Fourth International Conference
Modelling and Development of Intelligent Systems
October 28 - November 1, 2015
“Lucian Blaga” University
Sibiu - Romania

Developing Internet of Thinks- based environment smart sensing network using model view controller

Mircea Risteiu, Ioan Ileana, Constantin Hutanu, Gheorghe Marc

Abstract

The paper is a part of environment smart sensing, framed into Internet of Thinks. Based on the fact that the sensor network is dynamically changing its configuration, the storing, preprocessing and processing of the data are changeable in time by user, and the viewing and reporting also need to be changed quite often, an independent approach need to be implemented for each three part of the system. Model- View- Controller (MVC) is the technique we are experiencing in our application. In this three layer architecture, after the initialization of the MVC, we are testing real time meaning in sensor network, we updates different processing applications on the server side, for querying optimization, and we are measuring the user interaction with view which calls a specified controller action. We also measure and test how the controller updates the models and how the view is refreshed for two different kind of smart sensors. By combining XML- based technologies with JavaScript Object Notation (json), we have imagined a scenario with multiple file input control with one type list stored in the XML file, instead of creating multiple properties, one for each control. By using one model builder that updates the property with the values by json serialization, there is a way of independent update of the processing functions on the server side.

1 Introduction

The research summarized in this paper is dedicated to the management of a municipal landfill monitoring. The monitoring system consists in a reliable network of smart sensors that takes measurements from the underground lands, and surroundings air, sends data over the network to a processing system database. The measured parameters for soil are: temperature and humidity, pH, Nitrate, Calcium, Iodide, Conductivity (Salinity), Dissolved Oxygen, and Chloride. For the air we measure: CO, CO2, NO2, O3, CH4, H2S, NH3, C4H10, and H2. For the measurement implementation, we are using electronic active devices, which are calibrated in labs and in site, based on the national specific regulations, and local Environment Agreement technically elaborated for a specific purpose. The in site calibration is annually re- calibrated. The data is collected into a web database.
At this moment they are some processing programs running on the server- side, which are open to changes, but they are some programs which are secured, and any changes might be done only according to the general (security requirements) and specific regulations (sensors’ data interpretation). For the general management purpose, we designed a control interface of the sensors (sensor status, lifetime, sampling time changing). For the data management we have designed real time, history and alert representations. All this application facilities are able to be used independently, or in Google map’s services.

The major related conclusion is that all the hardware software components are continuously and independently changed.

1.1 The specific of the database systems for smart sensors- based measuring systems

Since years of '90, when the researchers defined as "federated database systems" the databases where the data has been manipulated as different formats, protocols, and query languages, where they were identified different ways of representing a and storing the data, where the table decompositions may vary, column names may be different, data encoding schemes may also vary, with, or without similar semantics.
Referring to the smart sensors, data across constituent databases may be different but related, that way there is necessary to design semantic heterogeneity. In this case, the data structures are different according to the sampling rules in the process of measurements in one side, and database design rules on the other side.

1.2 Municipal waste landfill underground and air contamination

Because of landfill location (very close with populated areas, important rivers) the paper proposes a networked smart measurement pollution. The major related specificities are: large landfill surroundings- the area covered by the monitoring system is much larger then landfill area, very depending by the landforms and integrated valleys and rivers; strong and close link between landfill and air and water.
The landfill gas composition is methane, carbon dioxide, nitrogen, oxygen, and hydrogen, pungent odors (hydrogen sulfide, H2S), volatile organic compounds, ground–level ozone (smog), benzene, toluene, ethyl benzene, and vinyl chloride.

1.3 Specifics of the real time environment sensors measurements

Communication is based on 1-wire protocol ([1]), ([2]) and it is used for communication between measuring device and active sensors. The protocol is very simple, and for parameters measurements there is a demand to check if the data transmission is right. Detection of the pollution gases is achieved by measuring the sensing resistor RS during operation, according to the sensor's datasheet ([3]). The sensor is based on resistance variance in the presence of small concentrations of measuring gases. The sensor's resistance in air, as well as its sensitivity, can vary between different units, so it is recommended to calibrate each one of them before finally inserting them in the application. The self- compensation consists in: gain error, non-linear transfer curve, offset error, hysteresis error, and gain variation with temperature / time. For instance, the check-sum for temperature and humidity (according to the datasheets) should be:

- Check sum=8 bit integral RH data+8 bit decimal RH data+8 bit integral T data+8 bit decimal T data. Example: measuring device has received 40 bits data from RHT03 as

0000 0010 1000 1100 0000 0001 0101 1111 1110 1110

- 16 bits RH data 16 bits T data check sum. Check sum=0000 0010+1000 1100+0000 0001+0101 1111=1110 1110

RH= (0000 0010 1000 1100)/10=65.2%RH; T=(0000 0001 0101 1111)/10=35.1^{o}C
When highest bit of temperature is 1, it means the temperature is below 0 degree Celsius. Example: 1000 0000 0110 0101, T= - 10.1 ^{o}C
To get data from the first sensor module (temperature and humidity) we declare parameters to be read as (including calibration process according with datasheets):

```
Vfloat temp = myDHT22.getTemperatureC();
temp = (temp*9/5) + 32 - 1; //calibrate here
float humidity = myDHT22.getHumidity();oid main()
```

where, myDHT22 is the specific library. Then, if the values have to be used as strings, the conversion method is:

```
String temp_string = dtostrf(temp*100, 4, 0, s);
String humidity_string = dtostrf(humidity*100, 4, 0, s);
```

In order to figure out the independency of the devices which are running for measuring and submitting the data to the web database, next we shown the structure of the site implementation devices. Measuring is done through two different devices (measuring device) and gateway (measuring and transferring data from the measuring network to the Internet).

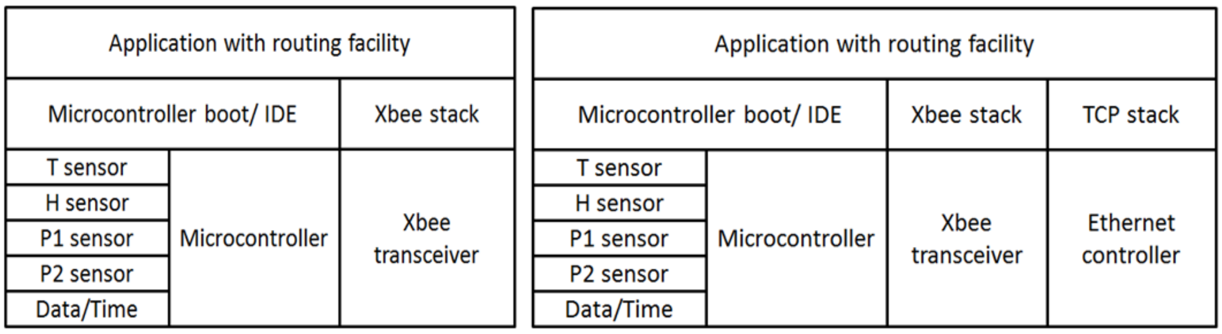

Figure 1. The software model for measuring, and gateway device

For both implementations, some logic level preparations must be done. Because, we need two serial ports, and because only one is hardware accessible, the other one have to created software. On the other hand, the measuring devices have to be proper configured.

1.4 Time scheduling model for software design

Time scheduling is related to measuring process demands not with software design. The local microcontroller has many additional tasks to do- it has to ensure that the data is sent to the database. The shown approach is also used for data storing locally (in the SD card). For that reason, it has to manage local data recording with sending data to the database. Based on these aspects, we have defined the Gantt diagram of the major tasks.

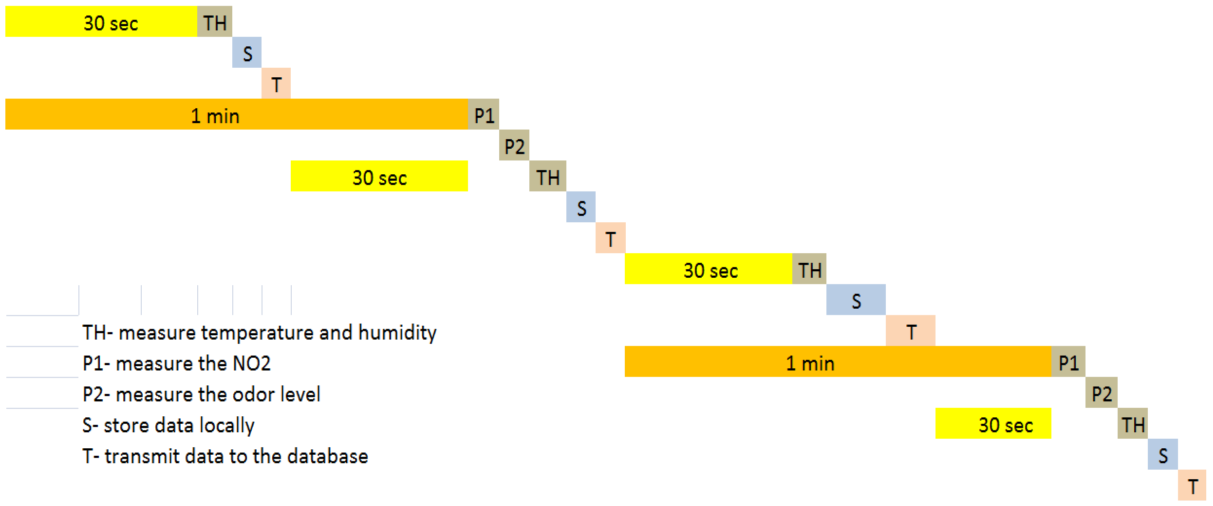

Figure 2 Two sensor measurement cycle with measuring, local storing, and transmiting tasks

The storing process is arranged as follow: T,H,P1,P2,T,H,ENTER. The sending data to the database is shown in next pseudo code:

```
if cycle== 1 minute/2
        use insert1.php
if cycle == 1 minute
        use insert2.php
```

Each 30 seconds data is sent through a php file. At 30 seconds, insert1.php sends only T and H, in one table. At 1 minute, insert2.php sends T, H in one table, and P1, P2 in the second table.

1.5 Premises for using model view controller technique for smart sensor software design

They are few premises which made possible to develop smart sensing systems- as major purpose of developing smart sensor networks. One of them is the huge switch between computing systems and distributed computing in one side, and the way of implementing the concept of big data [4]- as a structured documents and applications, [5]- as a result. Another premise is supported by the progress of computing devices and technologies [6]. As far as any device became connected to the web with a very low cost solution, we understand how easy became also to integrate the transduction element part together on one silicon chip. Since 1996 researchers proposed to integrate into LSVI devices functions like sensing element, signal conditioning electronic and controller/processor that support some intelligence in a single package [7]. One more, when the portability over different types of processors became real fact, the architecture of such device was meet in all vendors and software designers major tasks.

2 Application architecture design

The application architecture, based on the required tasks is shown next:

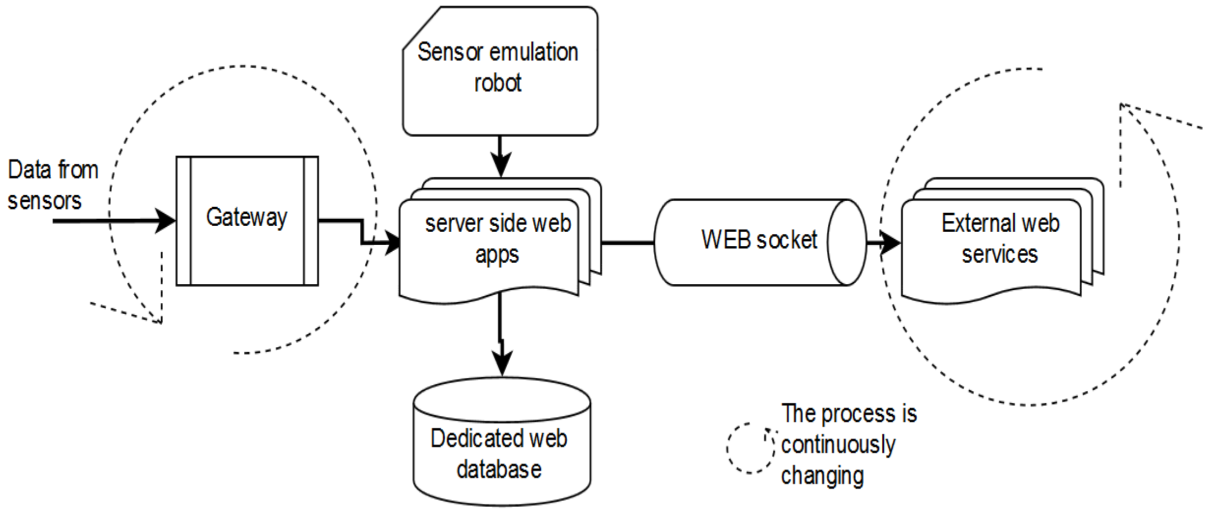

Figure 3 The designed software architecture, according to the required specifications. All block diagrams and models are designed with web resources ([8])

As it could be seen here, first designing constrains are related to the fact that, independently, the data which is coming from the site is not homogeneous, and the external services which are called (like Google map) are also working independently, and different versions means for our application continuous adaptations.
In two layer web application architecture, our application has to be continuously adapted.

2.1 Data coming application framework design

In order to design the application framework and its architecture, next we show the measuring sensor model.
As it could be seen in the figure 4, through the fusion process block, we accept different types of data, different sampling time, and in order to optimise the database design and exploitation, there

is required a strong data processing phase, before sending data through the gateway, to the database.

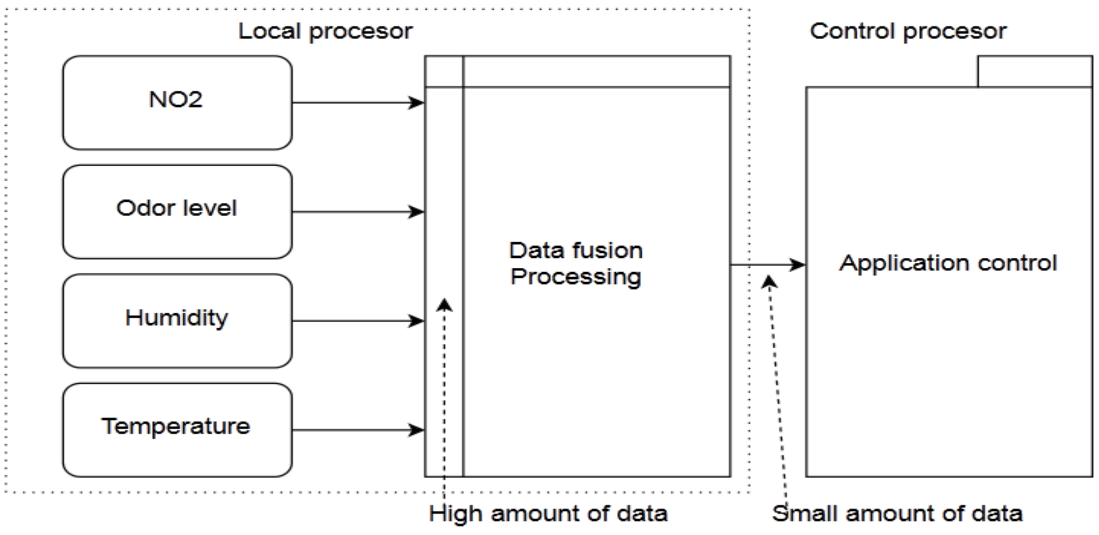

Figure 4 The designed architecture of data gathering device

2.2 Server side applications framework design

As far the application has to be continuously adapted when it interacts with both parts, one solution is to approach the design through model- view- controller (MVC) technique. All dedicated literature ([9], [10]) defines this technique like in figure 5:

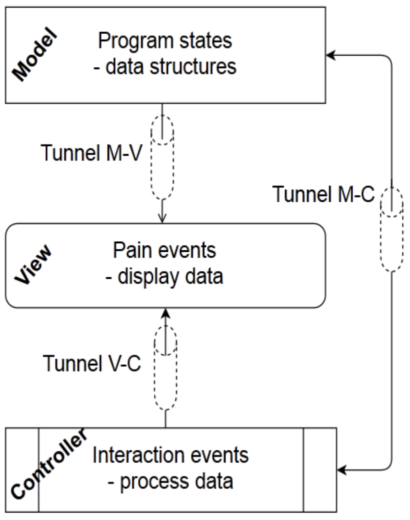

Figure 5 Adapted model-view-controller diagram

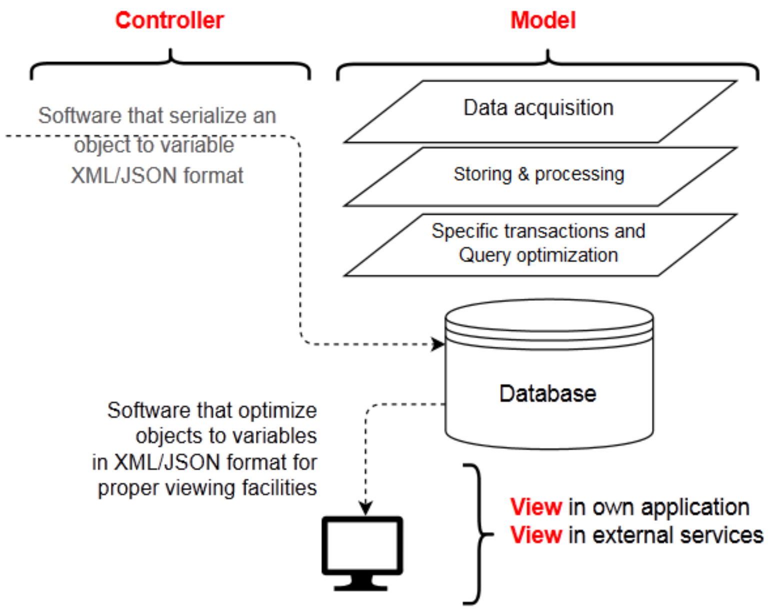
Controller
Model
Software that serialize an object to variable XML/JSON format
Data acquisition
Storing & processing
Specific transactions and Query optimization
Database
Software that optimize objects to variables in XML/JSON format for proper viewing facilities
View in own application
View in external services

Figure 6 Extended model with storing and processing functions

Because in many cases, the user (human of intelligent device) when produce data interact directly with the controller, we could say that in such structure any block architecture interact with all the others (figure 5). We insist in this pre-required designing condition because if we extend the MVC to the data gathering, the model becomes more complicated (figure 6). From the software point of

view, data acquisition function means heterogeneous data gathering, then the other two functions (processing and storing) are able to store properly the refreshed data to the database.
By running the application designed in this architecture, we are able to optimize some parts without taking care of how other application components react to this changes, simply by serializing data using common techniques like XML, and temporary formatting data in json templates until de controller adapt requirements according to the designed model.

2.3 View framework design

In order to understand to simple mechanism of the work we proposed an working algorithm of interaction between the four parts (including here also the user) which is shown in figure 7. Here, a simple interaction is explained. First, the model, view and controller are initialized at application start-up. This phase is not interfering with smart sensor start-up because after powering-up the sensor it will proceed with authentication, joining to the network, and first data sending sequence. The view displays sensors symbols to the user by reading data sent by the model. The user interacts with the view (e.g. presses a button) which calls a specified controller action. The controller updates the model in some way, than the view is refreshed (fetching the updated data from the model).

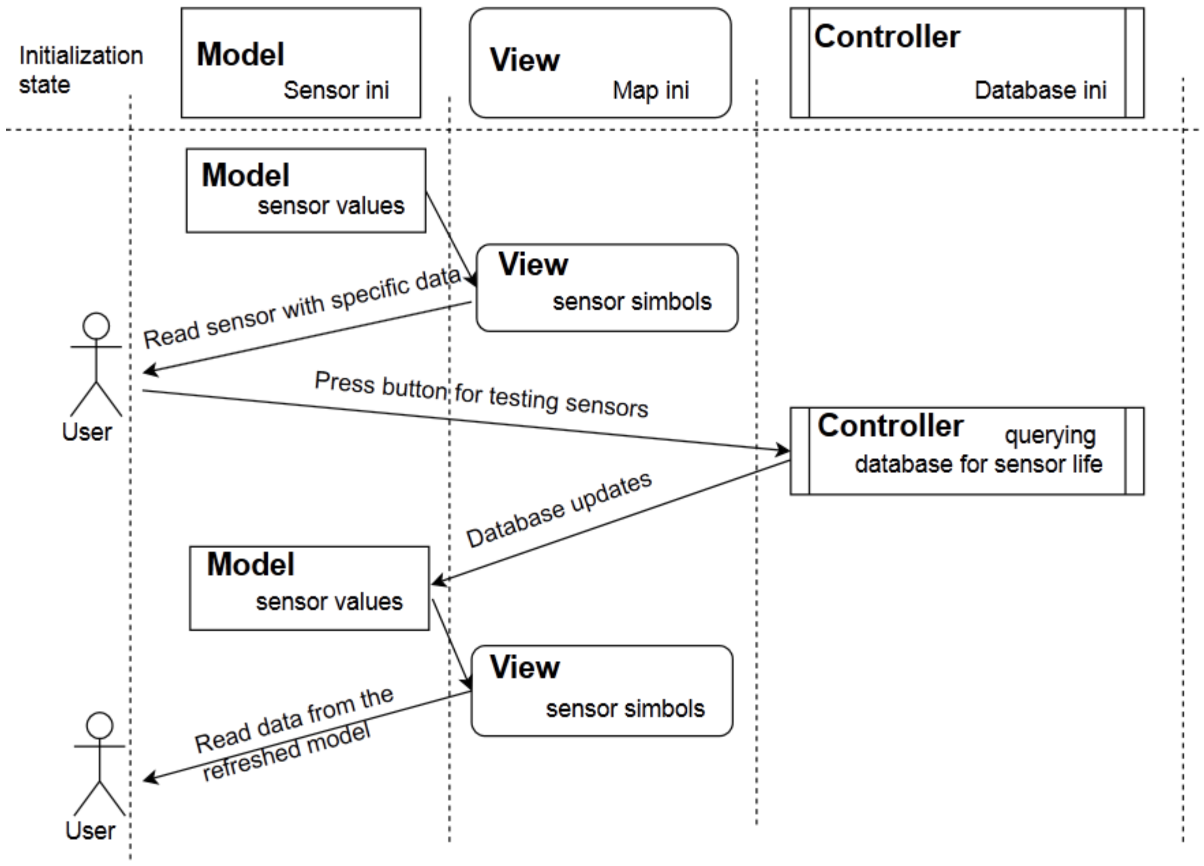

Figure 7 The specific designed algortihm for application management at client side

Based on this approach, the architecture of the designed application looks like in figure 8.
The used technologies for implementation are: PhP, MySQL, Ajax, and widgets from jQuery, with Linux servers. The major tests have been related to: user authentication, database connecting, new sensors value coming, own application accessing, external services use.

For an easy database management, we have proposed data storage into tables associated with parameters. Figure 9 shows how the data storage is implemented, in interaction with the controller (dynamic query that serialize data) and the model for viewing (php file).

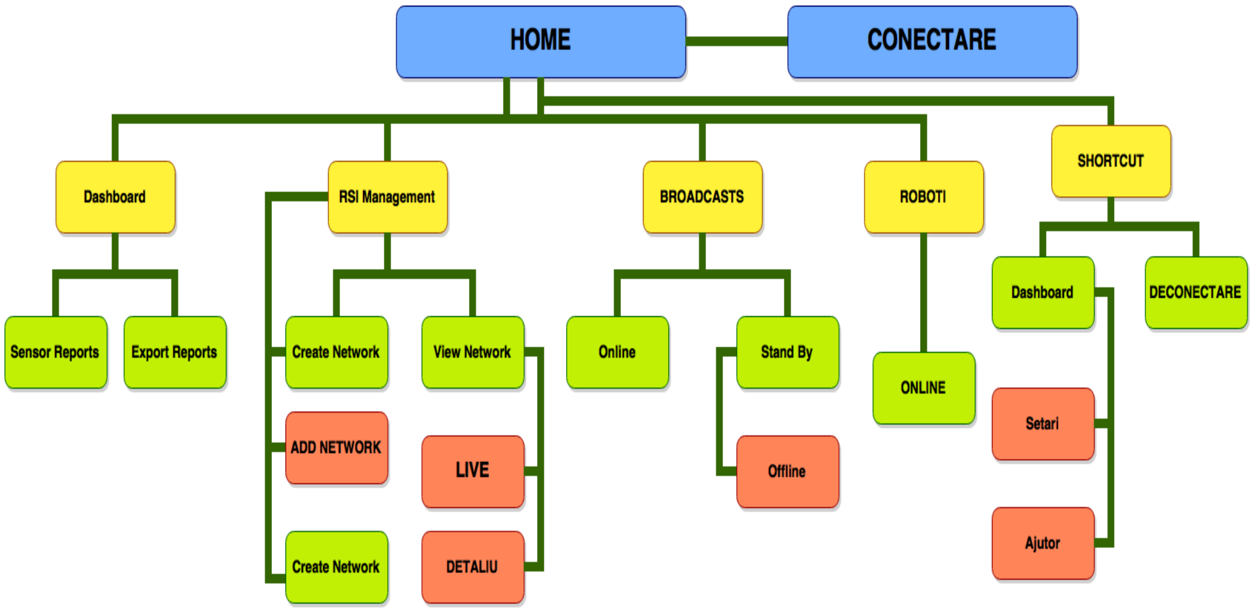

Figure 8. The application map of the designed architecture

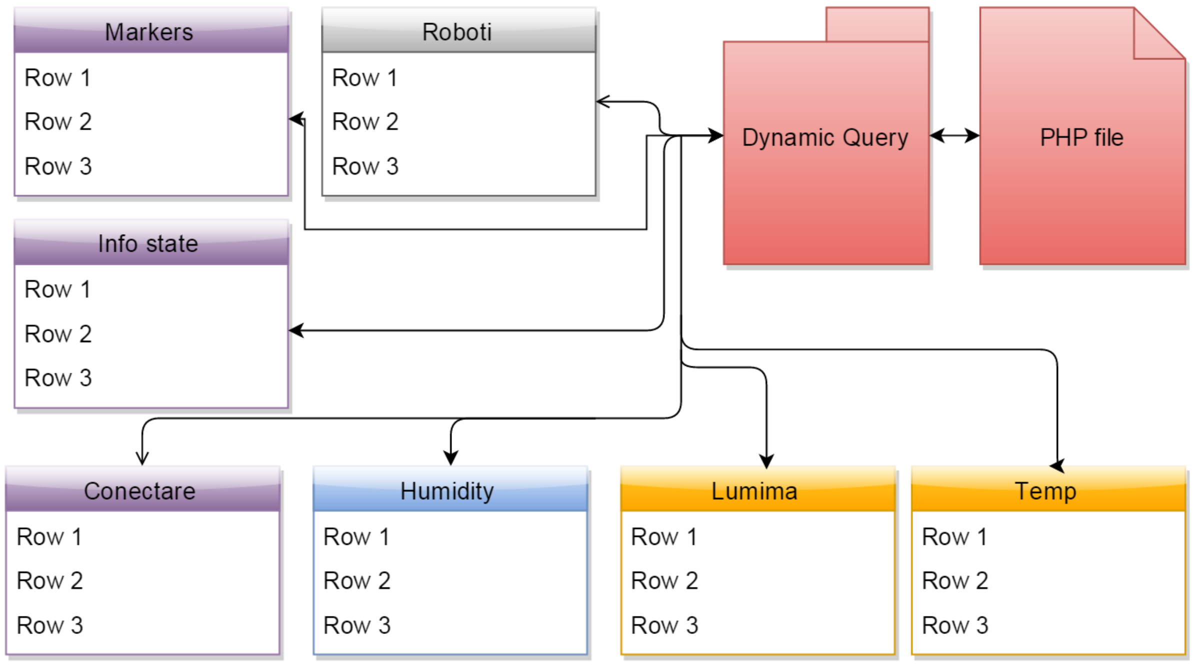

Figure 9 Sensors data in stored in dedicated table for each paramater, all values labeled with data and time

As it could be seen in figures 3, 7, and 9, by using own application models, the controller lets the different parts of the application to behave in own properties: the gateway will refresh the ports with own programmed parameters, then will push data to the database according to the server's privileges; the data is processed on the server when the files are invoked; the user, on the view side will wait until the controller will process the demanded actions, and the external services (Google map) will have own refresh time (5 seconds) and will process the dedicated scripts.

2.4 Designing socket model for external services

The structured communication between database and hetero data is made with XML file, which includes PHP code to collect simplify data ([12], [13]). The XML file will permit to be accessed by exterior application and fill up with data. In Google map service data is collected through markers. One example is shown next:

```
<markers>
<marker id="1" name="&amp;" address="not sepcific" temp="36.50" humy="&lt;u>Dezactivata&lt;/u>"
sun="&lt;u>Dezactivata&lt;/u>" lat="46.105194" lng="23.521086" status="activ" />
</markers>
```

The external service is invoked into:

```
<div id="map" class="map_arrange"></div>
```

Where an empty markup map from Google is loaded and it is waiting for markups:

```
google.maps.event.addDomListener(window, 'load', load);
```

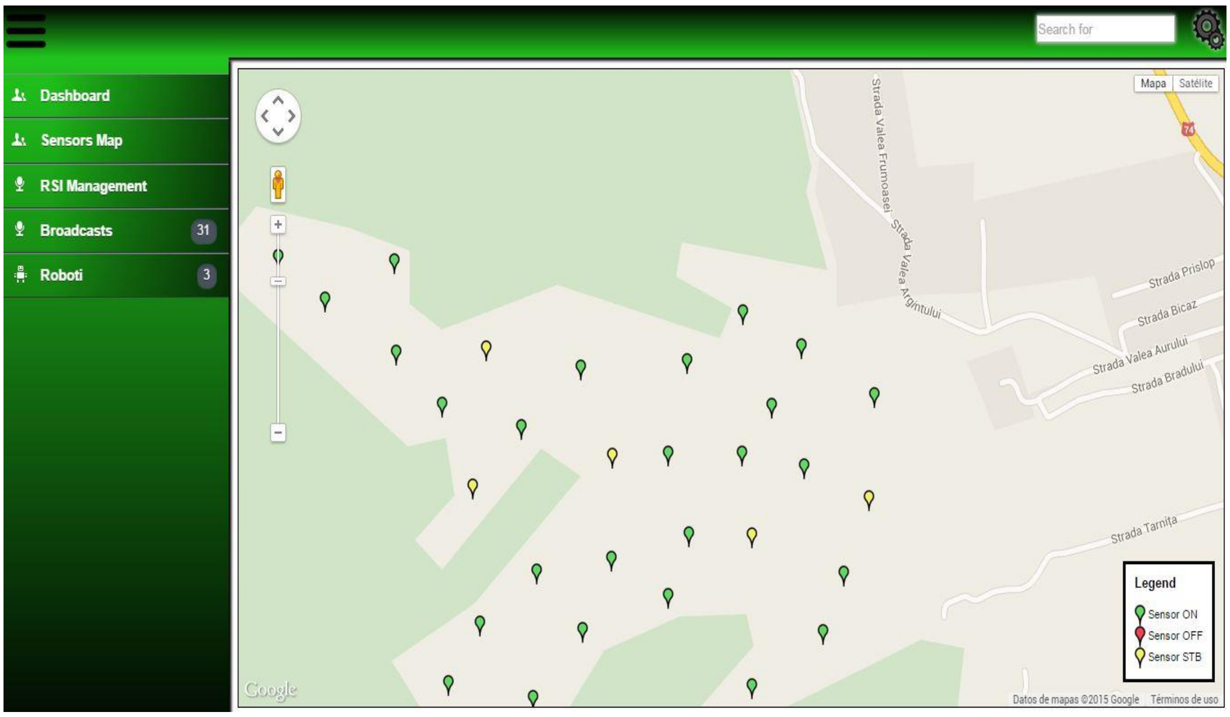

Figure 10 The front end of the application when external service is used to view the data

When this action is invoked, the PHP file will generate an XML from database with defined specified data. In order to get XML file, the XML process is made by the function *processXML(data)* with a single parameter.

```
function processXML(data) {
        var xml = data.responseXML;
        var markers = xml.documentElement.getElementsByTagName("marker");
        //clear markers before you start drawing new ones
        resetMarkers(markersArray);
        for (var i = 0; i < markers.length; i++) {
          var id = markers[i].getAttribute("id");
          var name = markers[i].getAttribute("name");
          var address = markers[i].getAttribute("address");
          var temp = markers[i].getAttribute("temp");
..........................................................................................
          var point = new google.maps.LatLng(
              parseFloat(markers[i].getAttribute("lat")), parseFloat(markers[i].getAttribute("lng")));
              //make a quick command button for temp.
          var menu = '<div align="left"><p></p><p>Temperatur&#259: '+temp+'&deg; C</p><p>Umiditate:
'+humy +'</p><p>Odor level &#259: '+sun +'</p><p>Via&#355&#259: 66%</p>'+
        '<div   onclick="senzoropt('+id+',\''+status+'\')"   id=""   class="button-style-info-iw"><p>Set&#259ri
Senzor '+ id+'</p></div></div>';
          var html = menu;
          var icon = customIcons[status] || {};
          var marker = new google.maps.Marker({
            map: map,
            position: point,
            icon: icon.icon
            });
            markersArray.push(marker);
            bindInfoWindow(marker, map, infoWindow, html);

        }
            setTimeout(function() {
                downloadUrl("inc/genfile_xml.php", processXML);
            }, 5000);
}
```

Every point from the map is a markup, which you can see it in xml file. The result looks like in figure 10.

3 Testing. Results

3.1 Working with XML and JSON formats in MVC

On the designed model let's analyse the JSON structure (data is in name/value pairs, data is separated by commas, curly braces hold objects, square brackets hold arrays) of the measuring sensor model:

```
{
  "name": "xxxxxxxx",  "id": "xxxx",  "isAlive": "true",  "batery": "value",
  "data":  { "year": "xxxx",  "month": "xx",  "day": "xx"  },
  "time":  { "hour": "xx",  "min": "xx", "sec": "xx",  "milisec": "xxx"  },
  "description":  {  "firstfield": "xxxxxxxx",  "secondfield": "xxxxxxxx",  "thirdfield": "xxxxxxxx"  },
  "values":
    [
    { "type": "humidity", "value": "xxxx.xx"  }, { "type": "temperature", "value": "xxxx.xx" },
    { "type": "NO2", "value": "xxxx.xx" }, { "type": "Odor level", "value": "xxxx.xx" }
    ],
}
```

This is quite similar structure with XML. As we know, the definitions in the XML Information Set specification are meant to be used in other specifications that need to refer to the information in a well-formed XML document. So they are some significant differences: the complexity of the structured controllers, the portability of the files, the possibility to store data according to the specific measuring sampling rules, and the position independent. Next they are two other JSON samples:

```
"time":  { "hour": "10",  "min": "22", "sec": "16",  "milisec": "315"  },
"values":  [ { "type": "humidity", "value": "44.00"  }, { "type": "temperature", "value": "34.15" },
    { "type": "presure", "value": "800.00" },  { "type": "so2", "value": "5.00" }    ],
....
"time":  { "hour": "10",  "min": "22", "sec": "33",  "milisec": "315"  },
"values":   [ { "type": "humidity", "value": "44.00"  }, { "type": "so2", "value": "5.00" }    ],
```

As a data oriented technique, the JSON is used closed with controller for heterogeneous data. The array of values is parsed when received, then, if they are available values according to the types of measuring parameters, the parser will store data into different tables, without inserting value "0" at each sampling period. The sample code shows how to parse JSON data in our application:

```
<p id="341"></p>
<script>
...
obj = JSON.parse(text);
document.getElementById("341").innerHTML =
obj.values[i].humidity + " " + obj.values[i].temperature + " " + obj.values[i].presure + " " +
obj.values[i].so2;
</script>
```

For data capturing , a simple PHP code is written ([13], [14]).

3.2 Smart sensors evaluation in the MVC technique approach

The main evaluation parameter for sensor proper working is the time delay between data refreshing. At this level of processing we are not taking into account lost, or uncompleted received packets because we are using XBee device in API mode ([11] [12]), where only packets that fulfils data integrity are considered.
In these conditions, the behaviours of the 10 XBee devices are explained in next two figures. This is the typical test we are running (10). First figure (11) shows that after 4-5 frames, each XBee device "learn" the way of data sending- after 5 frames the time delay is uniform increasing (from one XBee device to another) up to 5 seconds.

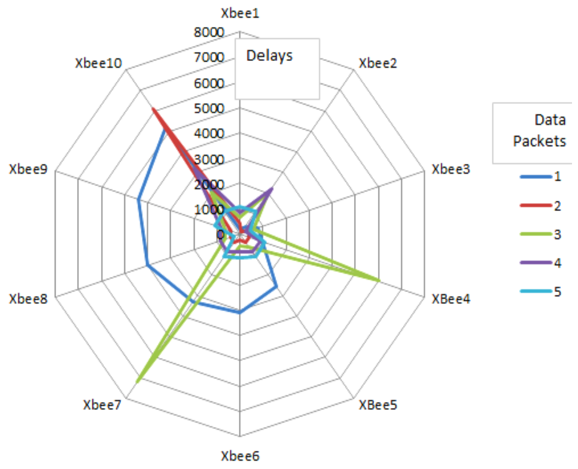

Figure 11 Delay evaluation at start-up of the sensor network

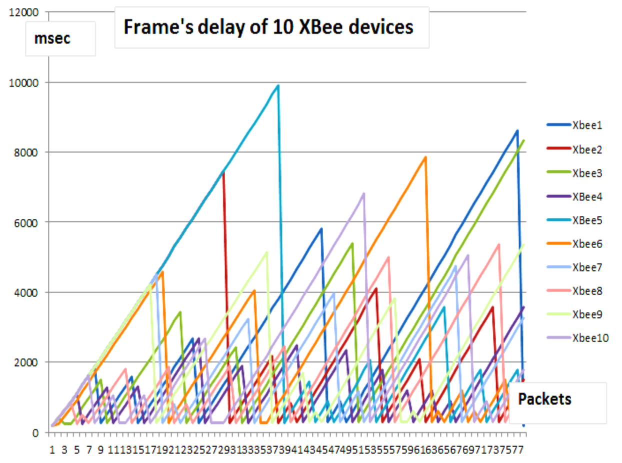

Figure 12 Measured delay during the netwrk testing

The second figure (12) shows records for long time delay in data packets reception process.
After the first 5 packets, the data is received from all 10 XBee devices (there not a gap into a device data transmission). The highest delay is below 10 seconds. The XBee number 5 shows the highest delay (not the farthest device), that means that in the established mesh network, there is a shorter way of data sending (possible because of radio propagation or perturbation conditions).
As it could be seen, there is interaction between application components.

3.3 Viewers evaluation as parts of MVC

The application is designed to work and switch automatically between own application and external services. This is possible because the two implementations are treated as different clients of the same application with the same management facilities. Ones the hardware implementation

is finished, the network viewing can be established in two ways- in own (local) application and Google Service.
In local application, the sensors are listed for collective and individual configuring and accessing. Independently by the option, each sensor is listed with own status (Online, Stand By or Offline).
Then each sensor (figure 13) can be accessed for configuration (by checking, we activate some measuring parameters- in sensor 20 we activate temperature and humidity sensing) for real time data displaying, or for history of values.

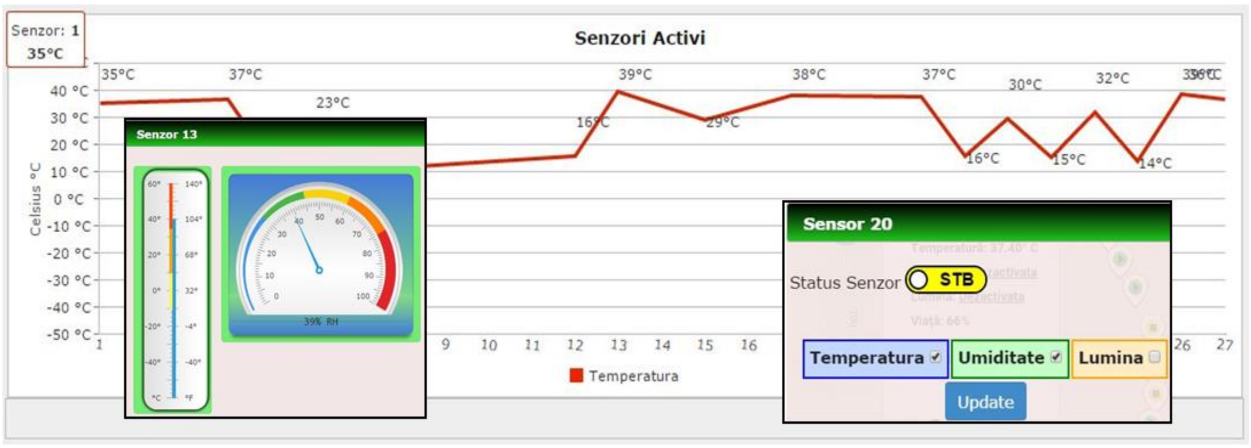

Figure 13 Captured screen for viewing facilities for sensor configuration (sensor 20), realtime viewing (sensor 11) and history values for sensor 1

For accessing data in Google map, we load the object as a child through main menu (figure 10). The screen capture shows the application frame that loaded the Google map with the geographically visualization of the installed sensors.
The application loads over a new layer with the specific legend (the most significant status information- ON, OFF, Standby). On the other hand, we have supplied with information the fast viewing facility of the Google map. By accessing the map "thumbtack" that represents a sensor, the provided information is shown 10 seconds.

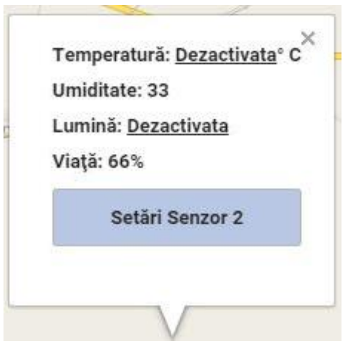
Temperatură: Dezactivata° C
Umiditate: 33
Lumină: Dezactivata
Viață: 66%

Figure 14 Captured screen with the most significant sensor information

The figure 14 tells (as a public profile) to the user that the temperature and light sensors are not active, the humidity value is 33%, and the "life" of the sensor is 66% (the accumulator's capacity is 66% from the nominal energy value). If we react on the "Setari Senzor 2", the frame application switches to the own application for listing, configuring the sensors from the network, as presented in first part of this section (see figure 13).
The sensor's values history is also available. Depending by the user options, graph, or histogram option can be used.

4 Conclusions

During application tests we were always focused on fails probabilities when changes are done in different parts of the hardware/software architecture.
By using the dedicated communication protocol, the maximum obtained delay in 5 minutes for two successive measurements in the area with an electricity high voltage network. By self-testing facilities of the sensors, any fail in measurements is forecasted, then easy detected through the sensors' specific protocol, and stored in the database.
The second major research task have been devoted to test for using a communication reliable system, by using an ultra-low power transceiver controlled by a dedicated application. The optimized application ensures information link in any condition, with an acceptable time delay. The recorded results prove an acceptable delay limit (less than seconds) which offer the possibility of adding as many transmission devices are necessary. The Zigbee protocol accepts 264 64-bits device addresses, but limitations are related to network forming and specific transmission delay.
During tests, some measuring adjustments have been required because we intended to reduce the communication time, as a major problem in networked smart sensors. According to Niquist data acquisition condition, the minimal sampling frequency limit is 3 x 150 Hz= 450Hz- or one sample at each 2.2 milliseconds. If we want to take 10 samples for each oscilation dirrection, it means that the sampling period is 0.22 milliseconds (4.545 kHz sampling frequency). In upper estimations, in any case (8/ 16 bytes) there is a cover limit in communication transfer (8333/4545 in reading phase, and 10526/4545 in writing process). There is a time window of 0.22 milliseconds which can be used for another process task First, there is a need to speed up the data transfer in this bus. Usually, it is done by improving the standard transfer frequency (from 100 kHz to 400 kHz) and bus length (from 32 bytes to 64 bytes). Most of the 8 bits microcontrollers allow these changes. During changing tests, no conflicts or unacceptable time delay were observed in working with the database.
In the last conclusions idea, the present paper has been dedicated to design an upper level monitoring and decision making system, which has the main role to respond properly to changes demand, to ensure a satisfactory level of data management coming from de network, and over the web-based application. The implemented hardware software solution for the municipal landfills management covers a large land surface, through reliable networked sensors systems.

References

[1] xxx 1-Wire® Communication with PIC® Microcontroller datasheet, (2015)
[2] xxx AVR318: Dallas 1-Wire® master datasheet, (2015)
[3] xxx (MiCS – 2710) NO2 Sensor datasheet, (2015)
[4] Chun-Wei Tsai, Chin-Feng Lai, Han-Chieh Chao, Athanasios V. Vasilakos, Big data analytics: a survey, http://www.journalofbigdata.com/content, visited in June 2015,
[5] Patrick Schwerdtfeger, Webify Your Business, Internet Marketing Secrets for the Self-Employed, ISBN 978-0-557-04901-1, Wiley, 2009
[6] Erlendur Kristjansson, Portability of the 32 bits microcontrollers' platforms as IoT support, Microchip Technology report, www.microchip.com, visited in March 2015.
[7] J. M. Giachino, “Smart sensors,” Sensors and actuators, 10(1986) 239-248.
[8] xxx www.draw.io, (2015).
[9] Trygve Reenskaug, James Coplien, The DCI Architecture: A New Vision of Object-Oriented Programming, http://www.artima.com/articles/dci_vision.html, visited in June 2015
[10] Atanas (Nasko) Rountev, Model-View-Controller (MVC). Design Pattern, http://web.cse.ohio-state.edu, visited in June 2015
[11] http://www.libelium.com/, visited in March 2015

[12] Dr. Ovidiu Vermesan, Dr. Peter Friess, EU, Belgium Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems, ISBN: 978-87-92982-96-4 (E-Book), © 2013 River Publishers, PO box 1657

[13] https://www.google.com/webmasters/tools/home?hl=ro, last visited in March 2015

[14] Mircea Risteiu, Ioan Ileana, A redundant solution for collective residences inflammable gas leaking monitoring using ultra-low power transmission system and automatic control, AQTR 2014 978-1-4799-3732-5 ©2014 IEEE

Mircea RISTEIU
"1 Decembrie 1918" University of Alba Iulia
Computer Science and Engineering Department
Gabriel Bethlen Str., No.5, 510009 Alba Iulia
ROMANIA
E-mail: mristeiu@uab.ro

Ioan ILEANA
"1 Decembrie 1918" University of Alba Iulia
Computer Science and Engineering Department
Gabriel Bethlen Str., No.5, 510009 Alba Iulia
ROMANIA
E-mail: iileana@uab.ro

Constantin HUTANU
"1 Decembrie 1918" University of Alba Iulia
Computer Science and Engineering Department
Gabriel Bethlen Str., No.5, 510009 Alba Iulia
ROMANIA
E-mail: chutanu@uab.ro

Gheorghe MARC
"1 Decembrie 1918" University of Alba Iulia
Computer Science and Engineering Department
Gabriel Bethlen Str., No.5, 510009 Alba Iulia
ROMANIA
E-mail: ghemarc@yahoo.ro