Management, Operation and Measurements with GSN, a middleware for Sensor Networks

Juan José Roncero, Teresa Olivares, Luis Orozco-Barbosa

Albacete Research Institute of Informatics
University of Castilla-La Mancha
Albacete, Spain
0034 967599200

Jjroncero, teresa, lorozco@dsi.uclm.es

1. INTRODUCTION

GSN [1] is a software Middleware designed to allow deployment and programming of wireless sensor networks. Thanks to GSN's open API, you can quickly wite sensor networks applications by leaving the tedious task of managing low level dependent details to GSN. In addittion, you can explote the enormous heterogeneity of kind of networks.

GSN assumes the simple model shown in Figure 1: A sensor network internally use arbitrary multi-hop, ad-hoc routing algorithms to deliver sensor readings to one or more sink node(s). A sink node is a node which is connected to a more powerful base computer which in turn runs the GSN middleware and may participate in (large-scale) network of base computers, each running GSN and servicing one or more sensor networks [2].

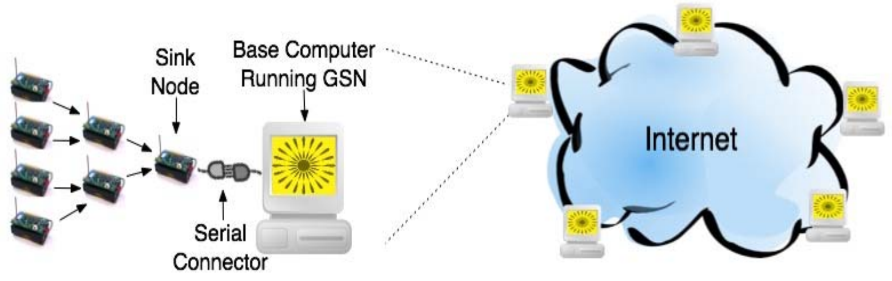


Figure 1: GSN model

2. MOTIVATION IN USE

We've chosen this Middleware because of some esential features. First, it is open source, so we could download and use it. This is not usual, due to this kind of frameworks are being developed in some universities, but they aren't available. Second, this is an active project and it's being maintaining at the present time. This is a very important thing, because we can consult problems and debts to original programmers or share ideas with the other maling-list users.

GSN is developed based on the observation that most of the logic requirements of the applications developed for sensor networks are similar.

Finally, GSN is designed to facilitate sensor networks applications development. Applications based in GSN are hardware-independent, making the sensor network changes invisible to the application.

3. VIRTUAL SENSORS

The key abstraction in GSN is the virtual sensor [2] . Virtual sensors abstract from implementation details of access to sensor data and correspond either to a data stream received directly from sensors or to a data stream derived from other virtual sensors. The specification of a virtual sensor provides all neccessary information required for deploying and using it, including:

- Metadata used for identification and discovery
- The structure of the data streams which the virtual sensor consumes and produces
- Functional properties related to persistency, error handling, life-cycle, management, and physical deployment.

To support rapid deployment, these properties of virtual sensors are provided in a declarative deployment descriptor.

Figure 2 shows an example which defines a virtual sensor tha reads id, humidity and temperature values coming from one sensor node.

A virtual sensor has a unique name (the name attribute in line 1). The example specification defines a virtual sensor with one input stream which is defined by its metadata (lines 22-23).

In GSN data streams are temporal sequences of timestamped tuples. This is in line with model used in most stream processing systems. The structure of the data stream a virtual sensor produces is encoded in XML as shown in lines 5-7.

Data stream processing is separated into two stages: (1) processing applied to the input stream (lines 25-26) and (2) processing for combining data from different input stream and producing the output strem (lines 28-30). In this case, only exists one input stream, so combine different input streams isn’t neccesary. To specify the processing of the input streams we use SQL queries which refer to the input stream by the reserved keyword WRAPPER. The attribute wrapper=”nuevoWrapper” indicates that the data stream is obtained from the wrapper “nuevoWrapper”. Previously we had implemented this wrapper to establish the communication with WSN and read messages from nodes.

In the given example the output stream manage parameters received from one StreamElement object generated inside the wrapper implementation and returns. To enable the SQL statemen in line 29 to produce the output stream, it needs to be able to reference the required input data streams which is accomplished by the alias atribute (line 19) that defines a symbolic name for each input stream. The definition of the structure of the output stream directly relates to the data stream processing that is performed by the virtual sensor and needs to be consistent with it, i. e., the data fields in the select clause (line 39) must match the definition of the output stream in lines 5-8.

On the other hand, the temporal processing is controlled by various attributes provided in the input and output stream specification, e.g., the attribute storage-size (line 19) defines the time window used for producing the input stream’s data elements. In addition to the specification of the data-related properties a virtual sensor also includes high-level specification of functional properties: The priority attribute (line1) controls the processing priority of a virtual sensor, the <life-cycle> element (line 12) enables the control and management of resources provided to a virtual sensor such as the maximum number of threads/queues available for processing, the <storage> element (line 16) allows the user to control how output stream data is persistently stored, and the disconnect-buffer-size attribute (line 20) specifies the amount of storage provided to deal with temporary disconnections.

Virtual sensors are a powerful abstraction mechanism which enables the user to declaratively specify sensors and combinations of arbitrary complexity. Virtual sensors can be defined and deployed to a running GSN instance at any time without having to stop the system.

```
1    <virtual-sensor name="HumedadTemperatura" priority="12">
2   <processing-class>
3      <class-name>gsn.vsensor.BridgeVirtualSensor</class-name>
4      <output-structure>
5            <field name="id" type="int"/>
6          <field name="humedad" type="int"/>
7          <field name="temperatura" type="int"/>
8      </output-structure>
9   </processing-class>
10   <description> Shows humidity and temperature
11                      from a sensor network</description>
12   <life-cycle pool-size="1"/>
13   <addressing>
14     <predicate key="geographical"></predicate>
15   </addressing>
16          <storage history-size="10h" permanent-storage="true"/>
17   <streams>
18     <stream name="datos" rate="100">
19        <source alias="wsn" sampling-rate="1" storage-size="8"
20                      disconnect-buffer-size="100">
21          <address wrapper="nuevoWrapper">
22             <predicate key="host">localhost</predicate>
23             <predicate key="port">9001</predicate>
24          </address>
25            <query> select nodeid as nd, humidity as hm,temperature
26          as tm from wrapper</query>
27        </source>
28        <query>
29     select nd as id, hm as humedad, tm as temperatura from wsn
30             </query>
31    </stream>
32   </streams>
33 </virtual-sensor>
```

Figure 2: Virtual Sensor description file

4. DATA STREAM PROCESSING AND TIME MODEL

The time model is a central building block in data stream processing. It defines the temporal semantics of data and determines the design and implementation of a system. GSN provides the essential building blocks for dealing with time, but leaves temporal semantics to applications allowing them to satisfy their specific requeriments.

In GSN a data stream is a set of timestamped tuples. The order of the data stream is derived from the ordering of the timestamps and GSN provides basic support for managing and manipulating the timestamps [2].

The temporal processing in GSN is defined as follows: The production of a new stream element of a virtual sensor is always triggered by the arrival of a data stream element form one of its input streams. The following processing steps are performed:

1) By default the new data stream element is timestamped using the local clock of the virtual sensor provided that the stream element had no timestamp.
2) Based on the timestamps for each input stream, the stream elements are selected according to the definition of the time window.
3) The input stream queries are evaluated and stored into temporary relations.
4) The output query for producing the output stream element is executed based on the temporary relations.
5) The result is permanently stored if required and all consumers of the virtual sensor are notified of the new stream element.

Additionally, GSN provides a number of possibilities to control the temporal processing of data stream, e.g.:

- The rate of a data stream can be bounded in order to avoid overloading the system which might cause undersirable delays.
- Data streams can be sampled to reduce the data rate.
- A windowing mechanism can be used to limit the amount of data that needs to be stored for query processing. Windows can be defined using absolute, landmark, or sliding intervals.
- The lifetime of data streams and queries can be bounded such that they only consume resources when actually active. Lifefimes can be specified in terms of explicit start and end times, start time and duration, or number of tuples.

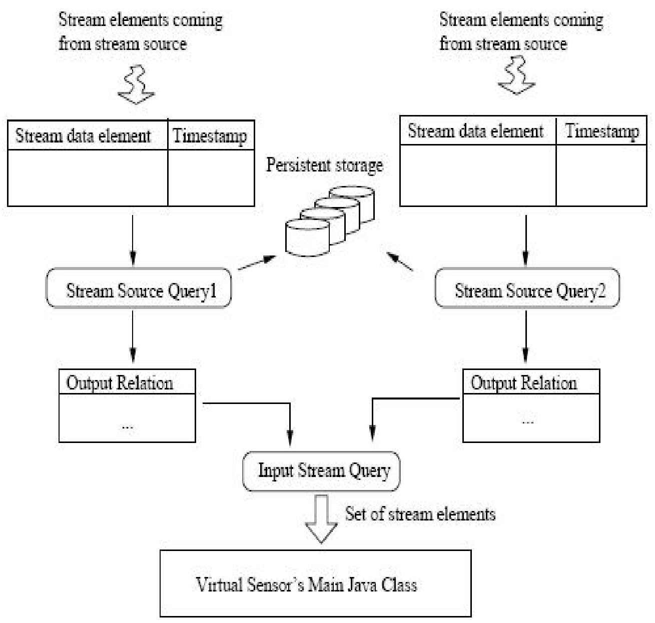


Figure 3: Data Time Model

5. SYSTEM ARCHITECTURE

GSN uses a container-based architecture for hosting virtual sensors. Every container hosts some virtual sensor which is responsible.

Using the declarative specifications, virtual sensors can be deployed and reconfigured in GSN containers at runtime.

Communication among different GSN containers is performed in a peer-to-peer style through stantard Internet and Web protocols. By viewing GSN containers as cooperating peers in a decentralized system, we avoid some of the scalability problems of many other systems which rely on centralized or hierarchical architecture.

Figure 4 shows the layered arquitecture of a GSN container. Each GSN container hosts a number if virtual sensors it is responsible for. The virtual sensor manager (VSM) is responsible for providing access to the virtual sensors, managing the delivery of sensor data, and providing the necessary administrative infrastructure. The VSM has two subcomponents: The life-cycle manager (LCM) provides and manages the resources provided to a virtual sensor and manages the interactions with a virtual sensor (sensor readings, etc.). The input stram manager (ISM) is responsible for managing the streams, allocating resources to them, and enabling resource sharing among them while its stream quality manager subcomponent (SQM) handles sensor disconnections, missing values, unexpected delays, etc., thus ensuring the QoS of streams. All data from/to the VSM passes through the storage layer which is in charge of providing and managing persistent storage for data streams. Query processing in turn relies on all of the above layers and is done by the query manager (QM) which includes the query processor being in charge of SQL parsing, query planning, and execution of queries. The query repository manages all registered queries (subscriptions) and defines and maintains the set of currently active queries for the query processor. The notification manager deals with the delivery of events and query results to registered, local or remote consumers. The notification manager has an extensible architecture which allows the user to largely customize uts functionality, for example, having results mailed or being notified via SMS.

The top three layers of the architecture deal with access to the GSN container. The interface layer provides access functions for other GSN containers and via the Web (through a browser or via web services). These functionalities are protected and shielded by the access control layer providing access only to entitled parties and the data integrity layer which provides data integrity and confidentiality through electronic signatures and encryption. Data access and data integrity can be defined at different levels, for example, for the whole GSN container or at a virtual sensor level.

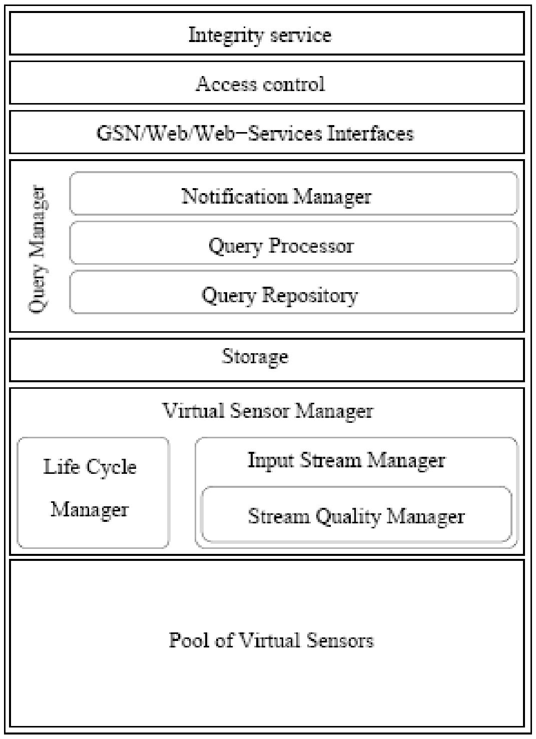


Figure 4: GSN Architecture

6. INTERCONNECTION

6.1 Sensor Network application

The Albacete Research Institute of Informatics [3] has developed several researches in this area. However, thus far no one of the applications had been implemented to interact with this kind of Middlewares.

Consequently, we based in a previous application developed in the institute to build a new one. The application called *IntellBuilding* was used. It vas implemented in NesC [4] to store humidity and temperature measurements, with the purpose of study the ambient conditions of the different areas in the institute. The rest of possible measures that this sensor board can carry out weren’t implemented to reduce de bateries consume (one of the main problems related to this kind of networks).

The network consited of:

- 1 standard ASUS desktop PC with Pentium® D, 3.4 GHz Intel® processor with 1 GB memory runnning Windows XP Professional. To make easy the project development, we decided install a virtual machine VMWARE which supported an XubunTOS [5] Operating System Image. This Operating System have integrated both TinyOS [6] versions (1.x and 2.x), so we shoudn't have to install them.
- A sensor network consisting of 3 MicaZ motes (see figure 5), each mote being equiped with light and temperature sensors.
- 1 serial wire to interconnect the sink node to the PC.
- 1 mts420CA sensor board
- 1 mib510 serial interface board.

Figure 5: MicaZ mote

6.2 Java representation of message structures

In order to interconnect GSN to our WSN we first needed to generate the java representation of the message structure used in the network (in TinyOS they are defined in the .h files). NesC language provides a tool called "mig" [7] (Message Interface Generator for NesC) for this purpose.

6.3 Wrapper Implementation

Once we generated the Java representation of the communication message structure, we needed to write a wrapper class for our message.

GSN can receive data from various data sources This is done by using so called wrappers. They are used to encapsulate the data received from data source into the standard GSN data model, called a StreamElements. A StreamElement is an object representing a row of a SQL table.

Consequently, we implemented a new wrapper which could satisfy our application requeriments. In this case, the StreamElement was build to encapsulate three parameteres: node id, humidity and temperature.

On the other hand, each wrapper is a Java class that extends the AbstractWrapper parent class. Usually a wrapper initializes a specialized third-party class in its constructor and interact with SerialForwarder tool [8] provided by TinyOS. The main use of this tool is connect the wrapper to the WSN.

The wrapper also provides a method which is called each time the library recives data from the monitored device. This method will extract the interesting data, optionally parse it, and create one or more StreamElement(s) with one or more columns. From this point on, the received data has been maped to a SQL data structure with fields that have a name (nodeid, humidity and temperature) and a type (int, int, int). GSN is then able to filter this using its enhanced SQL-like sintax.

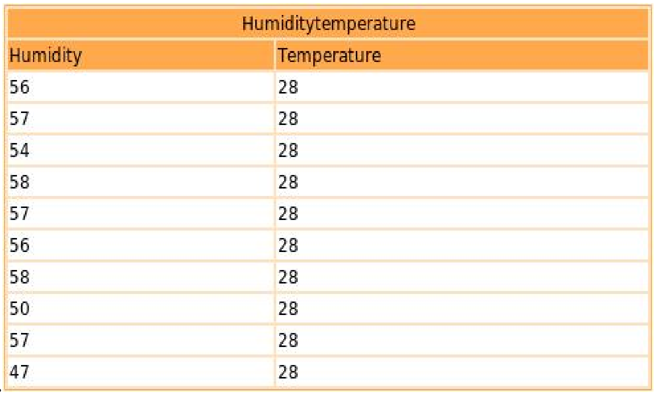
For simplicity, GSN uses short names to refer to the wrappers. We had to define this association in the file conf/wrappers.properties.

6.4 Virtual Sensor configuration files

In order to research the posibilities offered by GSN, we built some XML files to define the configuration of the Virtual Sensors which had to gather data received from wrapper and show them through GSN Web Interface.

6.4.1 BridgeVirtualSensor

First of all, we built a configuration file to check BridgeVirtualSensor funcionality. This Virtual Sensor inmediately publishes any data it receives without modifying it. The stream element object is not modified either. So we used it to filter and join sensor data through SQL queries without any other kind of post processing operation on the data (see figure 6).



Humiditytemperature	
Humidity	Temperature
56	28
57	28
54	28
58	28
57	28
56	28
58	28
50	28
57	28
47	28

.

Figure 6: data view

On the other hand, GSN web interface's data tab offers the possibility of show data based on different criterias and formats. We can plot data in bar or lines charts depending on the needs (see figures 7and 8).

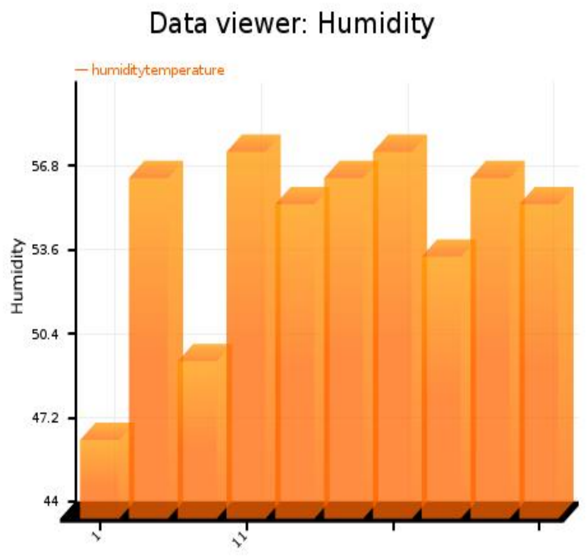


Figure 7: bar chart

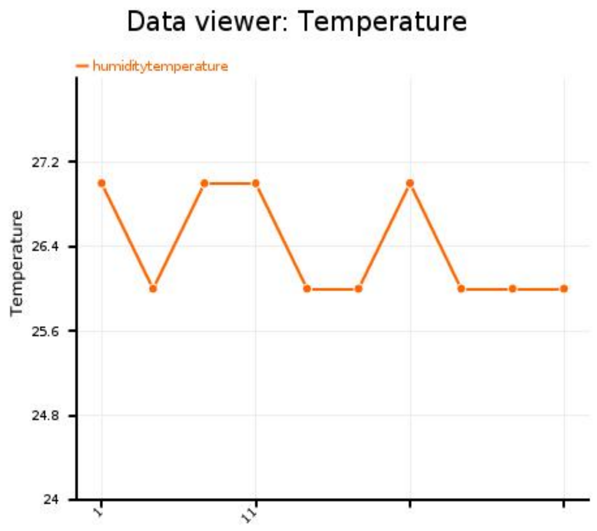


Figure 8: line chart

6.4.2 ChartVirtualSensor

The next step was checking how ChartVirtualSensor works. This virtual sensor generates graphs from the data it receives. Developers advised that this method could be computationally intensive, and we could se that they were right.

We checked Web Interface and we realized that this virtual sensor considerably slows down the system. It could be one of the main reasons for which developers improved the Interfaz Web settings as GSN couldn´t generate bar or line charts in its previous version. Differences between ChartVirtualSensor output (see figure 9) and new charts included into the Web Interface settings are evidents.

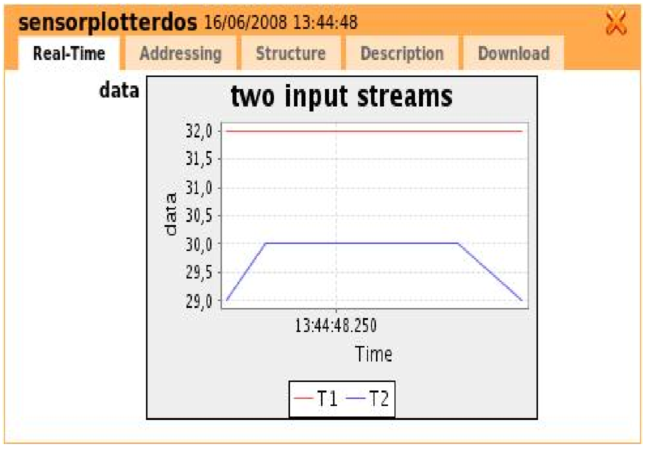


Figure 9: ChartVirtualSensor output

6.4.3 Remote Wrapper

The Remote Wrapper is a special wrapper that allows GSN to treat a remote virtual sensor (located on another machine in the network) as a data source for another virtual sensor. This data source can be configured on the same GSN server or it can be run on another computer.

We decided to use the data source in the same GSN instance to compare it with the local wrapper.

6.4.4 Local Wrapper

This is a special form of Remote Wrapper which always points to the GSN instance running on the same machine. Comparing both wrappers with the same Virtual Sensor, we could prove that remote wrapper suffers some overload from the network, so the local wrapper is quickly in this configuration.

6.4.5 HSQLDB Data Base

Thanks to StreamExporter virtual sensor, we could store data received from our network into the hsqldb [9] data base.

We had to modify the gsn.xml file situated inside */conf* directory in order to configure properly our data base. This file contains a set of configurations for every data bases it supports. In this case we selected:

```
<storage user=”sa” password=”” driver=”org.hsqldb.jdbcDriver”
url=”jdbc:hsqldb:file:MyFile” />
```

At the beginning of GSN’s execution, it automatically will generate a set of so called “MyFile” files with different extensions. Each file will store different elements:

MyFile.properties → The properties file contains general setting about the database.

MyFile.script → the script file contains the definition of tables and other database objects, plus the data for non-cached tables.

MyFile.log → The log file contains recent changes to the database.

MyFile.data → The data file contains the data for cached tables.

MyFile.backup → the backup file is a zipped backup of the last known consistent state of the data file.

MyFile.lck → This file is used to record the fact that the database is open. This is deleted at a normal SHUTDOWN. In some circumstances, a `test.data.old` is created and deleted afterwards.

To create the configuration file, we had to define a set of compulsory parameteres to establish our data base configuration. That parameters are precisely those that appear in the gsn.xml configuration file. Finally, we need to add the table name that will be built after inicialization.

6.4.6 Two input streams

We could do another test which consisted in check two input streams coming from two different nodes.

First of all, we had to built another configuration file which could gather the values coming from another sensor node. To differenciate between both sensor nodes, we asigned a differente id to the new node. Most of the new configuration file code was similar to the previous one.However, we had to add the sentence “WHERE nodeid=2” at the end of the SQL queries.

Finally we used both configuration files as sources from another virtual sensor which could join both input streams. Thanks to the Web Interface settings, we obtained the comparison between both input streams (see figures 10, 11, 12 and 13).

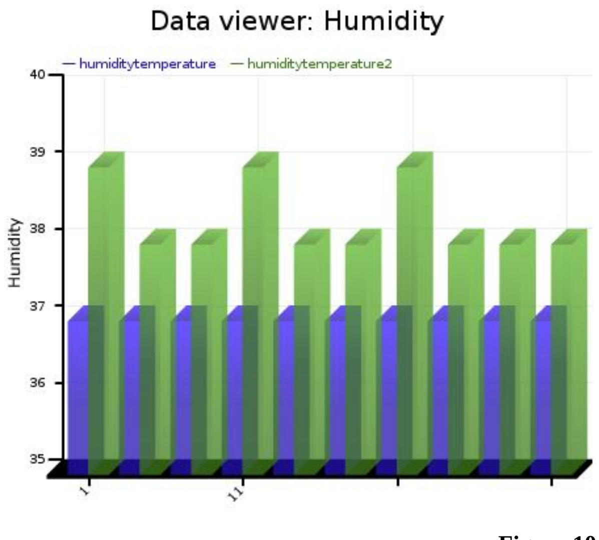


Figure 10: Humidity bar chart

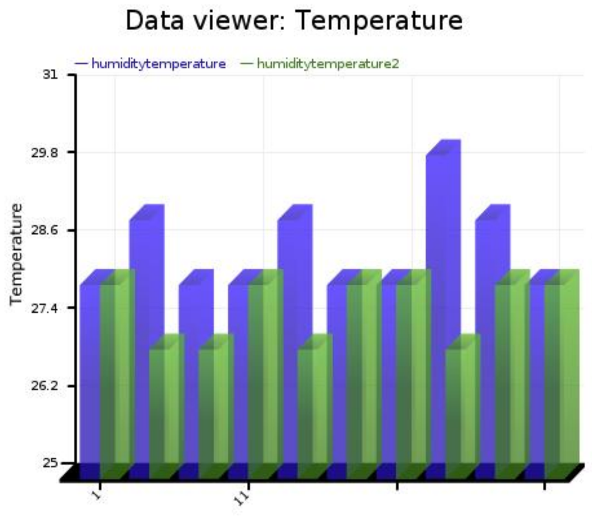


Figure 11: Temperature bar chart

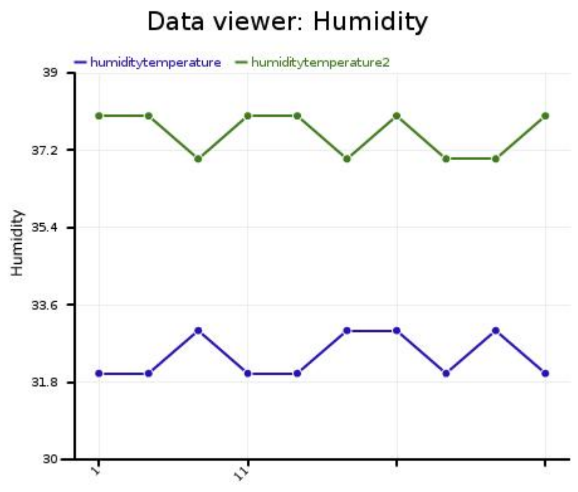


Figure 12: humidity line chart

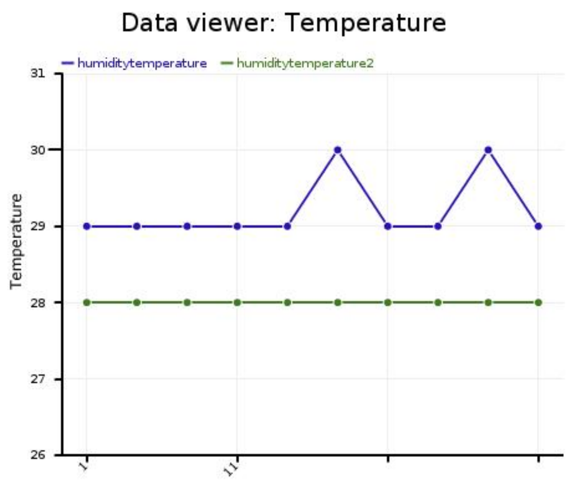


Figure 13: temperature line chart

7. ADVANTAGES AND DISADVANTAGES

A. It avoid routing queries and presents the network to the outside world as well as proccess, join and filter data output.

A. GSN proccesses common data

A. It solves heterogeneous problems.

A. GSN presents a new very intuitive GUI.

D. GSN is a system that is still being developed so we can't find too much information about it and sometimes the documentation is incomplete.

8. CONCLUSIONS AND FUTURE WORK

Our research group has performed several real wireless sensor networks experiments inclugin the Wisevine project [10]. In this project we deployed a wireless sensor network for vineyard monitoring, and we obtained several important results in terms of network performance. One of the main objetives of Wisevine project was studying the different kinds of Middlewares for sensor networks implemented in present time, as a software tool used to join platforms with the purpose of design and implement our own *Middleware*.

This Middleware is going to include QoS mechanisms. Therefore the next work is to adopt GSN as a reference to deploy our own middleware system, carrying out the purpouses of Wisevine project, but at the same time, trying to make it portable.

9. REFERENCES

[1] A. Salehi, Global Sensor Networks, Quick Start Guide.

[2] Karl Aberer, Manfred Hauswirth, Ali Salehi, Infrastructure for data processing in large-scale interconnected sensor networks, Mobile Data Management (MDM), Germany, 2007.

[3] The Albacete Research Institute of Informatics. http://www.i3a.uclm.es June, 2007.

[4] NesC programming language, http://nescc.sourceforge.net/

[5] XubunTOS Operating System Oficial Site, http://toilers.mines.edu/Public/XubunTOS

[6] TinyOS Operating System Oficial Site, http://www.tinyos.net

[7] Message Interface Generator, http://www.tinyos.net/tinyos-1.x/doc/nesc/mig.html

[8] Serial Forwarder information, http://www.tinyos.net/tinyos-1.x/doc/tutorial/lesson6.html

[9] HSQLDB data base Oficial Site, http://hsqldb.org/

[10] T. Olivares, L. Orozco-Barbosa, V. López, and P. Pedrón. Wisevine, Wireless Sensor Networks applied to Vineyards. In Proceedings of the REALWSN'06 ACM Workshop on Real-World Wireless Sensor Networks, 2006.