensor data are pushed to SensLog from sensor networks through RESTful API from Gateways. Receiver module contains check mechanism to prevent insertion of data in incorrect form. Service for inserting data is described in chapter SensLog interface.

Data consumers can receive data by one of the interfaces after authentication. There are several services to provide not only measured data or status data of the network but also structure of sensor network itself.

Users can use Web GUI that allows visualization of measured data in form of simple charts and maps. Main access to sensor data is provided by RESTful or SOS interface and it is expected further processing in other clients or applications.

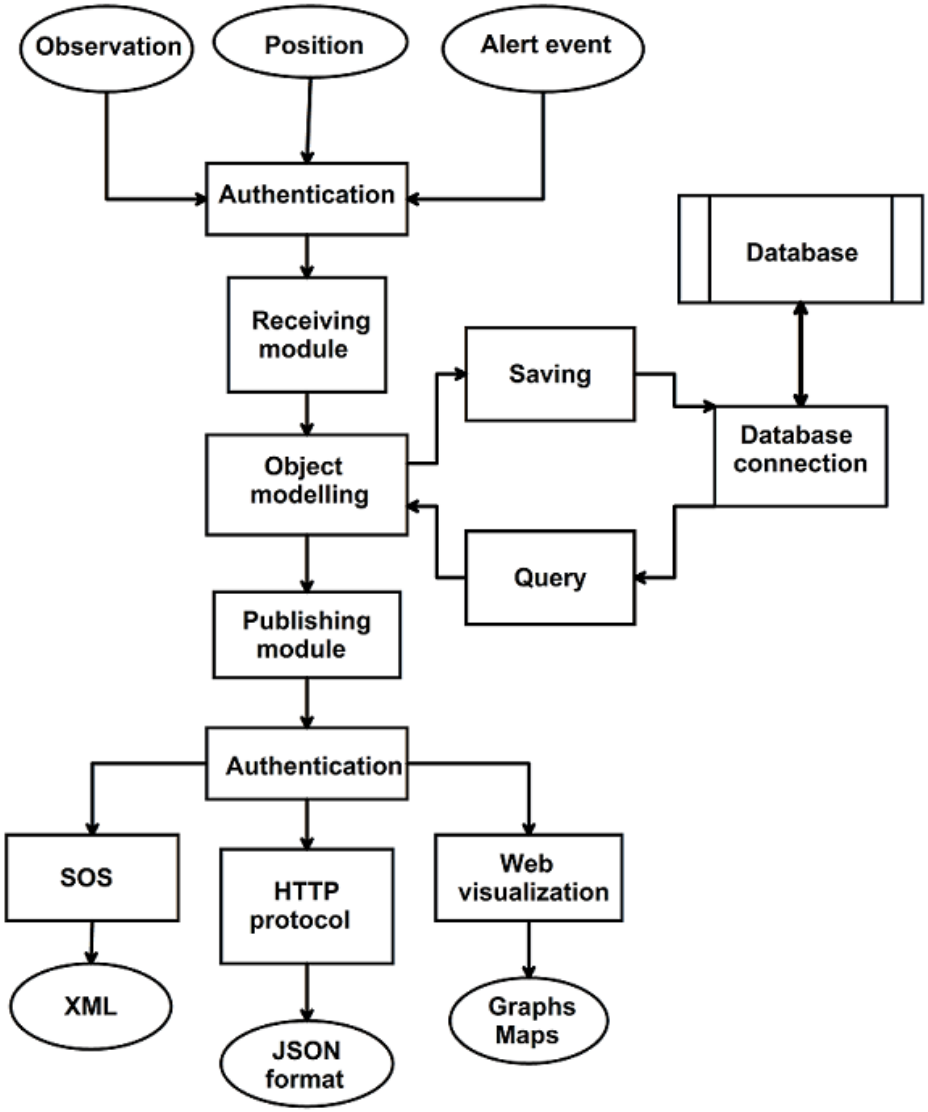


*Figure 8Schema of server-side part*

## 3.2 SensLog installation

SensLog runs as web application in Apache Tomcat and needs running PostgreSQL database.

Installation can be summarized in several points:

1. Installation of PostgreSQL with PostGIS extension

It is necessary to install last version of PostgreSQL DBMS and install PostGIS spatial extension. There is necessary to add enough empty disc capacity for storing sensor data and open port for accessing the DBMS from application. Last version of SensLog was tested with PostgreSQL 9.4 and PostGIS 2.1.

2. Building of complete database model

There is necessary to create empty database at installed PostgreSQL and create user account that will be associated with SensLog application. The database model will be constructed by SQL scripts provided with application installation. SQL scripts create all tables with relations and PL/SQL procedures and functions. After model construction there is necessary to initialize database model, that means create at least one system user and one corresponding group.

3. Installation of Apache Tomcat server

There is necessary to install Apache Tomcat on server and create one user that will be able to deploy web applications. Last version of SensLog was tested to be running with Tomcat version 7.0.x.

4. Configuration of SensLog instance

Configuration of SensLog consists of modifying configurations files according to templates. It means insert credentials for database connection and specify details of database model if it is necessary.

5. Deployment of SensLog instance to Tomcat

Deployment can be done by uploading war file to Tomcat instance or by deploying by Maven from source code. There is necessary to have rights to upload new applications to Tomcat.

Before starting of data collecting from sensor network it is necessary to insert into database model structure of sensor network and users' hierarchy. This can be done by direct inserting into database model or by services via API.

## 3.3 SensLog interfaces

SOS provides access to observations from sensors and sensor systems in a standard way. The same way is suitable for any type of sensor systems. It could be remote sensing, in-situ, fixed and mobile sensors. SOS leverages the O&M specification for modelling observations and the TML and SensorML specifications for modelling sensors and sensor systems. SOS is primarily designed to provide access to observations. The SensLog is mainly focused on publication of observations in standard form for consumers of observations.

### 3.3.1 SOS Core operations

Implemented SOS contains mandatory operations:

- GetCapabilities – provides the main access to SOS service metadata. It contains description of service identification, service provider, list of operations, filter capabilities and list of observations offerings. Example of the Capabilities document is shown of figure 5 below.
- GetObservation – provides access to sensor observations and measurement data, a spatio-temporal query filtered by phenomena can be used. It contains identification of time period and text block with sensor data. Example of ObservationCollection is shown on figure 6 below.
- DescribeSensor – retrieves detailed information about the sensors and processes generating those measurements. In context of SensLog operation DescribeSensor provides metadata about the whole unit with individual sensors as Outputs. Example of SensorML document is shown on figure 7 below.

All request and response messages are XML documents with structure and elements defined by OGC standard document. Schemas of the XML documents are part of OGC Schemas Repository.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns4:Capabilities xmlns="http://www.opengis.net/ogc" xmlns:ns2="http://www.opengis.net/gm
  <ns10:ServiceIdentification>
  <ns10:ServiceProvider>
  <ns10:OperationsMetadata>
  <ns4:Filter_Capabilities>
  <ns4:Contents>
    <ns4:ObservationOfferingList>
      <ns4:ObservationOffering ns2:id="admin">
        <ns2:name>admin</ns2:name>
        <ns2:boundedBy>
          <ns2:Envelope srsName="urn:ogc:def:crs:EPSG:4326">
            <ns2:lowerCorner>16.6147785186768 49.0214881896973</ns2:lowerCorner>
            <ns2:upperCorner>16.615421295166 49.0239219665527</ns2:upperCorner>
          </ns2:Envelope>
        </ns2:boundedBy>
        <ns4:time ns3:type="simple">
          <ns2:_TimeGeometricPrimitive xsi:type="ns2:TimePeriodType" frame="ISO-8601">
            <ns2:beginPosition>2010-04-27T06:47:06+02:00</ns2:beginPosition>
            <ns2:endPosition>2011-05-05T18:27:32+02:00</ns2:endPosition>
          </ns2:_TimeGeometricPrimitive>
        </ns4:time>
        <ns4:procedure ns3:href="0"/>
        <ns4:procedure ns3:href="2"/>
        <ns4:procedure ns3:href="3"/>
        <ns4:procedure ns3:href="4"/>
        <ns4:procedure ns3:href="32768"/>
        <ns4:procedure ns3:href="32769"/>
        <ns4:procedure ns3:href="32770"/>
        <ns4:procedure ns3:href="32771"/>
        <ns4:procedure ns3:href="32772"/>
        <ns4:procedure ns3:href="32773"/>
        <ns4:procedure ns3:href="10132866067"/>
        <ns4:procedure ns3:href="10414158052"/>
        <ns4:procedure ns3:href="101020551187"/>
        <ns4:procedure ns3:href="101471635475"/>
        <ns4:procedure ns3:href="101502765075"/>
        <ns4:procedure ns3:href="103653328915"/>
        <ns4:procedure ns3:href="103852820499"/>
        <ns4:procedure ns3:href="103975897107"/>
        <ns4:observedProperty ns3:href="Count" ns3:type="simple"/>
        <ns4:observedProperty ns3:href="Dew point" ns3:type="simple"/>
        <ns4:observedProperty ns3:href="Generated - TODO" ns3:type="simple"/>
        <ns4:observedProperty ns3:href="Humidity" ns3:type="simple"/>
        <ns4:observedProperty ns3:href="Moisture" ns3:type="simple"/>
        <ns4:observedProperty ns3:href="retransmission count" ns3:type="simple"/>
        <ns4:observedProperty ns3:href="Rssi" ns3:type="simple"/>
        <ns4:observedProperty ns3:href="Temperature" ns3:type="simple"/>
        <ns4:observedProperty ns3:href="Voltage" ns3:type="simple"/>
        <ns4:featureOfInterest ns3:href="fOi"/>
        <ns4:responseFormat>text/xml;subtype="om/1.0.0"</ns4:responseFormat>
        <ns4:resultModel>ns5:Observation</ns4:resultModel>
        <ns4:responseMode>inline</ns4:responseMode>
      </ns4:ObservationOffering>
    </ns4:ObservationOfferingList>
  </ns4:Contents>
</ns4:Capabilities>
```

*Figure 9 Example of Capabilities document*

```
  xmlns:ns10="http://www.opengis.net/ows/1.1" xmlns:ns11="http://www.w3.org/2001/SMIL20/Language"
  xmlns:ns2="http://www.opengis.net/sos/1.0" xmlns:ns3="http://www.opengis.net/gml"
  xmlns:ns4="http://www.w3.org/1999/xlink" xmlns:ns6="http://www.opengis.net/swe/1.0.1"
  xmlns:ns7="http://www.opengis.net/sensorML/1.0.1" xmlns:ns8="urn:us:gov:ic:ism:v2"
  xmlns:ns9="http://www.w3.org/2001/SMIL20/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
- xsi:schemaLocation="http://www.opengis.net/om/1.0 http://schemas.opengis.net/om/1.0.0/om.xsd">
-   <ns5:member ns4:type="simple">
-     <ns5:Observation>
-       <ns5:samplingTime ns4:type="simple">
-         <ns3:TimePeriod>
           <ns3:beginPosition>2011-06-20T08:09:53+02:00</ns3:beginPosition>
           <ns3:endPosition>2011-06-20T11:48:36+02:00</ns3:endPosition>
         </ns3:TimePeriod>
       </ns5:samplingTime>
       <ns5:procedure ns4:href="32768" ns4:type="simple" />
       <ns5:observedProperty ns4:type="simple" />
       <ns5:featureOfInterest ns4:href="urn:ogc:def:feature:OGC-SWE:3:transient" />
-      <ns5:result>
-        <swe:elementCount xmlns:swe="http://www.opengis.net/swe/1.0.1">
-          <swe:Count>
             <swe:value>19</swe:value>
           </swe:Count>
         </swe:elementCount>
-        <swe:elementType xmlns:swe="http://www.opengis.net/swe/1.0.1" name="Components">
-          <swe:SimpleDataRecord>
-            <swe:field name="Time">
               <swe:Time definition="urn:ogc:data:time:iso8601" />
             </swe:field>
-            <swe:field name="Accum growing degree">
-              <swe:Quantity definition="Temperature">
                 <swe:uom code="C" />
               </swe:Quantity>
             </swe:field>
-            <swe:field name="Eko air temp">
-              <swe:Quantity definition="Temperature">
                 <swe:uom code="C" />
               </swe:Quantity>
             </swe:field>
           </swe:SimpleDataRecord>
         </swe:elementType>
-        <swe:encoding xmlns:swe="http://www.opengis.net/swe/1.0.1">
           <swe:TextBlock blockSeparator="@" decimalSeparator="." tokenSeparator=";" />
         </swe:encoding>
-        <swe:values xmlns:swe="http://www.opengis.net/swe/1.0.1">
          2011-06-20T08:09:53+02:00;14.05@2011-06-20T08:10:53+02:00;2184.44166974999@2011-06-20T08:25:30+02:00;14.64@
          2011-06-20T08:41:08+02:00;15.01@2011-06-20T08:56:53+02:00;15.01@2011-06-20T09:10:53+02:00;2192.32833641666@
          2011-06-20T09:12:23+02:00;15.01@2011-06-20T09:28:00+02:00;15.94@2011-06-20T09:43:37+02:00;15.77@
          2011-06-20T09:59:16+02:00;16.219999@2011-06-20T10:10:53+02:00;2201.06333616666@2011-06-20T10:14:51+02:00;16.5@
          2011-06-20T10:30:31+02:00;17.9@2011-06-20T10:46:06+02:00;21.870001@2011-06-20T11:01:45+02:00;21.58@
          2011-06-20T11:10:53+02:00;2213.52583641666@2011-06-20T11:17:22+02:00;19.84@2011-06-20T11:33:00+02:00;20.959999@
          2011-06-20T11:48:36+02:00;20.9
          </swe:values>
       </ns5:result>
     </ns5:Observation>
   </ns5:member>
 </ns5:ObservationCollection>
```

*Figure 10 Example of ObservationCollection document*

```
xmlns:ns3="http://www.opengis.net/gml" xmlns:ns4="http://www.w3.org/1999/xlink"
xmlns:ns5="http://www.opengis.net/om/1.0" xmlns:ns6="http://www.opengis.net/swe/1.0.1"
xmlns:ns8="urn:us:gov:ic:ism:v2" xmlns:ns9="http://www.w3.org/2001/SMIL20/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1.0.1"
xsi:schemaLocation="http://www.opengis.net/sensorML/1.0.1 http://schemas.opengis.net/sensorML/1.0.1/sensorML.xsd">
  <ns7:member ns4:type="simple">
    <ns7:System>
      <ns7:identification ns4:type="simple">
        <ns7:IdentifierList>
          <ns7:identifier>
            <ns7:Term definition="unit_id">
              <ns7:value>101502765075</ns7:value>
            </ns7:Term>
          </ns7:identifier>
          <ns7:identifier>
        </ns7:IdentifierList>
      </ns7:identification>
      <ns7:capabilities ns4:type="simple">
      <ns7:outputs ns4:type="simple">
        <ns7:OutputList>
          <ns7:output ns4:type="simple">
            <ns6:Quantity definition="Temperature">
              <ns3:metaDataProperty>
                <ns6:TimeRange>
                  <ns6:value>2010-06-23T18:50:04+02:00 2011-06-09T06:12:11+02:00</ns6:value>
                </ns6:TimeRange>
              </ns3:metaDataProperty>
              <ns3:description>air temperature Papouch TH2E</ns3:description>
              <ns6:uom ns4:type="simple" code="C" />
            </ns6:Quantity>
          </ns7:output>
          <ns7:output ns4:type="simple">
          <ns7:output ns4:type="simple" name="Vlit volt 200">
          <ns7:output ns4:type="simple">
          <ns7:output ns4:type="simple" name="Rssi 200">
          <ns7:output ns4:type="simple">
          <ns7:output ns4:type="simple">
        </ns7:OutputList>
      </ns7:outputs>
      <ns7:positions ns4:type="simple">
        <ns7:PositionList>
          <ns7:position ns4:type="simple" name="Last unit position">
            <ns6:Position>
              <ns6:location ns4:type="simple">
                <ns6:Vector referenceFrame="EPSG:4326">
                  <ns6:coordinate name="longitude">
                  <ns6:coordinate name="latitude">
                </ns6:Vector>
              </ns6:location>
            </ns6:Position>
          </ns7:position>
        </ns7:PositionList>
      </ns7:positions>
    </ns7:System>
  </ns7:member>
</ns7:SensorML>
```

*Figure 11 Example of SensorML document*

### 3.3.2 SensLog RESTful API v1

SensLog RESTful API version 1.0 provides methods to insert sensor data from sensor network and provides services to get metadata about sensor units, user groups and to get observations, alerts and positions of sensors units.

#### 3.3.2.1 FeederServlet

| **URL:** /FeederServlet?Operation=<operationName> | | |
|---|---|---|
| ***Method*** | ***Functionality*** | ***Format*** |
| GET | Provides insertion of sensor data to the database | URL encoded plain text |

*Table 1 Feeder services*

InsertObservation

| **URL:** /FeederServlet?Operation=InsertObservation | | |
|---|---|---|
| ***Method*** | ***Functionality*** | ***Format*** |
| GET | Provides inserting of new observation for combination of sensor and unit into database. Return true if observation was successfully inserted, false in all other cases. | URL encoded plain text |

| ***Parameters*** | ***Format*** | ***Role*** |
|---|---|---|
| value | Double | Measured value (mandatory) |
| date | Time stamp (ISO 8601) | Time stamp of measured value (e.g. *2015-07-15 12:00:00+0200*) (mandatory) |
| unit_id | Numerical value | Identifier of unit (mandatory) |
| sensor_id | Numerical value | Identifier of sensor (mandatory) |

*Table 2 InsertObservation service*

InsertPosition

| **URL:** /FeederServlet?Operation=InsertPosition | | |
|---|---|---|
| ***Method*** | ***Functionality*** | ***Format*** |
| GET | Provides inserting of new position of the unit into database, position can be inserted only in WGS-84 coordinates. Return true if position was successfully inserted, false in all other cases. | URL encoded plain text |

| ***Parameters*** | ***Format*** | ***Role*** |
|---|---|---|
| lat | Double | Latitude (mandatory) |
| lon | Double | Longitude (mandatory) |
| alt | Double | Altitude in meters (optional) |
| speed | Double | Speed of the unit (optional) |
| unit_id | Numerical value | Identifier of unit (mandatory) |
| date | Timestamp (ISO 8601) | Timestamp of measured value (e.g. *2015-07-15 12:00:00+0200*) (mandatory) |

| | | |
|---|---|---|
| dop | Numerical value | Dilution of precision (optional) |

*Table 3 InsertPosition service*

### InsertAlertEvent

| **URL:** /FeederServlet?Operation=InsertAlertEvent | | |
|---|---|---|
| ***Method*** | ***Functionality*** | ***Format*** |
| GET | Provides inserting of new alert event that was detected in the sensor network into database, Return true if event was successfully inserted, false in all other cases. | URL encoded plain text |

| ***Parameters*** | ***Format*** | ***Role*** |
|---|---|---|
| alert_id | Numerical value | Identifier of alert (mandatory) |
| unit_id | Numerical value | Identifier of unit (mandatory) |
| date | Timestamp (ISO 8601) | Timestamp of measured value (e.g. *2015-07-15 12:00:00+0200*) (mandatory) |

*Table 4 InsertAlertEvent service*

#### 3.3.2.2 Data services

| **URL:** /DataService?Operation=<operationName> | | |
|---|---|---|
| ***Method*** | ***Functionality*** | ***Format*** |
| GET | Provides detailed information about sensor units | JSON |

*Table 5 Data service*

### GetUnits

| **URL:** /DataService?Operation=GetUnits | | |
|---|---|---|
| ***Method*** | ***Functionality*** | ***Format*** |
| GET | Provides detailed information about each units connected to login user. Response contains connected sensors, first and last time stamp of entered observation, last positions of unis. | JSON |

*Table 6 GetUnits service*

### GetPositions

| URL: /DataService?Operation=GetPositions&user=<username>&limit=<limit> | | |
|---|---|---|
| ***Method*** | ***Functionality*** | ***Format*** |
| GET | Request provides users specified number of last positions of all units in current group. | JSON |

| ***Parameters*** | ***Format*** | ***Role*** |
|---|---|---|

| | | |
|---|---|---|
| user | Text | Identifier of user group |
| limit | Numeric value | Number of positions to receive |

*Table 7 GetPositions service*

### GetLastPosition

| **URL:** /DataService?Operation=GetLastPosition&user=<username> | | |
|---|---|---|
| ***Method*** | ***Functionality*** | ***Format*** |
| GET | Request provides user last positions of all units in specified user group. | JSON |

| ***Parameters*** | ***Format*** | ***Role*** |
|---|---|---|
| user | Text | Identifier of user group |

*Table 7 GetLastPosition service*

### GetLastPositionWithStatus

| **URL:** /DataService?Operation=GetLastPositionWithStatus&user=<username> | | |
|---|---|---|
| ***Method*** | ***Functionality*** | ***Format*** |
| GET | Request provides user information about alert events and other attributes in addition to previous GetLastPosition request. | JSON |

| ***Parameters*** | ***Format*** | ***Role*** |
|---|---|---|
| user | text | Identifier of user group |

*Table 8 GetLastPositionWithStatus service*

### GetTracks

| **URL:** /DataService?Operation=GetTracks&user=<username>&limit=<limit> | | |
|---|---|---|
| ***Method*** | ***Functionality*** | ***Format*** |
| GET | Request returns entered number of trajectory geometries of all moving units in entered group. | JSON |

| ***Parameters*** | ***Format*** | ***Role*** |
|---|---|---|
| user | text | Identifier of user group |
| limit | Numeric value | Number of positions to receive |

*Table 9 GetTracks service*

### GetRecentTracks

| **URL:** /DataService?Operation=GetRecentTracks&user=<username> | | |
|---|---|---|
| ***Method*** | ***Functionality*** | ***Format*** |
| GET | Request returns trajectory geometries of all moving units in entered group. | JSON |

| *Parameters* | *Format* | *Role* |
|---|---|---|
| user | text | Identifier of user group |

*Table 10 GetRecentTracks service*

#### 3.3.2.3 Group services

GroupService provides detailed information about user groups. User groups can be arranged in hierarchy.

| URL: /GroupService?Operation=<operationName> | | |
|---|---|---|
| ***Method*** | ***Functionality*** | ***Format*** |
| GET | Provides detailed information about user groups. | JSON |

*Table 11 Group services*

### GetGroups

| URL: /GroupService?Operation=GetGroups&user=<username> | | |
|---|---|---|
| ***Method*** | ***Functionality*** | ***Format*** |
| GET | Request returns information about entered group. | JSON |

| *Parameters* | *Format* | *Role* |
|---|---|---|
| user | text | Identifier of user group |

*Table 12 GetGroups service*

### GetSuperGroups

| URL: /GroupService?Operation=GetSuperGroups&user=<username> | | |
|---|---|---|
| ***Method*** | ***Functionality*** | ***Format*** |
| GET | Request returns information about superior group to entered group name. | JSON |

| *Parameters* | *Format* | *Role* |
|---|---|---|
| user | text | Identifier of user group |

*Table 13 GetSuperGroups service*

### GetSubGroups

| URL: /GroupService?Operation=GetSuperGroups&group_id=<groupId> | | |
|---|---|---|
| ***Method*** | ***Functionality*** | ***Format*** |
| GET | Request returns information about subordinate groups to entered group. | JSON |

| *Parameters* | *Format* | *Role* |
|---|---|---|
| group_id | Numerical value | Identifier of group |

*Table 14 GetSubGroups service*

#### 3.3.2.4 Sensor service

SensorService provides information about sensors and enable to get measured or processed data.

| URL: /SensorService?Operation=<operationName> | | |
|---|---|---|
| ***Method*** | ***Functionality*** | ***Format*** |
| GET | Provides detailed information about sensors and provides methods to get sensor data. | JSON |

*Table 15 Sensor service*

##### GetSensors

| URL: /SensorService?Operation=GetSensors&unit_id=<unitId> | | |
|---|---|---|
| ***Method*** | ***Functionality*** | ***Format*** |
| GET | Request returns list of sensors connected to entered unit. | JSON |

| ***Parameters*** | ***Format*** | ***Role*** |
|---|---|---|
| unit_id | Numerical value | Identifier of unit |

*Table 16 GetSensors service*

##### *GetObservations*

| URL: /SensorService?Operation=GetObservations&&unit_id=<unitId>&sensor_id=<sensorId>&from=<fromTime>&to=<toTime>&trunc=<trunc> | | |
|---|---|---|
| ***Method*** | ***Functionality*** | ***Format*** |
| GET | Request provides access to measured or processed observations for entered unit-sensor pair and entered time range. If user doesn't enter time range, servlet returns all available observations for entered unit-sensor pair. Another optional parameter is trunc that executes average of values for entered epoch (hour, day, week...). | JSON |

| ***Parameters*** | ***Format*** | ***Role*** |
|---|---|---|
| unit_id | Numerical value | Identifier of unit (mandatory) |
| sensor_id | Numerical value | Identifier of sensor (mandatory) |
| from | Timestamp (ISO 8601) | Time stamp of beginning time range (optional) |
| to | Timestamp (ISO 8601) | Time stamp of end time range (optional) |
| trunc | Text | Average epoch (optional) |

*Table 17 GetObservations service*

#### 3.3.2.5 Alert service

AlertService provides information about alerts events that arrived in sensor network. Methods allow user to get description of potential alerts connected to specific unit and list of arrived alert events including

solving state.

| URL: /AlertService?Operation=<operationName> | | |
|---|---|---|
| ***Method*** | ***Functionality*** | ***Format*** |
| GET | Provides information about alerts events that arrived in sensor network. | JSON |

*Table 18 Alert service*

### GetAlerts

| URL: /AlertService?Operation=GetAlerts&unit_id=<unitId> | | |
|---|---|---|
| ***Method*** | ***Functionality*** | ***Format*** |
| GET | Request provides list of potential alerts for specified unit. | JSON |

| ***Parameters*** | ***Format*** | ***Role*** |
|---|---|---|
| unit_id | Numerical value | Identifier of unit |

*Table 19 GetAlerts service*

### GetAlertEventsByTime

| URL: /AlertService?Operation= GetAlertEventsByTime&unit_id=<unitId>&from=<fromTime>&to=<toTime> | | |
|---|---|---|
| ***Method*** | ***Functionality*** | ***Format*** |
| GET | Request provides list of arrived alert events for specified unit and specified time range. | JSON |

| ***Parameters*** | ***Format*** | ***Role*** |
|---|---|---|
| unit_id | Numerical value | Identifier of unit |
| from | Timestamp (ISO 8601) | Time stamp of beginning time range (optional) |
| to | Timestamp (ISO 8601) | Time stamp of end time range (optional) |

*Table 20 GetAlertEventsByTime service*

## 3.3.3 Web GUI

SensLog contains simple GUI for visualization of measured data on the Web. Web GUI contains visualization of unit positions and visualization of current values in sensor network and history of values for selected time span for specified sensor. Visualization of position of unit is shown on figure 8 below, current values of specified unit is shown on figure 9 and 7-day history of air pressure sensor is shown on figure 10.

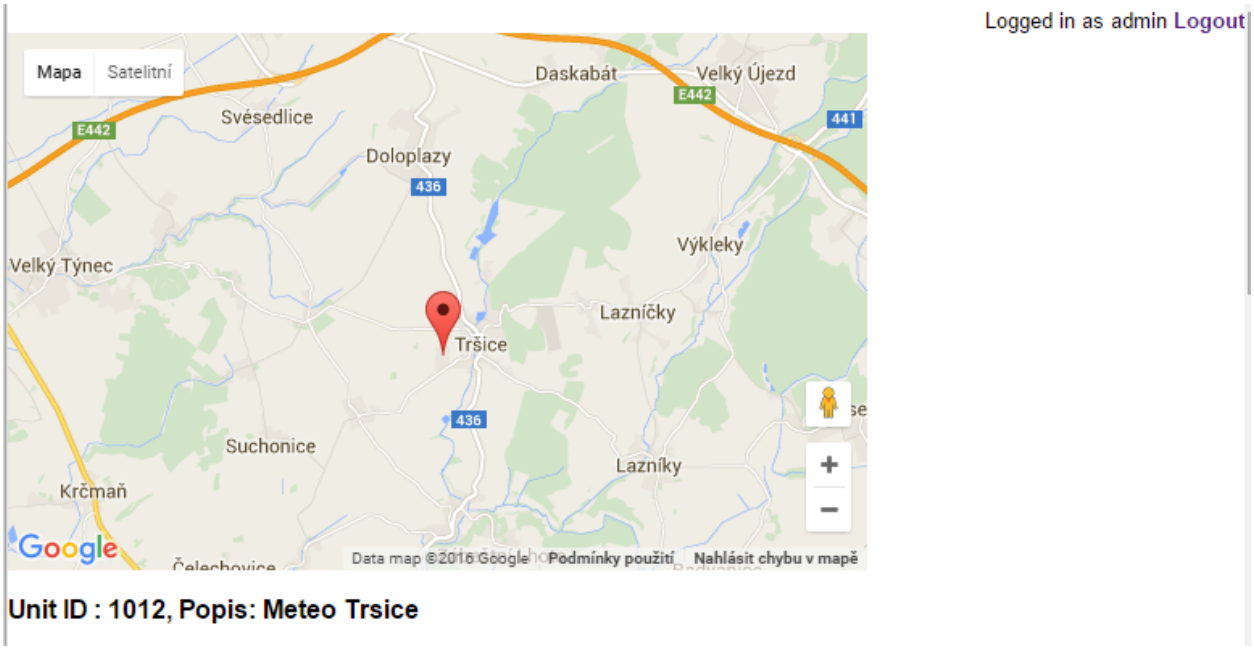


*Figure 12 Example of unit position visualization*


*Figure 13 Example of current observed values of selected unit*

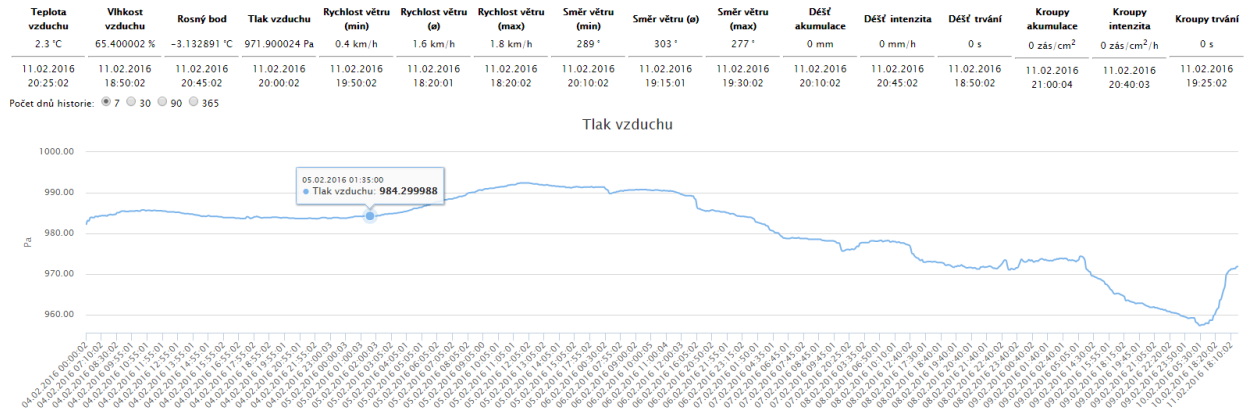


*Figure 14 Example of 7-day history of air pressure sensor on selected unit*

# 4 Conclusion

The described solution was laboratory tested and is now deployed on Czech pilot. It is ready also for deployment on other pilots. Now is also running integration with Web GIS