An Online Platform for Real-Time Sensor Data Collection, Visualization, and Sharing

PROJECTO DE MESTRADO

Salvador Martinho Rodrigues de Faria
MESTRADO EM ENGENHARIA INFORMÁTICA

UNIVERSIDADE da MADEIRA
A Nossa Universidade
www.uma.pt

Novembro | 2010

An Online Platform for Real-Time Sensor Data Collection, Visualization, and Sharing

In Partial Fulfillment of the Requirements for the Degree of
Master of Informatics Engineering

Presented to the Department of Mathematics and Engineering of the University of Madeira, Portugal by

Salvador Martinho Rodrigues Faria

in November 2010

Supervisor: Prof. Vassilis Kostakos, PhD

<u>Declaration</u>

Hereby I declare that I wrote this thesis myself with the help of no more than the mentioned literature and auxiliary means.

Up to now, this thesis was not published or presented to another examinations office in the same or similar shape.

Salvador Martinho Rodrigues de Faria

Abstract

Sharing sensor data between multiple devices and users can be challenging for naive users, and requires knowledge of programming and use of different communication channels and/or development tools, leading to non uniform solutions. This thesis proposes a system that allows users to access sensors, share sensor data and manage sensors. With this system we intent to manage devices, share sensor data, compare sensor data, and set policies to act based on rules. This thesis presents the design and implementation of the system, as well as three case studies of its use.

Acknowledgments

I would like to start by thanking my supervisor, Professor Vassilis Kostakos, for the support and guidance from the beginning to the end of the project. He provided a fruitful weekly meeting for project discussion, and shared his bright ideas which improved the project development.

My thanks also go to Professor Jose Carlos Marques, for providing access to the wine laboratory, and for all explications and information's when needed.

Lastly, I would like to thank to my family and offer my regards to all of those who supported me in any aspect, during the completion of the project.

Table of Contents

List of Figures

List of Tables

List of Software Developed or Modified

SOXLIB - SOX library is library developed by Sensor Andrew project. It provides a set of common functions and a uniform interface for all Sensor Andrew applications. This library implements the basic XMPP Pub-Sub functionalities and XMPP user administration functions.	79
SOX_TOOLS - Is a utility tool that uses SOX library and allows users to manage XMPP nodes, affiliations and subscriptions. Also allows publishing sensor data, listen, listen for actuator commands and publishing actuator commands.	69
SAWA_TOOLS - This tool is stripped version of SOX_TOOLS, and is used by the web application to retrieve data from XMPP and to make XMPP requests.	71
SAWA - The web application provides a set of common functionalities, network and sensor management, data exporting, chart generation, policies and administration.	50
ACTIONCHECKER - The actionchecker service is responsible for matching in real-time user defined policies with sensor input, and then to execute the associated notification.	41
DATARECORDER - The datarecorder service is responsible for subscribing and storing sensor data, when requested by users.	39
SCHEDULER - The scheduler is a coordinator service, responsible for managing new requests and other relevant operations. This service is also responsible for communicating with other services to delegate new tasks.	48

List of Case Studies

1 Introduction

A sensor can be defined as a device that measures a physical quantity and converts it into a signal which can be read by an observer or by an instrument [1]. They can be found everywhere, from sensor networks around the world to personal processors in our clothes that detect nutrition and hydration deficiencies in future [2].

Sensors come in all shapes and sizes, and are widely used in many fields, industrial, commercial and home applications. The term sensor covers a variety of applications and devices, which ultimately becomes a very broad concept. Sensors can be used to alert people or systems; however the majority of sensors are used to regulate and control the existing operations [3]. The most common sensors are the temperature and fire detectors in alarm systems.

Other common term is the transducer, which can be defined as “device that converts one type of energy to another” [4], and this conversion can be in different ways: electric, electromagnetic or mechanic. Transducers can be categorized by their application, whether they are used as sensor or actuator or as even as a combination.

With big advances in processor technologies and wireless communication, sensor networks and home automation systems have been growing in the last years [5,6]. Home automation is one field where the communication between sensors, actuators and supervisor system is necessary. In the home automation systems, many sensors distributed in the house collect various physical data such as temperature, humidity, motion, and light to provide information to the *HVAC* (Heating, Ventilating, and Air Conditioning) control system [7].

Researchers have already successfully built many similar sensor networking applications, but they are typically isolated, small-scale and short-lived experiments [8]. This raises the necessity of interconnecting various sensors, one good example is home automation where sensors and actuators are all connected, but these systems are at smaller scale, they use bespoke transducers, and mostly they are proprietary.

Many systems have been proposed for connecting sensors and sensor networks. We will use *Sensor Andrew* middleware as the basis of our system, and we intent to build a system to facilitate the process of sharing sensor data, and access to data. More specifically, we want the system be able to maintain the devices, whether registering sensors and actuators, allowing persons or corporations to monitor their devices and define who has access to information; present data to users in form of charts, giving a better interpretation and meaning to data; supporting groups, allowing a set of permissions and accesses; recording of sensor data when requested, enabling monitoring at long term; and user policies which verify certain sensor conditions and execute a set of actions, including calls to remote functions in actuators.

Main proposed functionalities:

- Sensor network and sensor management
- Access groups
- Data visualization
- Data recording
- Policies

With the proposed system we hope to facilitate the access to sensor data, providing a uniform and common access. We expect the system will not be used essentially on web, but we hope the system will be a good basis, to be used in smart homes and small organizations.

In this thesis we will describe our system as fallows. In Section 2 we present related work that we based our system or we get some ideas and next we describe the differences between our system and others. In Section 3.1, some hypothetical scenarios of possible use the system are described. And in section 3 we describe the system high-level architecture and the detailed information about each component of the system.

1.1 Motivation

In this section we analyze the main factors of motivation that led to the development of our work.

Sharing sensor data

Information available anywhere and anytime about the physical world has value, with sensor networks deployed across the globe, by various organization, governments, scientist and general public, it becomes clear that data sharing is an important step to get more value [9].

With increasing penetration of embedded sensors in networked devices, such as *GPS* sensor in mobile phones, enables the creation of applications that take into account the current state of the real world [10]. This kind of information may be useful to be shared in community places, such as social networks, where users may share their personal state with their friends. With such advantages in sharing, our system shall have the possibility of sharing information, either for specific users or a bigger audience, to accomplish the system will provide access policies.

Data visualization

Collected data from sensors is normally difficulty to interpret, and combined with the fact that the specialists who interpret the data are usually not experts in computers, they need an easy tool to use for manage the collection of data [9]. The application domain may help scientists to leverage computational power to simulate, visualize, manipulate, predict and gain intuition about monitored phenomenon [11].

Visual representation of data can be done in many ways, the most common are charts. Evolution charts are charts that represent one measure over an evolving dimension, time for instance. These types of charts are useful to compare new data with previous data, as well with different periods of time. Our system shall use these types of charts, being possible to be configured either by different sensors, and different periods of time.

1.2 Contribution

The broad objective of our proposed system is to simplify capturing and sharing sensor data. Having multiple systems, which operate with their own interface/tools, can be time consuming, and can be very difficult to execute the same options in all. We will develop a platform where users can add their sensor networks and sensors allowing sensor management from a common and uniform portal.

Our system will give users the possibility of adding their networks and sensors, along with the use of data recording and policies services. With the integration of disperse sensors, it will be possible to reuse sensors, for example a fire alarm siren could be used in gas leaks situations as well for another critical situations.

The system will provide a set of different notifications, and users can define their own policies, to act based on sensor input. For example, a policy can be defined to evaluate if no presence is detected in a room for more than 20 minutes, and in affirmative case it can turn the lights off. The system policies allow great flexibility, making possible to create policies that were difficult to implement without our system.

2 State of The Art & Related Work

In this section we will start by introducing the different methods for retrieving sensor data, next by introducing the wireless sensor networks and the proposed middleware projects that had in mind the sensor nodes limitations in wireless sensor networks.

Thereafter we introduce sensor web services which apply social networking principles in sensor data sharing, the smart environments which learn user behaviors to automate them, and finally a high-level middleware.

2.1 Sensor data collection

There are different methods for retrieving data from sensors, continuous, event-oriented, query-oriented and hybrid. This can depend on network design and resources constraints, like power and limited communication. In continuous method, the sensor data is sent continuously at predefined rate (e.g. temperature every hour). In event-oriented mode, the sensor data is sent when an event of interest occurs (e.g. presence in room). In query-oriented model, the applications are responsible for defining witch events are of interest and then querying sensors (e.g "select sensors where temperature > 40 and C02 > 15"). Finally the hybrid method can use more than one of the other three methods [12].

2.2 Wireless sensor networks

Sensing has evolved from manual metering to centralized data acquisition system to a new era of distributed wireless sensor networks [13]. A wireless sensor network (WSN) consists of spatially dispersed autonomous sensors nodes to cooperatively monitor physical or environment conditions, such as temperature, sound, vibration, pressure, motion and pollutants [14]. Motivated by military applications, the development of sensor networks is currently applied to industrial and civil areas, including industrial process monitoring and control, traffic control and home automation [15].

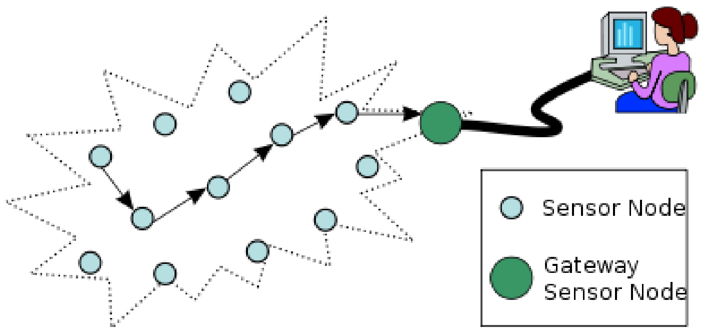


Figure 1 - Wireless Sensor Network

A sensor node, also known as mote, is a node in a *WSN*, and is capable of performing processing, gathering sensor information, and communicating with other network nodes. Sensor nodes may contain more than one sensor, and is typically equipped with a radio transceiver or other wireless communication device, a small microcontroller, and an energy source [16].

Wireless sensor networks are exposed to many technical limitations, like low power, limited computational capacity and transmission rate. In these networks, the nodes are heterogeneous and the available applications have in mind the devices constraints.

Due to these limitations, many protocol and middleware solutions have been proposed, in order to minimize the resource usage. Mostly of these solutions take in account low-level protocols and hardware implementations, making difficult to create applications and integrate multiple systems without "reinventing the wheel".

2.3 Message oriented Middleware

Mires, is a message oriented middleware (MOM), that focus in *WSN*, and was proposed in 2005, as solution to address applications requirements, considering the characteristics of sensor networks (low processing power, cost of communications).

In *Mires*, a sensor node advertises his attributes (e.g. temperature and humidity) collected from local sensors. This advertisement is routed to the sink node. User applications connected to the sink node are able to select the advertised nodes attributes. The subscription is broadcasted to sensor nodes, which will start to publish sensor data [17].

2.4 Sharing sensor data

Services like *Pachube*[1] and *SensorPedia*[2] have the objective of joining the social network principles to sensor data. *Pachube* is an innovative web service, which enables users to share and discover real time sensor data from objects, devices and spaces around the world. A user can easily register a sensor feed and start uploading sensor data.

These services have a map service, like *Google Maps*[3], where sensors can be searched geographically and be tagged. *SensorPedia* is currently in beta version and is limited to invited testers only. However, *Pachube* is online for a few years, has 10 years of evolution and has constructed a big community. This community helped the project development, and currently has available third party applications to interact with the system (mobile applications, charts, etc).

Recently *Patchube* started to offer triggers, allowing users to have *URL* event notification and experimental *SMS* notifications. The *Pachube* web service has the major advantages of being simple, a big community and a set of third party applications. As disadvantages; *Pachube* relies on Representational State Transfer (REST) paradigm, no support for actuators and has basic and limited triggers.

Our system is similar to *Pachube*, in the way it has the same objectives, allowing users to easily share sensor data and visualize them. Besides the similar objectives, our system allows a high scalability, extensibility, security and privacy in the middleware. Our system allows multiple entities to subscribe sensor data in a push method, define user access type and groups, encryption and other relevant *XMPP Pub-Sub* features[4]. In the web interface, it is possible to manage sensors networks and devices, allowing users to add any sensor and actuator device with images and related information. From the web interface it is possible to directly call commands in the actuators and create advanced policies, which can use different types of rules and actions.

[1] www.pachube.com
[2] www.sensorpedia.com
[3] http://maps.google.com
[4] http://xmpp.org/extensions/xep-0060.html

2.5 Smart environments

A smart environment can be defined as a small world where different kinds of smart devices are continuously working to make inhabitants lives more comfortable [18], and its major goal is to anticipate actions of a human inhabitant and then automate them [19]. Smart environments are a recently and promising area of study, it focuses in making use of context-aware data from environments to predict inhabitants habits and then automate them.

Smart environments are classified by having remote control of devices, device communication, information acquisition/dissemination, enhanced services by intelligent devices, and predictive and decision-making capabilities [18].

Smart environments can be used in many useful ways, like inhabitant's safety; resource optimizations (e.g., energy consumption); task automation and can be very useful for disabled people. We could imagine arriving at home, and then the doors open automatically, the lights turn on if the light intensity is low, the TV turn in the favorite TV program at that specific hour.

Currently in many systems, these kinds of events are only user defined; the users expressly define the rules and respective actions (static policies). Static polices work in most of the situations, but they do not adapt to the changing environment and require user to change these policies. The smart environment is a solution to introduce more dynamism to the user defined policies. As cited in article [19], it is possible to learn human actions using sensors, and then make predictions to execute actions.

Automation can be viewed as a cycle with two phases, with the perception phase and the actuation phase. In smart environments, the perception of an environment is a bottom-up process, sensors and physical devices continuously monitor de environment, and upload the data to a central database. The data archived on database is mined and processed by specific applications to knowledge (models, patterns, etc). Finally, hidden knowledge is used for intelligent decision making and subsequent actions [20].

Our system does not address smart environments, but due to nature of the used middleware, in terms of scalability and extension, it is relatively simple to integrate an artificial intelligence agent to cooperatively work with our platform. An intelligent agent (IA) could benefit from the middleware and the platform, the platform has user defined rules and centralizes all sensor data in a database. The central database would be used for data mining, and the platform policies, could be used to rate/disable/limit the IA actions.

2.6 Sensor Andrew

Sensor Andrew is a scalable campus-wide sensor middleware, developed with the objectives of supporting ubiquitous large-scale monitoring and infrastructure control [21]. The *Sensor Andrew* is a project developed by Carnegie Mellon University in order to integrate multiple systems, and was designed to be extensible, easy to use, and secure while maintaining privacy.

Sensor Andrew middleware allows application developers to be able to transmit sensor data with no need to re-invent lower-level interfaces.

Sensor Andrew design goals:

- **Ubiquitous large-scale monitoring and control**: support for sensing and actuation.
- **Ease of management, configuration and use**: easy to use, manage and develop applications.
- **Scalability and extensibility:** support for any device, and support extensions
- **Built-In security and privacy:** support for security and privacy, encryption, key management, access control and user management.
- **Infrastructure sharing**: allow application to reuse of infrastructure devices.
- **Evolvability**: support for different computational paradigms and support changes.
- **Robustness**: built-in robustness and able to reconfigure itself.

For the communication protocol, they defined as requirements standard messaging protocol; extensible message types; point-to-point and multicast messaging; support for data tracking and/or event logging; security, privacy and access control; and internet-scale. Given the requirements, they used the *e**X**tensible **M**essaging and **P**resence **P**rotocol* (XMPP). *XMPP* is an open protocol, based in *XML*, and traditionally used in messaging applications. In Section 2.7.2 we detail more about the *XMPP* protocol.

Like our requirements, *Sensor Andrew* focuses on high-level applications, and not on low-level sensor protocols which addresses the constraints of sensor nodes and specific technologies. We selected *Sensor Andrew* solution as our starting point, because many of our requirements were met by its design.

Next we describe the *Sensors Over XML* extension, the XMPP, XMPP protocol, and finally the XMPP *Pub-Sub* extension.

2.6.1 Sensors over XMPP

The *Sensors Over XMPP* (SOX) library is developed by Sensor Andrew project; it provides a set of common functions and a uniform interface for all Sensor Andrew applications. This library implements the basic XMPP Pub-Sub functionalities and XMPP user administration functions. This library implements the Sensor Andrew XML schema available at our project wiki[5] page; this XML schema allows devices to "talk" the same common language.

SOX library main dependencies are *Glib*[6] and *Loudmouth*[7], the last one is a lightweight C library for programming with XMPP protocol, this lightweight library provides a set of functions to interact with XMPP server, including *TLS/SSL* security protocols. In the other side, *Glib* is a cross-platform utility library which provides advanced data structures, doubly and singly-linked lists, hash tables, dynamic strings and string utilities, and other advanced functions [22].

Create XMPP connection	Subscribe
Create new user	Unsubscribe
Delete user	Retrieve subscriptions
Create node	Create sox message
Delete node	Publish sox message
Retrieve last published item	Send direct message

Table 1 - SOX library main functions

[5] http://hci.uma.pt/wiki/index.php/SAWA#Documents
[6] http://library.gnome.org/devel/glib/stable
[7] http://lexs.it.cx/loudmouth

2.7 XMPP

The *Extensible Messaging and Presence Protocol* is an open protocol based on *Extensible Markup Language* (XML), designed for real time communications, being the XML the base format for exchanging information. With XMPP protocol it is possible to support a vast quantity of services, such as, channel encryption, authentication, presence, contact lists, one-to-one messaging, multi-party messaging, service discovery, notifications, structured data forms, workflow management and peer-to-peer media sessions [23].

XMPP is used by many types of applications, instant messaging, multi-party chat, voice and video calls, collaborations, lightweight middleware and content dissemination [24,25], also expanded to the domain of message-oriented middleware's. Built to be extendible, the protocol has been extended[8] with many features, including the *Publisher-Subscriber* paradigm.

For example, *Google Talk* and *Google Wave* use XMPP protocol and extensions for information interchange. Referring this article [26], by 2003 was estimated that software using XMPP was installed in hundreds of server across the internet and be used by ten millions of people.

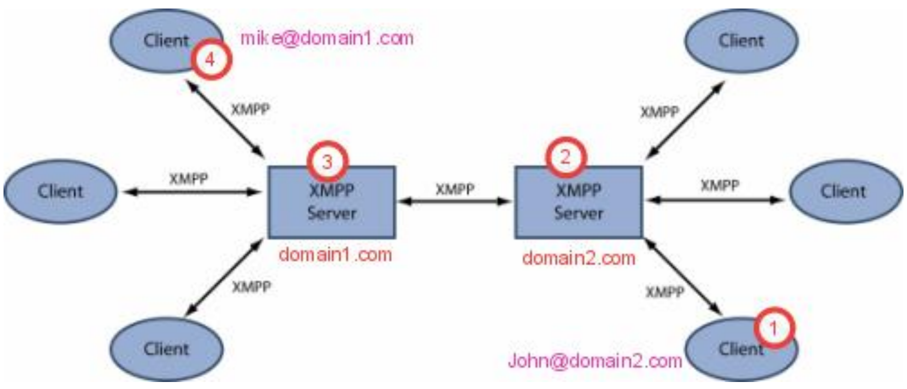


Figure 2 – Sending XMPP Message

[8] http://xmpp.org/xmpp-protocols/xmpp-extensions

Figure 1 shows how XMPP uses a decentralized addressing, making it highly scalable, in the same way a domain can run its own email server. Addressing in XMPP is defined first with a client identification (referred to as a JID) followed by a domain name and then a resource name [21].

salvador@hci.uma.pt/home

Messages are not sent between clients, but instead are sent using the client XMPP server. XMPP is both, a client-to-server and server-to-server protocol [27,28]. When a client sends a message, the message is sent to the client local server (1 to 2), if the receiver XMPP client is using the same XMPP server, then server will send the message directly to receiver recipient, if it uses another XMPP server, the local XMPP server will send the message to the corresponding server (2 to 3). Finally the message is sent when the message reaches the server where the receiver recipient is in the same XMPP server (3-4). For more technical details about the message protocol, please check the XMPP core specifications[9].

Advantages of XMPP:

Open Standards - the XMPP protocols are free, open, public, and easily understandable; in addition, multiple implementations exist in the form clients, servers, server components, and code libraries [25,29].

Decentralized - the architecture of the XMPP network is similar to email; as a result, anyone can run their own XMPP server, enabling individuals and organizations to take control of their communications experience [25,29].

Security - XMPP has built-in support for channel encryption and strong authentication, inherent resistance to many forms of malware. Any XMPP server may be isolated from the public network, robust security using SASL and TLS has been built into the core XMPP specifications, and the XMPP network is virtually spam-free [25,29].

[9] http://xmpp.org/rfcs/rfc3920.html

Flexibility & extensibility - Custom functionality can be built on top of XMPP; to maintain interoperability, common extensions are managed by the XMPP Software Foundation. Because XMPP is at its core a technology for rapidly delivering XML from one place to another, it has been used for a wide range of applications beyond instant messaging, including network management, content syndication, collaboration tools, file sharing, gaming, remote systems monitoring, web services, lightweight middleware, cloud computing, and much more [25,29].

Weakness of XMPP:

In-band binary data transfer is inefficient - Because XMPP is encoded as a single long XML document, binary data must be first base64 encoded before it can be transmitted in-band. Therefore any significant amount of binary data (e.g., file transfers) is best transmitted out-of-band, using in-band messages to coordinate [25].

2.7.1 Streaming XML

XMPP is, in essence, is a technology for streaming XML. When you want to start a session with an XMPP server, you open a long-lived TCP connection and then negotiate an XML stream to the server [30].

XMPP is built on top of three main blocks (<message/>, <presence/> and <iq/>), these blocks are called *stanzas* and all others stanzas are build on top of these primitives. An XMPP session consists of many XML stanzas sent over an XML stream; each XML session has two streams, one stream from client to server and another stream from server to client. After completing an XML stream negotiation with the server, the client and the server can send an unlimited numbers of stanzas over the stream [30].

In the following example we exemplify an XMPP session. The client requests are marked as green and server response in red. To start a XMPP stream with XMPP server the client sends a *<stream:stream>* element. The first stanza sent is *<presence>*, which sends the client presence to server; this stanza can have attributes, such as client status (away, working). The second stanza used is *<iq>* (information query) to retrieve the roster contacts. Then server retrieves user contacts from user roster, and in this case the roster list only contains two users, Mike and Julia.

Client

```
<stream:stream>

   <presence/>

   <iq type='get'>
      <query xmlns='jabber:iq:roster'/>
   </iq>
```

Server

```
   <iq type='result' to='sapo@hci.uma.pt/mobile'>
      <query xmlns='jabber:iq:roster'>
         <item jid='mike@hci.uma.pt' name='mike'/>
         <item jid='julia@hci.uma.pt' name='julia'/>
      </query>
   </iq>
```

The client sends a message to user Mike, using the *<message>* stanza and sometime after, the client is notified with the contact response coming from server. Before closing the session by sending the *</stream:stream>* element, the user updates his presence to unavailable using the presence stanza.

Client

```
<message type='chat' to='mike@hci.uma.pt'>
    <body>hey, are you there?</body>
</message>
```

Server

```
<message type='chat' to='sapo@hci.uma.pt/home' from=mike@hci.uma.pt/hp'>
    <body>yes, what’s up?</body>
</message>
```

Client

```
    <presence type="unavailable"/>
</stream:stream>
```

After this brief example, in the next topic we pass to describe in more detail the communications primitives. In section 0 we introduce the Publisher-Subscriber paradigm and in section 2.7.4 we describe the XMPP Publisher-Subscriber extension.

2.7.2 XMPP Protocol

In previous section we introduced XMPP protocol by exemplifying a XMPP stream with the primitive stanzas. A stanza is no more than a basic packet for communication, similar to packets and messages in other network protocols. In this section we will introduce each stanza and explore the properties, the type of attributes and payload definitions.

Communication primitives:

Presence - The presence stanza reports the availability of an entity in a network, this availability can be more complex than online and offline status. Presence is published under the paradigm of Publisher-Subscriber, allowing user to subscribe or unsubscribe user presence.

Example of a Presence stanza with status:

```
<presence from="salvador@hci.uma.pt/laptop">
        <show>xa</show>
        <status>down the rabbit hole!</status>
</presence>
```

Message - Message stanzas are a basic method for sending messages from one entity to another. They are used mostly for Instant Messaging, group chat, and other similar applications. In this type of stanza, the messages are typically not acknowledged.

chat	One-to-one online chat, most common in IM applications
groupchat	Type of messages exchanged in multi-user chat rooms
normal	Default type for messages, similar to email messages
headline	This type of message are used to send alerts an notifications
error	This messages are sent when any entity detects a problem

Table 2 - Message Types

Messages may contain more than normal message text. They can carry application specific payloads or any kind of formatted data. The following example shows a message stanza, with chat type, and caries application specific payload (geoloc).

```
<message from=”salvador@hci.uma.pt/laptop” to="mik@hci.uma.pt/home" type="chat">

        <body>Who are you?</body>

        <geoloc xmlns='http://jabber.org/protocol/geoloc' xml:lang='en'>
                <country>Italy</country>
                <lat>45.44</lat>
                <locality>Venice</locality>
                <lon>12.33</lon>
        </geoloc>

</message>
```

IQ – The majority of XMPP Instant Messaging traffic is composed of message and presence packets, most of the work in implementing an IM client or server lies in supporting a variety of administrative and management protocols that support messaging and presence [31]. XMPP addresses these features with the generic protocol Information/Query (IQ). The *<iq>* stanza provides a request and response mechanism for XMPP communication. Like in messages, IQ stanzas come in four flavors, differentiated by the stanza's type attribute.

get	The entity asks for information, similar to HTTP GET.
set	The entity provides some information or makes a request, similar to HTTP POST.
result	The responding entity returns the result of a get operation, or acknowledges a request.
error	The responding entity notifies the requesting entity when a problem is detected (malformed message, not implemented, permissions).

Table 3 - IQ Types

Example: Requesting roster contacts. In this example, the user asks the XMPP server to retrieve his contact list, by sending an IQ-get stanza, containing an empty payload qualified by the *jabber:iq:roster* namespace.

```
<iq type='get'>
        <query xmlns='jabber:iq:roster'/>
</iq>
```

After receiving the IQ request, server replies with a non-empty payload qualified by the same namespace.

```
<iq type='result' to='sapo@hci.uma.pt/mobile'>
        <query xmlns='jabber:iq:roster'>
                <item jid='mike@hci.uma.pt' name='mike'/>
                <item jid='julia@hci.uma.pt' name='julia'/>
        </query>
</iq>
```

2.7.3 Publisher Subscriber

Publisher-Subscriber or Pub-Sub is a paradigm of asynchronous communication, based on notifications by events. This paradigm has three main components, publishers, subscribers and event nodes. The publishers are the entity who publishes information; the subscribers are the entities who consume information. And event nodes are virtual channels, where the information is published and consumed. These nodes can have many properties, such as access permissions, subscription options, members and storage policies [32,33]

This paradigm provides the possibility of consumers subscribe to events, to be notified when new information is published [34]. As Figure 3 shows, the Pub-Sub paradigm is much more efficient than Representational State Transfer (REST) paradigm, to retrieve data from server without knowing the data availability.

The loosely coupling between publisher and subscribers, allows a higher scalability and more dynamism in the network topology [32]. Due to this nature, of scalability and flexibility, the publisher-subscriber middleware's are becoming popular for distributed applications [33].

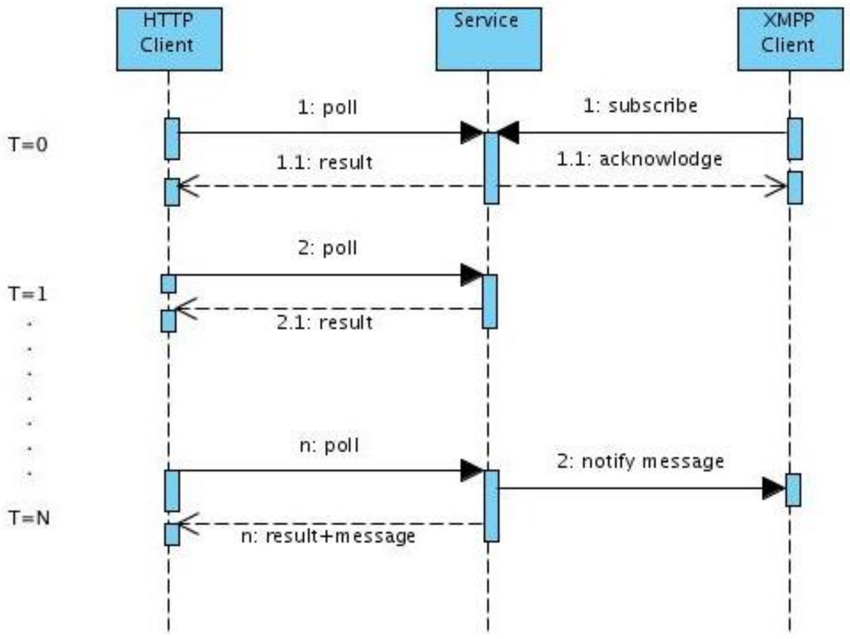


Figure 3 - PubSub vs HTTP

2.7.4 Publisher-Subscriber in XMPP

XMPP has currently over 150 published extensions[10], including the publisher subscriber extension (XEP-060). The Pub-Sub extension defines a generic protocol, enabling any application to implement the most basic Pub-Sub features. In this section we will describe some features associated with nodes, publishers, subscribers, and exemplify basic Pub-Sub operations using IQ stanzas.

Affiliations – an affiliation defines the role that a user has in relation to a node, it can be the owner, publisher, member, or other. The Pub-Sub extension currently has six different types of affiliations, and an application may not be required to support all types [35].

Affiliation	Subscribe	Retrieve Items	Publish Item	Delete Item	Configure Node	Delete Node	Purge Node
Owner	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Publisher	Yes	Yes	Yes	Yes	No	No	Yes
PublishOnly	No	No	Yes	No	Yes	No	No
Member	Yes	Yes	No	No	No	No	No
None	Yes	No	No	No	No	No	No
Outcast	No	No	No	No	No	No	No

Table 4 - Node Affiliations and Privileges

Nodes - nodes are virtual channels, where information can be published and subscribed; each node has an access model and type. A node by default is created with default configuration, but is possible to specify or change a node configuration. This configuration may contain options like: max_items, item_expiration, max_payload_size, notify_delete, and others more.

Another aspect of nodes, it is the hierarchy, a node can be the type of leaf node, where information is published and can be consumed; or the type of collection node, this type of nodes only allow subscription and are used to form an hierarchy structure. The collection feature is supported by XEP-0248[11] extension, this extension defines subscription for collection of nodes, making the subscription process simpler when an entity is interested in notifications from a set of nodes [36].

[10] http://xmpp.org/extensions
[11] http://xmpp.org/extensions/xep-0248.html

Node Type	Description
Leaf	A node that contains published items only, and cannot contain other nodes.
Collection	A node that contains nodes and/or other collections, but no published items.

Table 5 - Node Types

Access Model	Description
Open	In this access mode, any entity may subscribe to the node, without any subscription approval. And any entity may retrieve items without being subscribed.
Presence	This access model applies mainly to the instant messaging systems. Any entity with a subscription of type "from" or "both" may subscribe to the node and retrieve items from the node
Roster	This access model applies mainly to instant messaging systems. Any entity in the specific roster may subscribe and retrieve items from node.
Authorize	The node owner must approve all subscription requests, and only subscribers may retrieve items from the node.
Whitelist	A node with this access model has list, the users who are in the list, are allowed to subscribe or retrieve items from node. The node owner is responsible for managing the list, and can add or remove members.

Table 6 - Node Access Models

In the following example we define an IQ stanza, to create an XMPP Pub-Sub node named “madeira”, with *whitelist* access model.

```
<iq type='set' to='pubsub.hci.uma.pt' id='create1'>
        <pubsub xmlns='http://jabber.org/protocol/pubsub'>
                <create node='madeira' access='whilelist'/>
        </pubsub>
</iq>
```

IQ stanza to affiliate user *“julia@hci.uma.pt”* as publisher in node “madeira”.

```
<iq type='set' to='pubsub.hci.uma.pt' id='affil1'>
        <pubsub xmlns='http://jabber.org/protocol/pubsub#owner'>
                <affiliations node='madeira'>
                        <affiliation jid='julia@hci.uma.pt' affiliation='publisher'/>
                </affiliations>
        </pubsub>
</iq>
```

IQ stanza to publish the book status in node *"latest_books"*.

```
<iq type='set' to='pubsub.hci.uma.pt' id='publish1'>
        <pubsub xmlns='http://jabber.org/protocol/pubsub'>
                <publish node='latest_books'>
                        <item>

                                 <book xmlns='jabber:x:data' type='result'>
                                        <field var='title'>
                                                <value>XMPP for Noobs</value>
                                        </field>

                                        <field var='status'>
                                                <value>available</value>
                                        </field>
                                </book>

                        </item>
                </publish>
        </pubsub>
</iq>
```

IQ stanza to subscribe from *"madeira" node.*

```
<iq type='set' to='pubsub.hci.uma.pt' id='publish1'>
        <pubsub pubsub xmlns='http://jabber.org/protocol/pubsub'>
                <subscribe node='madeira' jid='julia@hci.uma.pt/home'/>
        </pubsub>
</iq>
```

IQ stanza to delete node *"madeira"*.

```
<iq type='set' to='pubsub.hci.uma.pt' id='del1'>
        <pubsub xmlns='http://jabber.org/protocol/pubsub'>
                <delete node='madeira'/>
        </pubsub>
</iq>
```

3 System Description

This section gives a high level overview of system architecture, starting with example scenarios to exemplify system's potential. Secondly, we demonstrate a high view of architecture, and finally we describe each component individually. By the end of this section, we finalize with a description of system particularities and technical details.

3.1 Example Scenarios

For example, it is possible to create a policy to alert a person via SMS, to notify when a value is detected, which can be the concentration of a gas. As example, Mike has a butane gas sensor in the kitchen, when by any reason a gas leak happens the system will inform Mike about the situation.

The system has big potential for real-time uses, for non real-time, it can be used also as data recording tool. A good example is monitoring energy consumption. It is possible to monitor many situations, like average time and frequency of televisions, computers, and lights turned on without any person present.

Using GPS technology is possible to create policies to detect when a device enters or exits a geographical area. Considering a bus sending its GPS position every minute, for instance, when a bus is detected in special area, let's say a bus stop, an action can be triggered with many purposes.

A more interesting example is to have a policy with geographical and history rules, for example, when detecting a new user via Bluetooth technology in a touristic place, and sometime after, detect the same user in another touristic place, the system could send a message via Bluetooth with touristic information or nearby shop information's.

The system can be used to optimize energy consumption and to execute common human actions. It is possible to detect a person in a room and the system can turn automatically the lights on if the luminosity is sufficient (e.g. during day). And turn off the same lights when presence is not detected. Thus avoiding users to repeat continuously the same everyday actions as well can reduce energy consumptions.

3.2 System requirements

The initial objectives were to build a middleware, where people could add sensors and actuators, read data from sensors and control actuators.

First functional requirements:

R1. Read data from sensors
R2. Control actuators
R3. Report status of environment
R4. Act based on policies
R5. Support for multiple sensors and actuators

After some search in related work, we found that many of these requirements were addressed in existing solutions. We selected Sensor Andrew middleware solution as the starting point of our work, since many of the first requirements were meet and to avoid "reinventing the wheel" we suggested implementing a platform, which used the middleware benefits with a web interface to users easily manage their devices.

Sensor Andrew is a campus-wide high-level middleware that supports sensing and actuation (meets R1and R2). Allows the registry and use an unlimited number of installations, sensors and actuators (meets R5).

New functional requirements:

- **Support user authentication** – The user's access platform should be authenticated before accessing platform functionalities and sensor data.
- **Support account creation** – The system should be able to create new user accounts.
- **Support device registration** – The system must allow users to add sensors and actuators.
- **Supports sensor search** - The system should have a mechanism to search for sensors.
- **Supports definition of device access** – The system should allow only authorized users to perform operations with user's devices.
- **Supports sensor data recording** – The system should be able to store sensor data.

- **Supports sensor data download** – The system should allow user to download their sensor data in multiple formats and from allowed devices.
- **Supports charts generation** - The system should support the generation of different charts using sensor data.
- **Supports creation of Policies** – The system should allow users to define policies, and match every sensor data against these policies. And execute the defined actions when a policy matches.

3.3 High Level Architecture

In order to understand system functionalities, we present a keyword list, with a brief description to make a simpler understanding.

Web application – The component responsible to represent all information to users, and enable users to manage their devices and associated options.

Datarecorder – The component responsible for storing sensor data, when requested by users.

Actionchecker – The component responsible for matching sensor input to existing policies, and trigger actions when a policy matches.

Scheduler – The component responsible for handling new recording requests and new policies. It is also responsible to communicate with datarecorder and actionchecker services to enable these new requests.

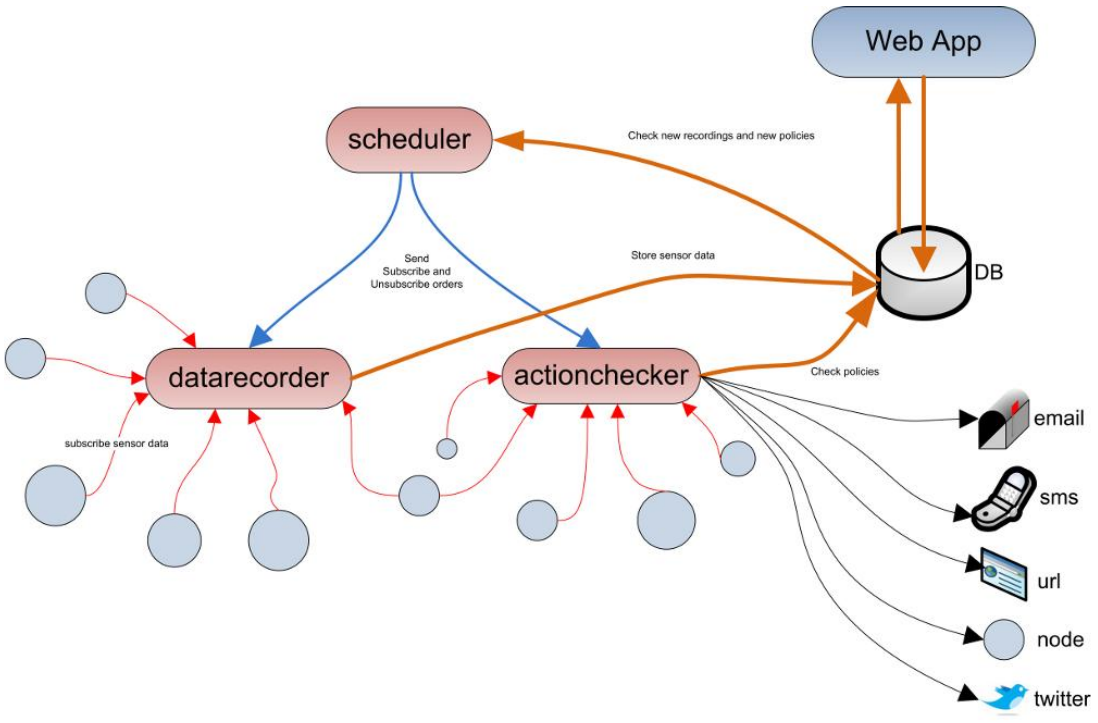


Figure 4 - High-Level Architecture

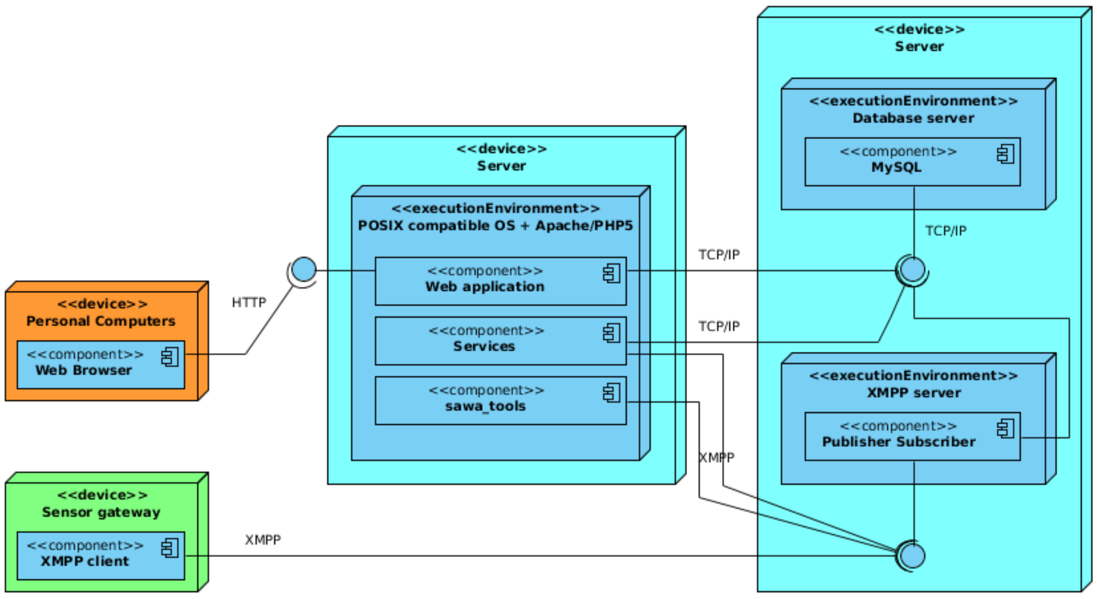


Figure 5 - Deployment Diagram

In Figure 5 is represented a high-level architecture of our system using a deployment diagram. Our system mainly consists in two parts: the middleware, composed by the XMPP server with publisher-subscriber support, which is responsible for transporting and handling all sensor data. The other part consists in the applications (web application and services) that interact with the middleware.

The architecture style of our system is a shared repository, being this repository passive, unlike the blackboard architecture style. Architectures based on repository allow a loosely coupling between system components. All interactions are made through database, allowing a greater facility in modifying system components without affecting other components.

The web application interacts with the middleware using a dedicated XMPP client, but most of the work done by the web application is stored in repository. As Figure 4 shows, the services also use the repository to modify, read and create new entries. The services datarecorder and actionchecker connect to the XMPP server and the repository; these services listen for published data in Pub-Sub nodes and then process the sensor data according to user definitions in repository.

Given a brief overview of the high-level architecture, we now proceed to describe individually each component in the system.

3.4 System components

3.4.1 Services

In our system we developed three services, the datarecorder, the actionchecker and the scheduler service. The datarecorder service allows users to record sensor data, enabling users to download sensor data or generate charts from the web application. On the other hand, actionchecker is a service which enables to execute certain actions when a certain group of rules matches. Finally scheduler is a utility service that is responsible for checking and resolving new user requests.

3.4.1.1 Datarecorder

The datarecorder service, written in C and using cross-platform utility library *GLIB*, is responsible for recording sensor data to database. The main dependencies of this service is *GLIB*, a cross-platform C library that contains functions like data structure and string functions. The other dependency is a modified *SOX* library, which is used for establishing XMPP connections and parse messages according to XSL schema.

Datarecorder process starts by reading the main configuration file, in order to read database connection settings (host, username and password), and repeats the same operation for XMPP connection settings. After reading the configuration file, the process tries to establish a database connection, and a long lived XMPP connection to receive data from sensors.

For communicating with scheduler service, a socket named "datarecorder.sock" is created in to receive subscribe/unsubscribe orders.

At the time of the creation of XMPP connection, a callback function (handle_event) is specified to parse messages. The *handle_event* function receives a pointer to an *LmMessage* data type.

```
static void handle_event(LmMessage* message)
```

The received *LmMessage* is parsed using XML parser *Expat*, by setting a parser handler function for starting and end tags of XML elements.

```
XML_SetElementHandler(p, startElement, endElement);
```

The next XML block, is an example of a received message payload.

```
<System xmlns="http://jabber.org/protocol/pubsub">
        <DeviceInstallation id="6" regid="6"  timestamp="1281526867">
                <TransducerInstallation name="" id="6" regid="6" canActuate="false">
                        <TransducerValue
                                                typedValue="12.13"
                                                rawValue=""
                                                timestamp="1281526867"
                                                variableId="14"
                                                unit="">
                        </TransducerValue>
                </TransducerInstallation>
        </DeviceInstallation>
</System>
```

The XML parser handler verifies if the message contains "DeviceInstallation" as element, in affirmative case, the XML element attributes are copied, and the same steps are produced for "TransducerInstallation" elements. If "TransducerInstallation" contains a child element "TransducerValue", then **this message is recognized as message with sensor data**. The values are copied and then the sensor value is added to a SQL insert query. This insert query will only insert if there is at least one recording request. If the message does not contain sensor values, the message is ignored.

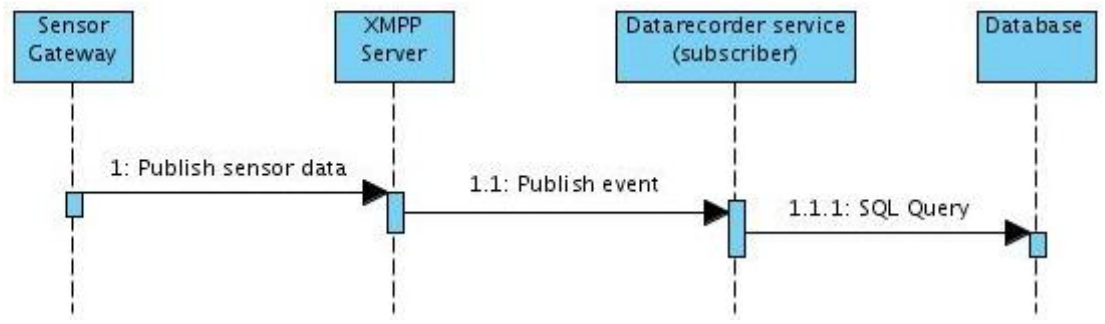


Figure 6 - Datarecorder Service

The sequence diagram in Figure 6 exemplifies the steps executed for recording the sensor data. A publisher (sensor gateway) publishes sensor data and the XMPP server accepts the message and then replies for all subscribers. When any message arrives to datarecorder, the process parses the message, looking for sensor data, and then sends an SQL query to insert the sensor in database.

To avoid unnecessary waste of bandwidth and CPU processing, the datarecorder service only subscribes to nodes, where sensor data is published from sensors with recording requests. The scheduler process takes the job of verifying which nodes to subscribe or unsubscribe and manage the active subscriptions when the services start.

The datarecorder service runs a socket, where it can receive “subscribe” and “unsubscribe” orders from the scheduler process. The message format is defined as follow:

```
#subscribe#node1#node2#node3#....#nodeN
#unsubscribe#node1#node2#node3#....#nodeN
```

When messages with the exemplified format are received, the service will subscribe/unsubscribe to specified nodes in message. The request is made to the XMPP server using the appropriate protocol, and when the approval arrives, the service is able to receive message from these new nodes.

3.4.1.2 Actionchecker

Actionchecker is also a service written in C, and has the responsible for matching sensor values to existing policies. It starts by reading connection settings from the main configuration file, and loads a set of specified action plug-ins. After the registration of plug-ins, a XMPP and database connections are established.

For every received event, actionchecker starts by querying the database, to see if there is any policy with the sensor in rules. If there is any policy, the service retrieves all policies rules to start matching. If there is any policy with all rules matching, it means that policy matches. When a policy matches, the process delegates the notification execution on plug-ins, these plug-ins are responsible for communicating with external services, like email, SMS, URL’s and other kind of tools and services.

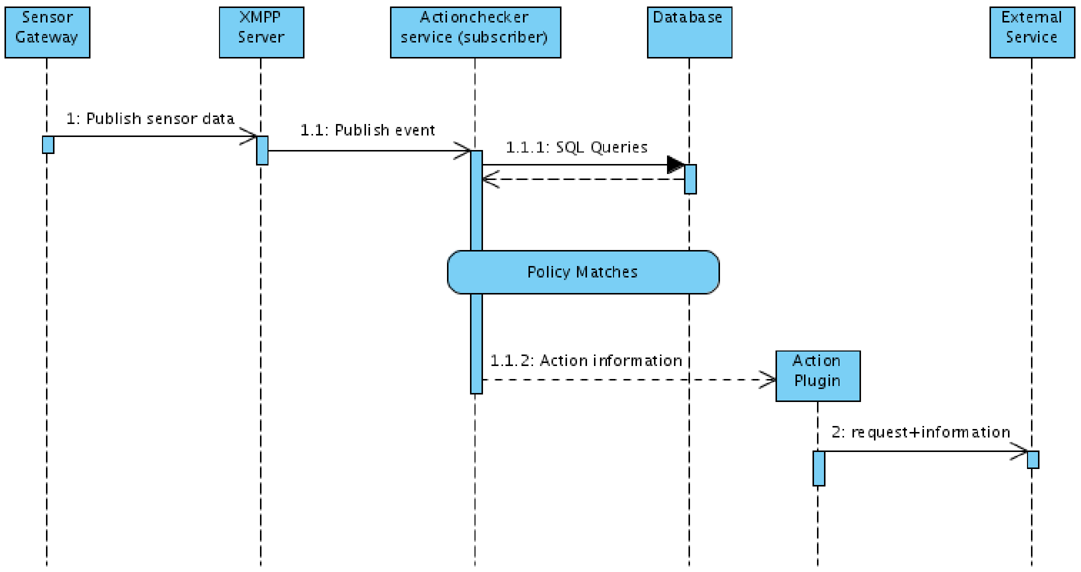


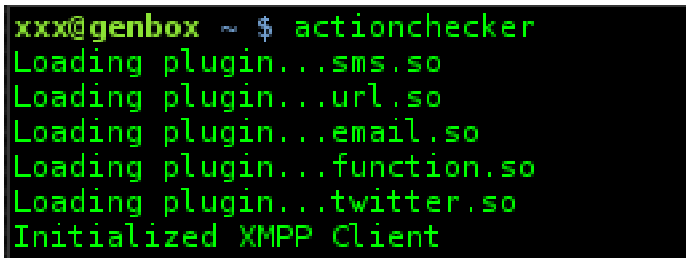
Figure 7 - Actionchecker Notification Execution

The execution of notifications can take a few seconds or a few more time, in order to handle this situation with more precaution, a pool of threads is used to handle the execution of notifications. Using thread functions available in *GLIB*, we can set restrictions to these threads, such as thread reuse, pool size, and other options.

The actionchecker service also accepts “subscribe” and “unsubscribe” orders from scheduler, by activating a socket and handling socket messages in the same way as datarecorder process.

3.4.1.3 Actionchecker Plug-ins

The advantages of using plug-ins are well known, they are easy to deploy, they are small, and they increase the extensibility of applications. To achieve this functionality, we used *GModule* functions, which provide a portable way to dynamically load object files.



```
xxx@genbox ~ $ actionchecker
Loading plugin...sms.so
Loading plugin...url.so
Loading plugin...email.so
Loading plugin...function.so
Loading plugin...twitter.so
Initialized XMPP Client
```

Figure 8 - Initializing Actionchecker

3.4.1.3.1 Email plug-in

Email is a very used communication method and is used in almost every notification system. The email plug-in was the first developed, we now exemplify in two lines of code how the plug-in can be easily created. First we use a command line tool called "mail" to send emails, this command line tool is a Mail User Agent (MUA), which can be used to read and send emails.

In first line of code, a command line string is formatted using the message, subject, destination address and the body message. In the second line the plug-in tells the operating system to execute the created command line.

```
sprintf(fpBuffer, "echo '%s' | mail -s '%s' %s", message, subject, to);
system (fpBuffer);
```

3.4.1.3.2 URL plug-in

A good notification can be the URL notification, a simple and basic notification, but can be very useful to communicate between two unknown systems. Using the HTTP GET method is possible to notify another web application, using a given URL with or without parameters. Like in email plug-in, we also used a command line tool called "GET", which is a command line tool that implements HTTP Get Protocol.

As example, let's consider a web application running in host example.com that turns lights on and off.

http://example.com/lights/set/33/off

With the given example, we can set a policy with URL notification, which turn on or off lights, in a simpler and firewall friendly way.

3.4.1.3.3 Function plug-in

The function plug-in enables the publication of events in XMPP nodes, in particular, it allows the publication of commands to actuators, such as “activate”, “turn_off”, “increase”, or any possible command.

The Sensors Over XMPP extension has the “TransducerCommand” element defined, which is intended to send commands to actuators.

As stated, a function can have arguments, but a simple command can be like “00129”, these strategies can be chosen by the users. Considering the light example, and we want to turn lights on for a period of time, we could send a message with the function name as “activateLight(int)”.

Example of a message payload directed to an actuator with id 7.

```
<System xmlns="http://jabber.org/protocol/pubsub">
        <DeviceInstallation id="4" regid="4" type="" description="" timestamp="1281545117">
              <TransducerInstallation name="" id="7" regid="7" canActuate="false">
                     <TransducerCommand
                                                name="activateLight(int)"
                                                value="33"
                                                timestamp="1281545117">
                     </TransducerCommand>
              </TransducerInstallation>
       </DeviceInstallation>
</System>
```

When the message is received, the gateway parses the XML message to get the command or function, and can easily identify if the function contain arguments. After that, the sensor gateway is responsible for calling the local actuator with the right communication protocol. Optionally the sensor gateway may report the new status of the actuator, such as “running” or “active”.

3.4.1.3.4 SMS plug-in

Short Message Service (SMS) is a widely spread text communication service, allowing people to send text messages to other people. With the spread of mobile phones, almost everyone has an instant access to SMS messages. Unlike Email, SMS is a paid service, and to send SMS messages, there is two possible ways, using a GSM modem, or use SMS Gateway providers.

Meanwhile there is an experimental and non-reliable Perl script, which connects to a mobile provider client web page, and sends SMS messages from that page. After testing the script, we decided to use the script as experimental SMS plug-in.

In SMS plug-in, well call the Perl script to send an SMS message for the specified destination along with message body and user credentials.

```
gchar *cmd;

cmd = g_strconcat("perl ", script_path, " ", user, " ", pass, " ", destination_number, NULL);
cmd = g_strconcat( cmd, " ", msg, "' > /dev/null", NULL);

system(cmd);
g_free(cmd);
```

The use of SMS notifications is better than email notifications in some points, GSM coverage, and availability, making a good choice for notifications in critical situations (e.g., gas leaks, fire, motion detected). In Figure 9, is shown a received SMS message, alerting that a mail was received in mail box.

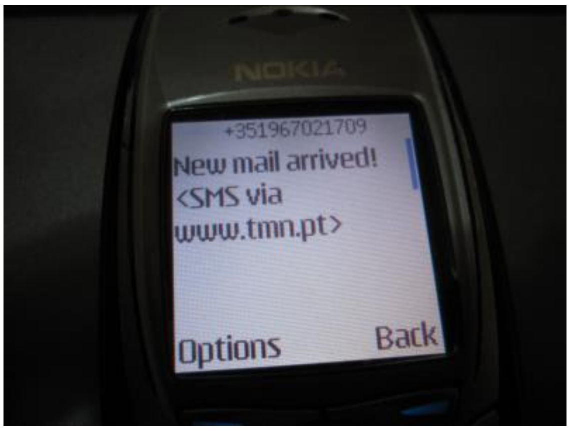
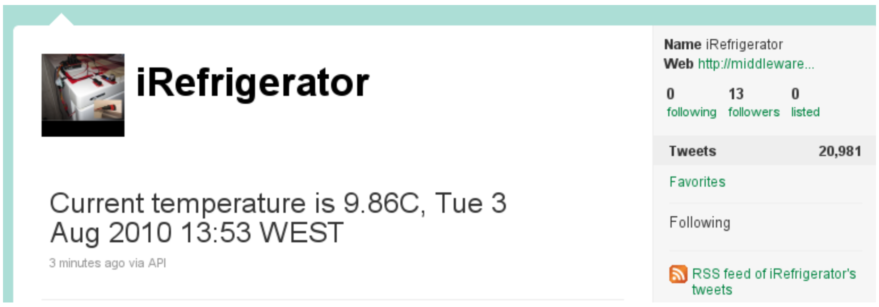


Figure 9 - SMS Notification

3.4.1.3.5 Twitter plug-in

Twitter[12] is a social networking and micro blogging service that enables its users to send and read messages known as tweets [37]. We created a plug-in for sending post's to this social network.

Following the *official tutorial*[13] we developed the twitter plug-in which uses *CURL*[14] to create a new twitter post.



iRefrigerator

Current temperature is 9.86C, Tue 3 Aug 2010 13:53 WEST

3 minutes ago via API

Figure 10 - Twitter Notification

3.4.1.3.6 Creating a plug-in

Create a plug-in is very easy to achieve. We now demonstrate the creation of the URL plug-in.

At start, we must include sawa_plugin.h header file, this header file includes the *GLib* and *GModule* include declarations, and the definitions of Policy and Action data structures.

```
#include <sawa_plugin.h>

G_MODULE_EXPORT void
run(gpointer data) {

        Action* act;
        Policy* pl;
        char** tokens;

        pl=(Policy*)data;
        act=pl->action;

        tokens = g_strsplit(act->cmd->str, "#", 0);
        system(g_strconcat("GET ", tokens[1], " > /dev/null", NULL));
}
```

[12] http://twitter.com
[13] http://apiwiki.twitter.com/Authentication
[14] http://curl.haxx.se

The Policy data structure contains a name, status, id, the execution interval and an Action structure. The Action structure contains the command, the type and the date of last execution.

Since we are delegating the execution of the action on the plug-in, we do not want to receive any result. The plug-in “run” function argument is a *gpointer*, a *gpointer* is an untyped pointer, which can be used to pass any data type. The actionchecker process passes a Policy structure pointer to the “run” function in plug-in, which contains all information needed for the executing of the notification.

The command string is concatenated with cardinal characters; the function *g_strsplit* is used to split in multiple tokens. When the split is complete, any action can be executed with the available information, from a simple print, to the most complex operations.

At the end of code, is used the “*system”* function, which runs the given string as a system call, in this example is a simple HTTP GET to an URL.

After compiling the plug-in and moving it to the plug-ins folder, the actionchecker can load this new plug-in, if is set to load in main configuration file.

3.4.1.4 Scheduler

Scheduler is a helper service, responsible for managing new recordings, new policies and their status. Before explaining how the process works, a brief description of policies and recording status is necessary to a better understanding.

A policy has four types of status, active, pending, removing and inactive.

Active – the policy is active and is recognized by actionchecker process

Pending – the policy is a new policy and waits to be activated by scheduler process.

Removing – the policy is ignored by actionchecker process and waits to be deleted by scheduler process.

Inactive – the policy is suspended and only the owner can activate or remove.

A recording has three types of status, running, starting and stopping.

Running – the recording is activated and the process stores sensor data.

Starting – the recording is waiting to be activated by scheduler process.

Stopping – the recording waits to be stopped.

The scheduler service acts as intermediary between the Web application and the other two services. The policies and recording requests made by users in the web application and sent to the database where they wait to be resolved.

In Figure 11 the sequence diagram represents the interactions between scheduler and other participants, when a new policy is created. When a policy is created by one user, the policy is stored in database with pending status. The scheduler service checks periodically for new policies, when a new policy is detected, the scheduler process retrieves the networks nodes which are not being subscribed by actionchecker process, and then sends a subscribe order to actionchecker process. Finally the scheduler changes the policy status to active.

To remove or disable a policy, the process is almost identical as explained above, but instead subscribing orders, the scheduler process sends a unsubscribe order with unused network nodes.

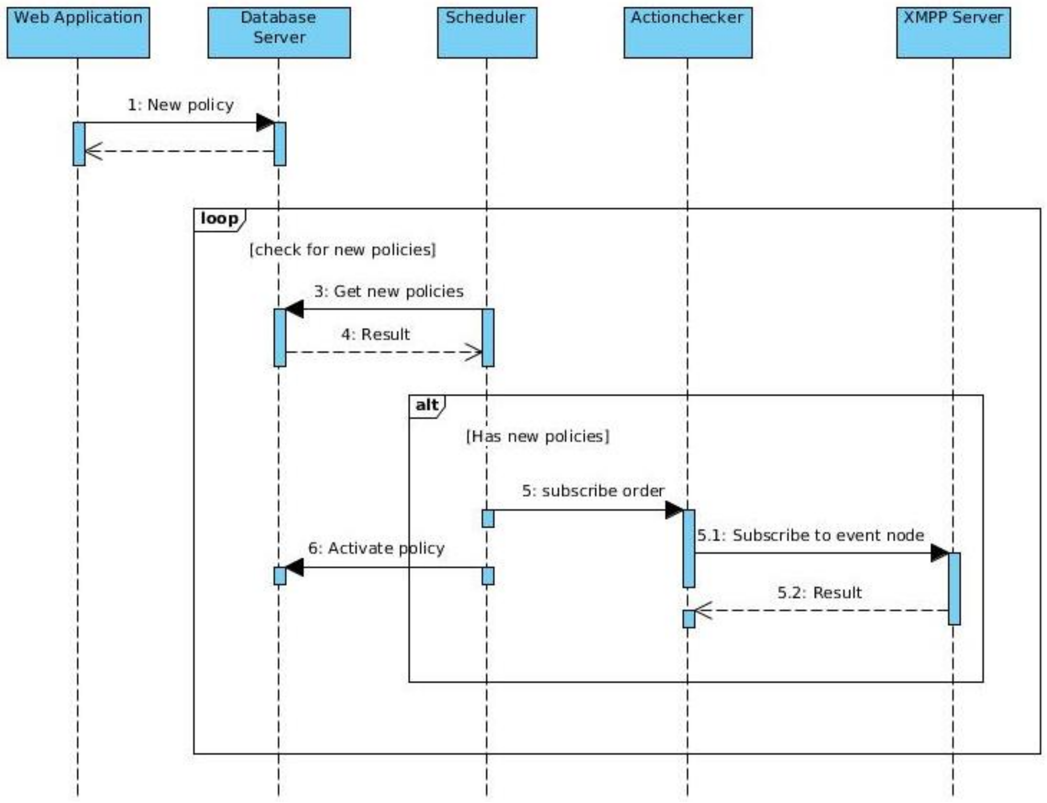


Figure 11 - New Policy

Besides new policies, by default all sensors have the recording enabled, being possible to stop and resume. The scheduler process handles the resume of recordings, and new recording, by sending a subscribe order to datarecorder socket, in the same way as actionchecker process.

3.4.2 Web Application

The web application was built with the objective to enable users to manage their sensors and sensor networks. The application was built with CodeIgniter[15] PHP Framework, which uses the Model-View-Controller (MVC) development pattern. With the use of a PHP Framework, a fast and easier development was achieved.

The management of a sensor network in web application can be done using the most basic operations, create and remove. In a sensor network it is possible to have an unlimited number of sensors and actuators. Also in sensors is possible to have an unlimited number of sensor measured parameters, which we call “variables”. In actuators is possible to add actuator commands and functions, and is possible to call these commands from the web application.

Besides the ability to manage sensor and networks, in web application is possible to record sensor data and share our sensors with other users. Users may share their network devices, and to do that, the owner must add members to the network group. A user which is a member of a network is allowed to access sensor data from that network, and is able to create policies with networks sensors, as well export data and generate charts.

Sensor data can be represented in charts. We included two chart components called Open Flash Charts 2 and Dygraphs, enabling interactive and dynamic charts. We have four types of charts, the basic line chart, bar charts, pie charts and radar charts. The representation of sensor data starts with the selection of a chart type, the selection of sensor variables and finally the time interval (day, month, etc).

Policies are an important feature of our system; they can be used to alert persons or systems. The web application allows the creation of policies with a combination of different rules, and the respective action.

[15] http://codeigniter.com

Given the brief description of the web application functionalities, in this section we present simple and illustrative examples of application functionalities.

3.4.2.1 Main Interface

The main interface consists in a modified *Openfire*[16] Cascade Style Stylesheet (CSS), in the construction of the web application we faced the problem of a growing number of features, being impossible to fit every option in a single menu. The interface contains a menu, similar to a tree, starting from generic root options, and then subdividing into multiple sections. This menu allows us to expand the platform features without having to constantly change the menu or interface layout.

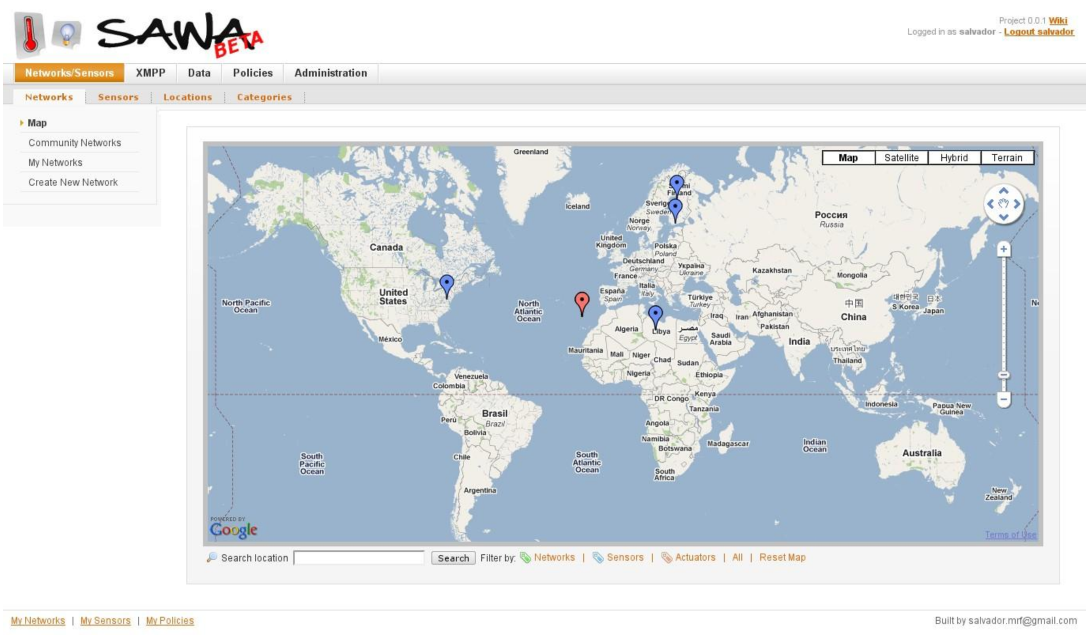


Figure 12 – Main Interface

[16] http://www.igniterealtime.org/projects/openfire/

3.4.2.2 Browsing Sensors and networks

3.4.2.2.1 Map

In the web interface is possible to locate geographically the devices. The Map panel allows the selection of sensors, actuators and networks. Another aspect is support for area search and connection between networks and their devices.

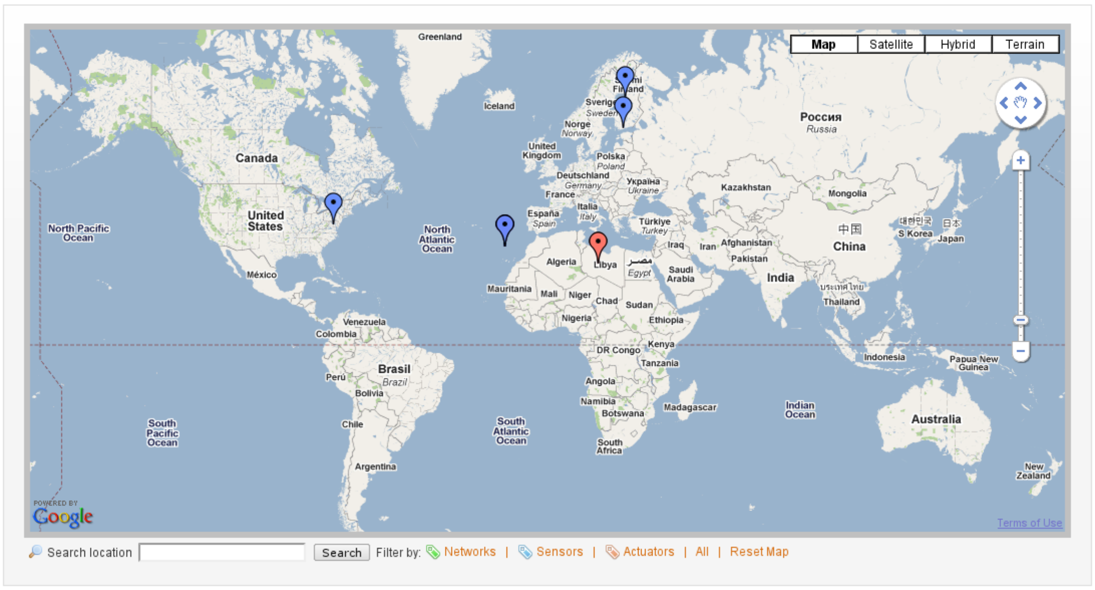


Figure 13 - Devices Map

The web interfaces uses Google Maps API, this web mapping service contains geographical data (places, images, streets) from a numerous set of countries.

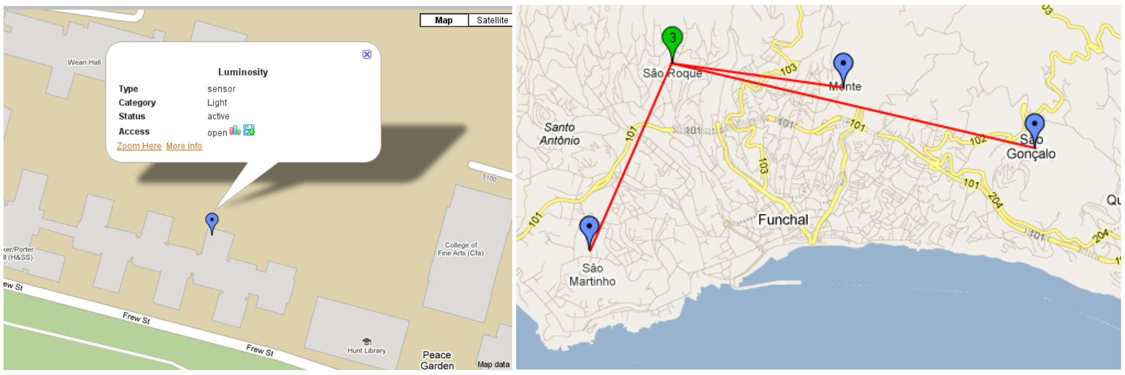
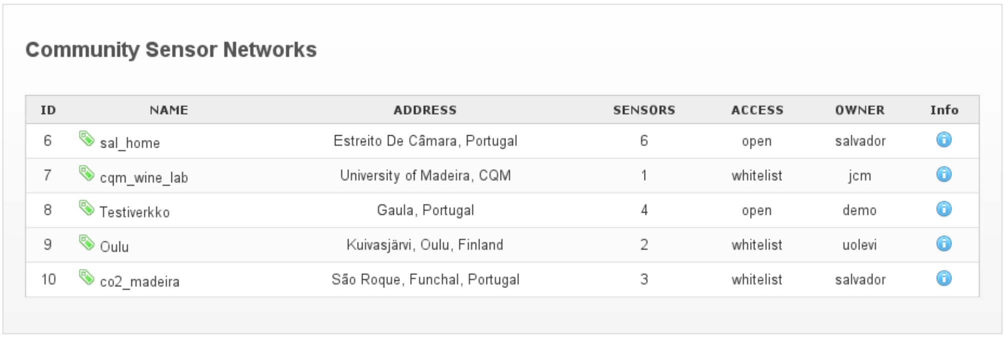


Figure 14 - Sensor details and Network connections

3.4.2.2.2 Networks & Network information

User networks and community networks are listed in two different pages, in community page, it is possible to browse all network and network details. The user networks page contains a set of administration options (remove, manage gallery, etc).

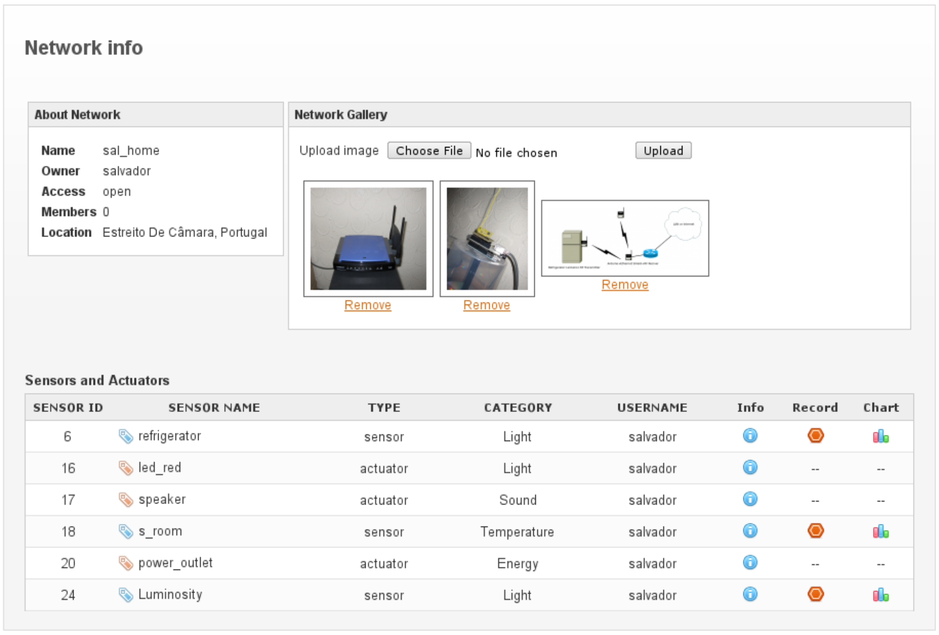


Community Sensor Networks

ID	NAME	ADDRESS	SENSORS	ACCESS	OWNER	Info
6	sal_home	Estreito De Câmara, Portugal	6	open	salvador	ⓘ
7	cqm_wine_lab	University of Madeira, CQM	1	whitelist	jcm	ⓘ
8	Testiverkko	Gaula, Portugal	4	open	demo	ⓘ
9	Oulu	Kuivasjärvi, Oulu, Finland	2	whitelist	uolevi	ⓘ
10	co2_madeira	São Roque, Funchal, Portugal	3	whitelist	salvador	ⓘ

Figure 15 - Community Networks

The Figure 16 shows the detailed information about “sal_home” sensor network. Sensor and actuators are listed and is possible to a set of options (view more info, record, view charts). The page also contains an image gallery, allowing users to add and show images from their resources.



Network info

About Network

Name	sal_home
Owner	salvador
Access	open
Members	0
Location	Estreito De Câmara, Portugal

Network Gallery

Upload image [Choose File] No file chosen [Upload]

Remove Remove Remove

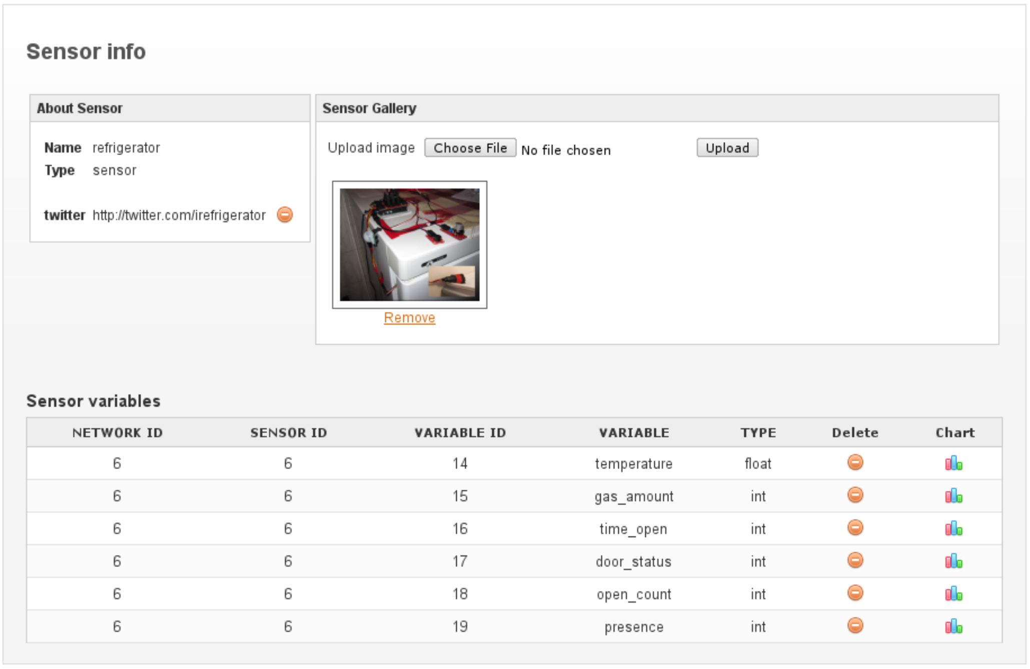
Sensors and Actuators

SENSOR ID	SENSOR NAME	TYPE	CATEGORY	USERNAME	Info	Record	Chart
6	refrigerator	sensor	Light	salvador	ⓘ	●	▮
16	led_red	actuator	Light	salvador	ⓘ	--	--
17	speaker	actuator	Sound	salvador	ⓘ	--	--
18	s_room	sensor	Temperature	salvador	ⓘ	●	▮
20	power_outlet	actuator	Energy	salvador	ⓘ	--	--
24	Luminosity	sensor	Light	salvador	ⓘ	●	▮

Figure 16 - Network details

3.4.2.2.3 Device Details

The page "sensor info" shows detailed information about a specific sensor. This detailed information contains the basic sensor information, an image gallery and the sensor variables list.



Sensor info

About Sensor

Name refrigerator
Type sensor

twitter http://twitter.com/irefrigerator

Sensor Gallery

Upload image Choose File No file chosen Upload

Remove

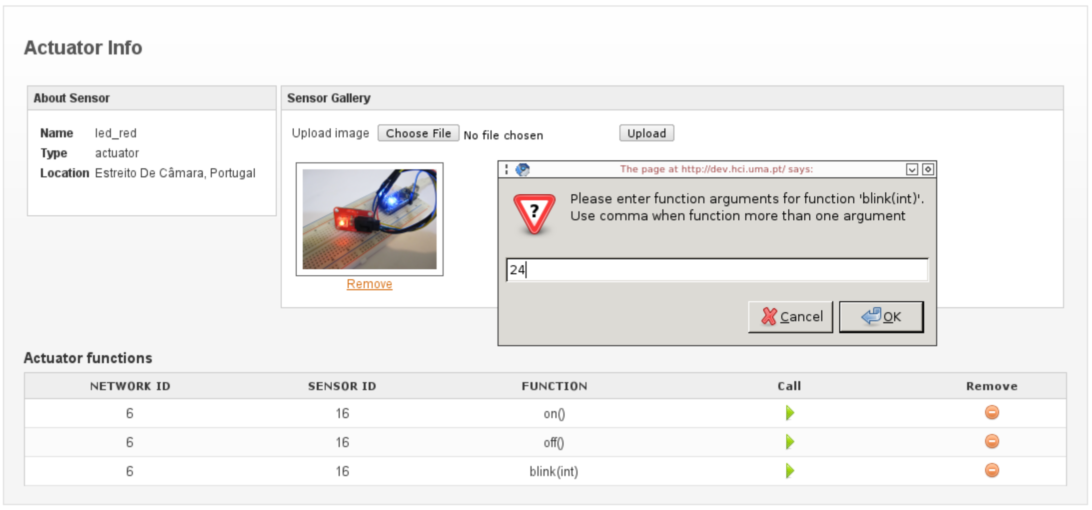
Sensor variables

NETWORK ID	SENSOR ID	VARIABLE ID	VARIABLE	TYPE	Delete	Chart
6	6	14	temperature	float	⊖	📊
6	6	15	gas_amount	int	⊖	📊
6	6	16	time_open	int	⊖	📊
6	6	17	door_status	int	⊖	📊
6	6	18	open_count	int	⊖	📊
6	6	19	presence	int	⊖	📊

Figure 17 - Sensor details

The web application provides the possibility of calling the defined actuator functions and commands. When a function has arguments, a javascript dialog input box is created and waits for user input. When the user confirms, the system publishes the command with arguments to the XMPP node. The actuator gateway, which subscribes the network node, parses the actuators commands, and call's the actuators using the local network protocols (e.g. RF, Bluetooth).

In Figure 18, the "actuator info" page displays the actuator information along with the actuator commands and functions. The image exemplifies the calling of an actuator function, as describe in previous use case.



Actuator Info

About Sensor

Name	led_red
Type	actuator
Location	Estreito De Câmara, Portugal

Sensor Gallery

Upload image Choose File No file chosen Upload

Remove

The page at http://dev.hci.uma.pt/ says:

Please enter function arguments for function 'blink(int)'.
Use comma when function more than one argument

24

Cancel OK

Actuator functions

NETWORK ID	SENSOR ID	FUNCTION	Call	Remove
6	16	on()	▶	⊖
6	16	off()	▶	⊖
6	16	blink(int)	▶	⊖

Figure 18 - Calling Actuator Function

3.4.2.3 Management

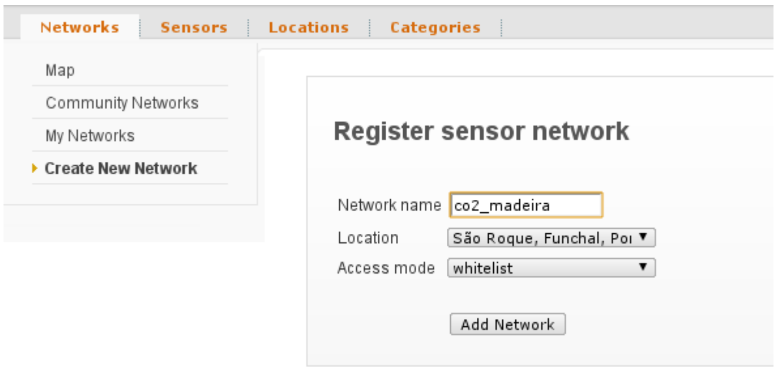
3.4.2.3.1 Add Network

To create a network, three steps are needed: the specification of network name, the access mode, and the location where the network will be located. There is also three types of access modes, whitelist, open and authorize.

Open – any user of the system can access the sensor data, create policies with network sensors, and can subscribe to published data

Whitelist – in this access mode, the network has a list of members with access to the network resources. The whitelist is managed by owner, he can add or remove members.

Authorize – an access mode based on requests, when a user wants to access a network resources, an authorization request has to be made. The owner of the network may allow or deny. This access mode was not implemented, but is an important feature to be implemented in future.



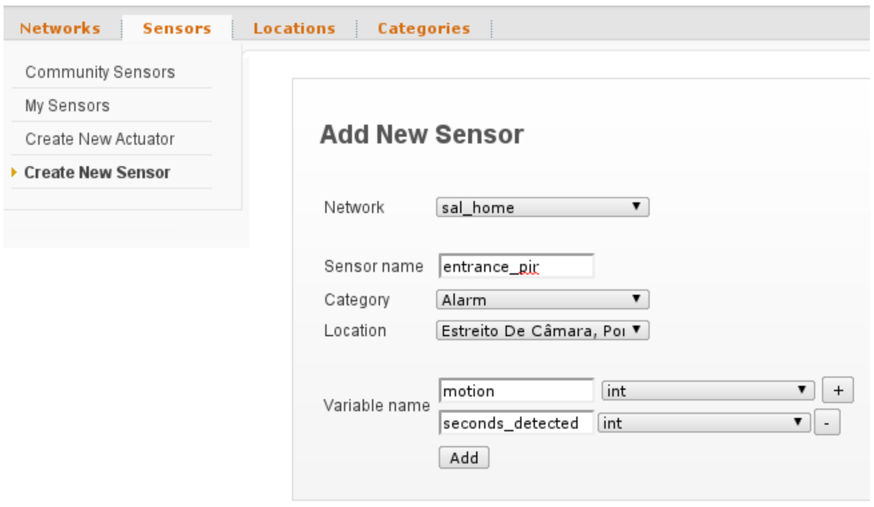
Register sensor network

Network name: co2_madeira
Location: São Roque, Funchal, Poı
Access mode: whitelist

Figure 19 - Adding a new Sensor Network

3.4.2.3.2 Add Sensor

To add a new sensor, the network, category and location are specified. A sensor can sense more than one physical parameter: e.g. accelerometers can report X, Y and Z axis values, as well pre-processed and derived data. We call variable as a sensor measurement either in raw, processed or derived. In Figure 20, we add a presence sensor, which senses motion in a room. For this sensor we add two sensor variables, the “motion”, meaning “0” for no motion and “1” for motion. And the variable “seconds_detected”, that represents the time in seconds which the presence sensor detected the motion continuously.



Add New Sensor

Network: sal_home

Sensor name: entrance_pir

Category: Alarm

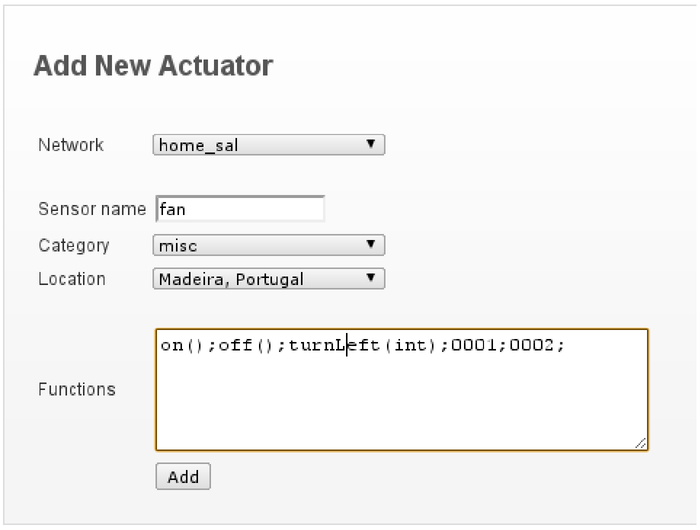
Location: Estreito De Câmara, Poi

Variable name: motion int; seconds_detected int

Figure 20 - Adding a new Sensor

3.4.2.3.3 Add Actuator

Similar to adding a sensor is adding a new actuator, like sensors, the network name, sensor name, category and location fields are specified. The difference stays in commands, these defined commands or functions enables us to change the behavior of actuators. The actuator commands and functions are added using semicolons to separate them, and a function with arguments contains brackets with arguments types separated by commas. As example "turnOn(int,float);".


Add New Actuator

Network	home_sal
Sensor name	fan
Category	misc
Location	Madeira, Portugal
Functions	on();off();turnLeft(int);0001;0002;

Figure 21 - Adding a new Actuator

3.4.2.4 Policies

We introduce policies with a description about the policy associated concepts and we exemplify the use of a Policy for gas leaks.

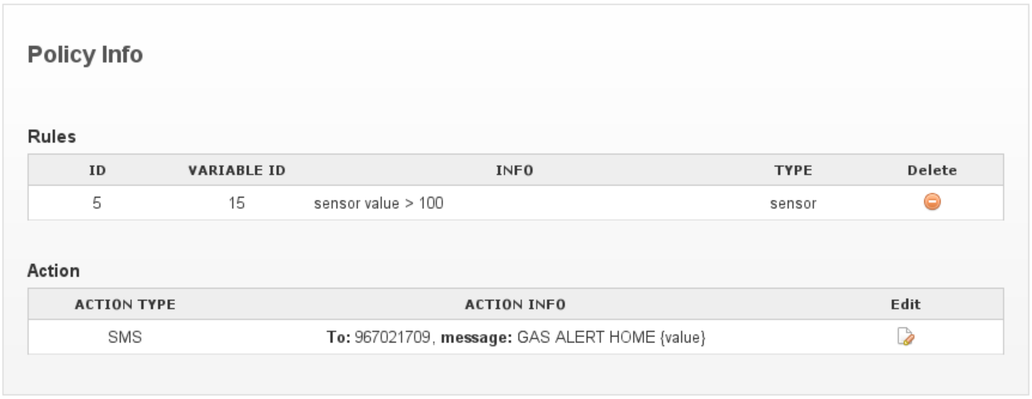
Notification interval - is the time in seconds, the system should execute the repeatedly the same action when a policy matches. This avoids the continuous notifications when a policy is constantly matched (We do not want to receive a SMS notification every second).

Notification type - is the type of notification, which can be SMS, URL, or other. Each notification has a different form, and is possible to add tags ("{value}", "{date}") in messages. These tags are replaced by the real values and sent along with the message.

Rule – is a part of policy and is the definition of specific conditions, either expected or not expected. A rule is associated to a sensor variable and dependently of type, may contain conditions and/or intervals.

3.4.2.4.1 Add Policy

In Figure 22, we created a policy named "gas_alert", the selected notification type is SMS, and in the message field we used the tag ({value}) to be replaced on SMS message. This policy contains only one rule, the defined rule uses gas_value variable, the operator '>' and the match value 100. The refrigerator use case has a gas sensor which detect the amount of gas in the air, we use the gas sensor to detect butane gas leaks, resuming, this policy alerts via SMS when there is a gas leak (gas_value > 100).


Policy Info

Rules

ID	VARIABLE ID	INFO	TYPE	Delete
5	15	sensor value > 100	sensor	⊖

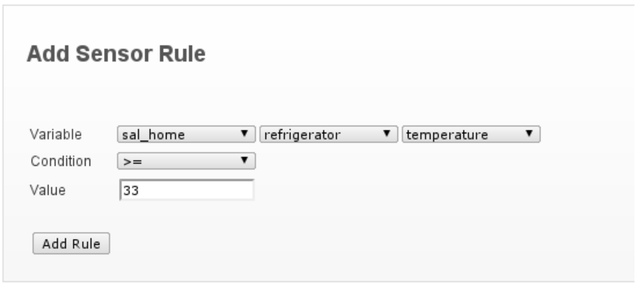
Action

ACTION TYPE	ACTION INFO	Edit
SMS	**To:** 967021709, **message:** GAS ALERT HOME {value}	✎

Figure 22 - Policy details

3.4.2.4.2 Add Rule

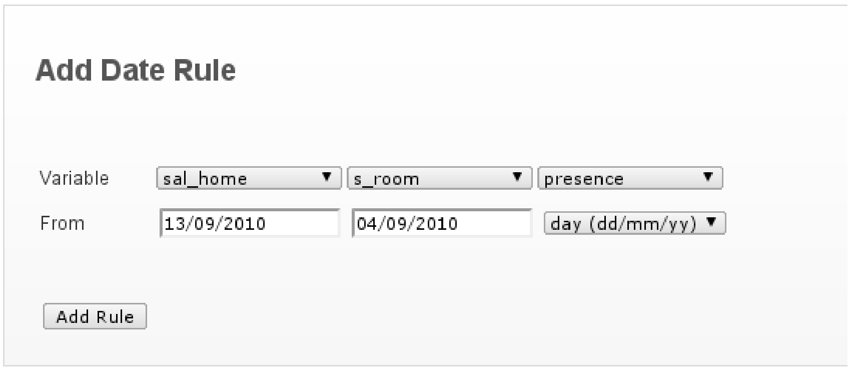
Three types of rules are supported by the system, the sensor rule accepts a sensor input and verifies if the input value matches a certain condition (e.g., >300). In numeric variables we can use the operators *“<”*, *“<=”*, *“=”*, *“>=”* and “>”. And for variables of type string, *“is”*, *“is not”*, *“starts with”*, *“ends with”*, *“contains”* and *“does not contain”* operators can be used.


Add Sensor Rule

Variable	sal_home ▼	refrigerator ▼	temperature ▼
Condition	>= ▼		
Value	33		

Figure 23 - Sensor Rule

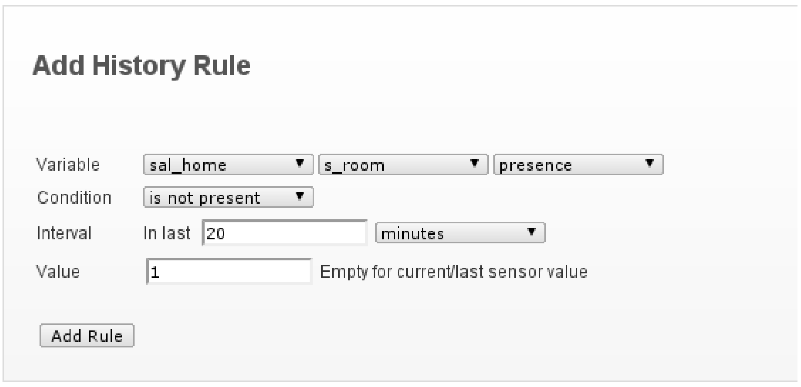
The date rule, allows us to check if a sensor input is between a specified time interval (day or hour). As example, we may set a policy to notify when presence is detected in our office between 2am and 5 am.


Add Date Rule

Variable	sal_home ▼	s_room ▼	presence ▼
From	13/09/2010	04/09/2010	day (dd/mm/yy) ▼

Figure 24 - Date Rule

History rules are similar to simple SQL queries, it allows to define a sensor input, define a condition (“is present”, “is not present”), an interval (minutes or records) and a matching value. As shown in Figure 25, the history rule will verify if *presence* (“denoted as 1”) is not present in the last 20 minutes. With this simple rule, is possible to turn off lights when human presence is not detected in a room.

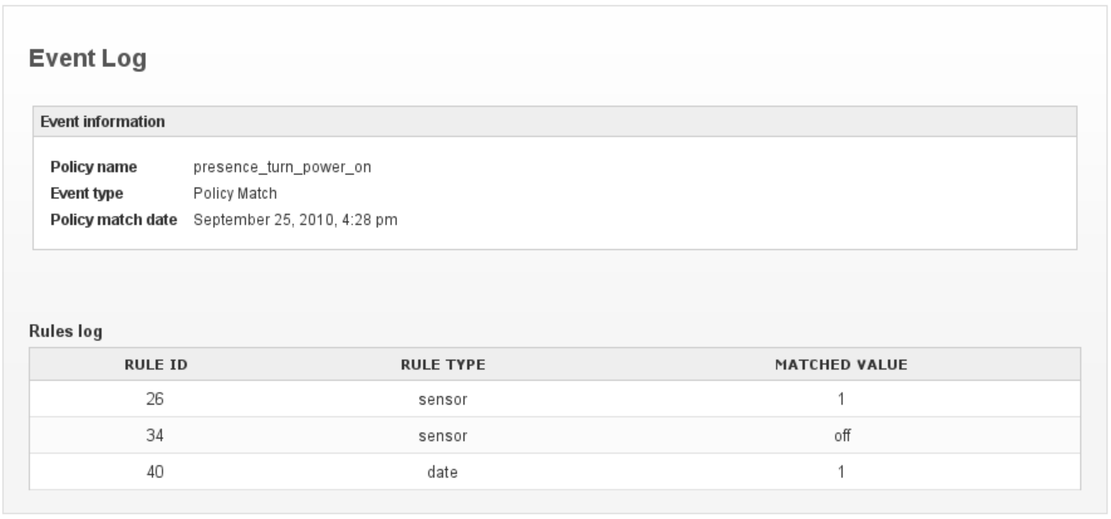

Add History Rule

Variable	sal_home ▼ s_room ▼ presence ▼
Condition	is not present ▼
Interval	In last 20 minutes ▼
Value	1 Empty for current/last sensor value

Figure 25 - History Rule

3.4.2.4.3 Event log

When a policy matches, the system creates a log record called event, where its stores the date and sensor values at matching time.


Event Log

Event information

Policy name	presence_turn_power_on
Event type	Policy Match
Policy match date	September 25, 2010, 4:28 pm

Rules log

RULE ID	RULE TYPE	MATCHED VALUE
26	sensor	1
34	sensor	off
40	date	1

Figure 26 - Event Log

3.4.2.5 Data & Charts

The web application provides time based charts, with four types of charts, line, bar, pie and radar charts. In every chart, is possible to add sensor variables to compare. As shown in Figure 27, we are representing the gas amount, temperature and human presence. For a better user interaction, we added a configuration panel, which provides basic chart options. Saving the chart as image, selecting the time interval of the chart (minutes, hour, day, week, month, year or years), the Y-axis minimum and maximum size, and offset to change the period of time, are some options of the chart panel.

Another important aspect is how we represent sensor data. We use two methods, aggregation and raw mode.

Aggregate chart – the sensor data is grouped by interval and an average is represented.

Raw chart – All data points are represented.

One problem that arose when using charts to represent many sensor data points, was the rendering of thousands of data points. The browser takes time to download the sensor data; the flash plug-in allocates a larger amount of memory to render the data; the rendering takes more time and the user interaction with charts was slow.

To reduce this problem, we used aggregate data and a faster chart component to represent thousands of points. Aggregate charts allows grouping by an interval of interest, and by using average, the number of records are reduced drastically. With fewer records, the data transmission, rendering and chart interaction are faster. And for representing data in raw mode, we reduced the time interval which users can select to display sensor data.

	Aggregation Charts	**Raw Charts**
Advantages	Less data points to represent Faster to transmit and render Less memory consumed Faster chart interaction	True representation of data Ideal for detecting anomalies
Disadvantages	Difficulty to detect anomalies	Many data points More time to transmit and render Consumes more memory Slow chart interaction

Table 7 - Data Representation Comparison

3.4.2.5.1 Chart types

The system provides a charting component, the chart pages contains the chart area, a chart configuration panel and a sensor variable list.

Configuration panel:

Save chart – Enables the users to export and download the chart as image.

Interval – Is the desired time interval to be represented in chart. As example, if one month is selected “1M”, the chart will show all days from that month.

Area fill – If checked, the chart will fill the area. When disabled, the chart only represent the lines.

Auto-Y – When enabled, the chart finds the best Y-axis range to fit all data inside of chart area. When disabled the user can specify the Y-axis range (minimum and maximum).

Auto-reload – The auto reload option enables to chart to reload itself after a minute.

The variable panel contains the variables from the sensor we have selected to generate the chart. This panel allows to choose what sensor variables to represent, and is possible to compare with variables from other sensors.

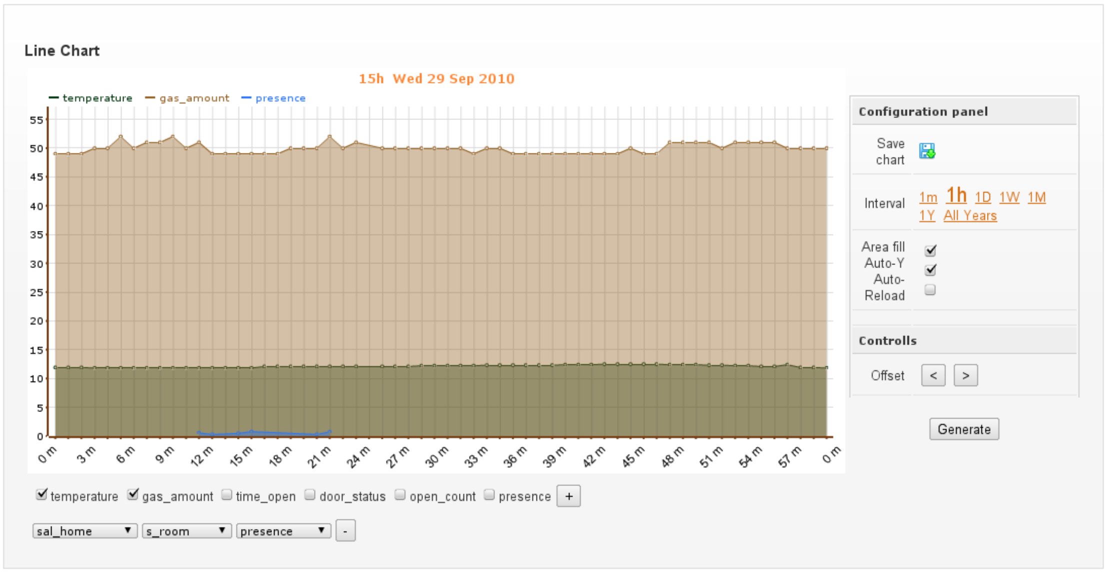


Figure 27 - Line Chart

In Figure 27 is represented a line chart, which represents sensor data over the time. This type of charts allows simple comparisons between sensor variables, as well as a browser to the sensor data history.

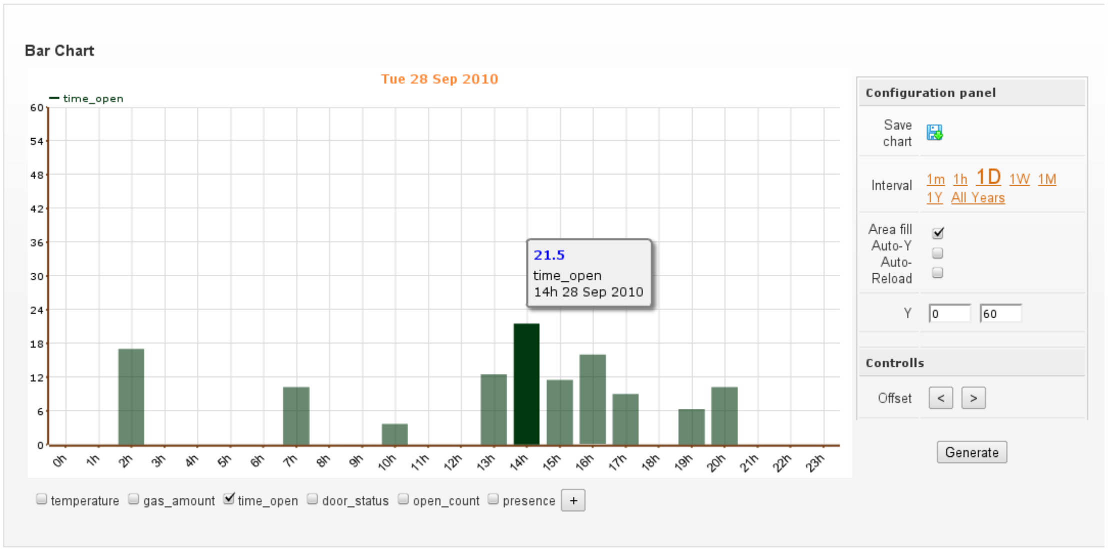


Figure 28 - Bar Chart

The Figure 28 represents the average time open (in seconds) of the refrigerator door from a particular day. The graphs show hours with no data, meaning that refrigerator door was not open in that hour. With this simple chart, we can make conclusions about many aspects (time open, particular hours, difference between hours, etc).

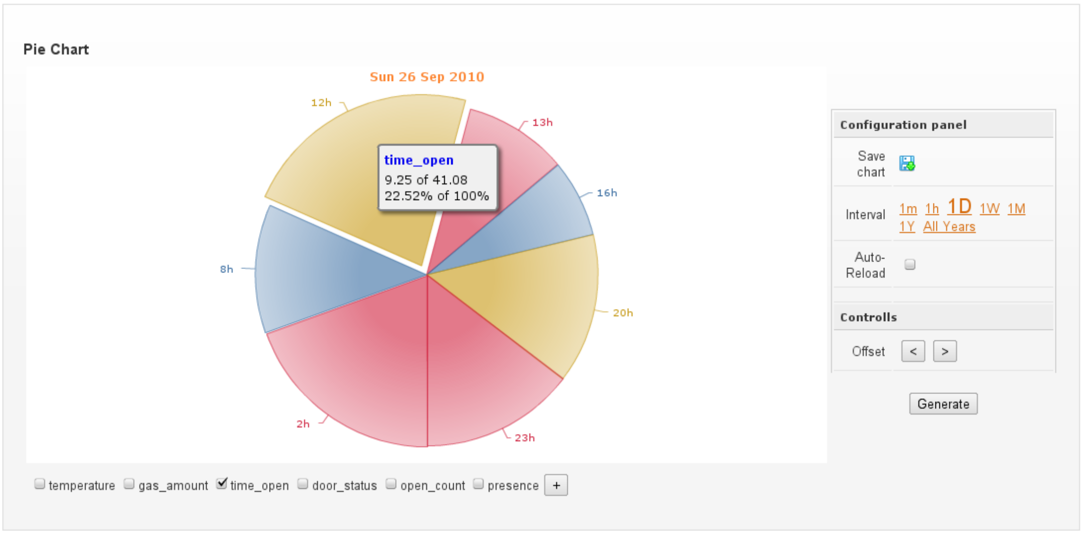


Figure 29 - Pie Chart

Pie charts, represents the proportion of each slice, in relation to the total quantity represented. The pie chart in Figure 29 represents the proportion between hours of the refrigerator door open time (average in seconds).

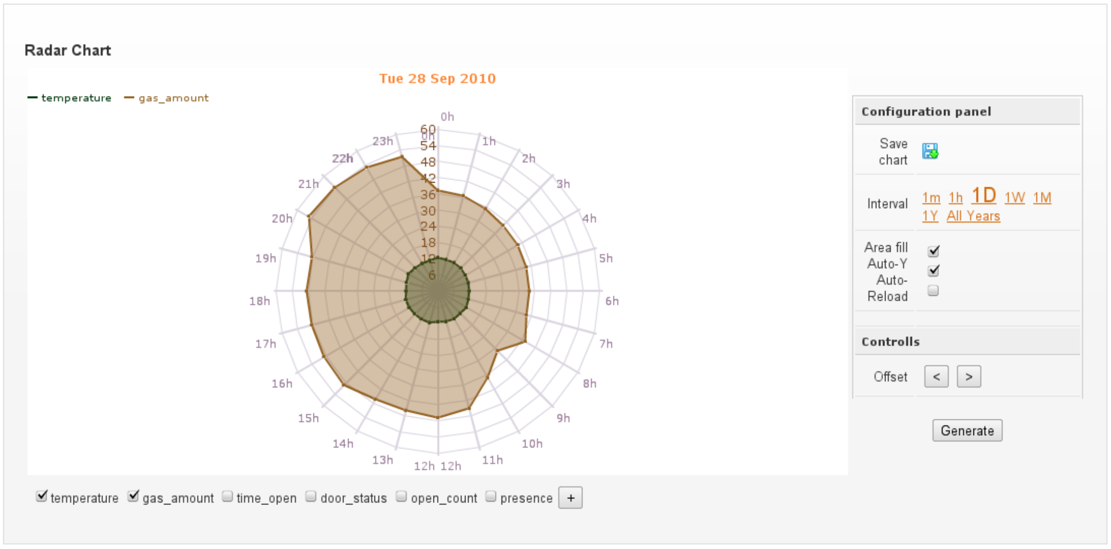

Radar Chart
Tue 28 Sep 2010
temperature
gas_amount
0h 1h 2h 3h 4h 5h 6h 7h 8h 9h 10h 11h 12h 13h 14h 15h 16h 17h 18h 19h 20h 21h 22h 23h
60 54 48 42 36 30 24 18 12 6
Configuration panel
Save chart
Interval 1m 1h 1D 1W 1M 1Y All Years
Area fill
Auto-Y
Auto-Reload
Controlls
Offset < >
Generate
temperature gas_amount time_open door_status open_count presence +

Figure 30 - Radar chart

Radar charts display the desired variables using the equivalent polar coordinates. The x-axis values are mapped clockwise, and y-axis values are indicated by their distance from the center of the chart.

In Figure 31 is represented an overlay of two charts, the darkest green line represents an aggregate chart line, from refrigerator temperature; the other line represents all data points from the same day. The aggregate line behaves as a trend line, and hides important data, such as maximum, minimum and other important aspects.

We can notice the frequency of work in refrigerator without noise, between 3 am and almost 8 am, when there is less human activity on refrigerator. The graph shows the refrigerator temperature as expected, the temperature starts to increase after the cooling process finishes. When the temperature increases to a value, the process of cooling starts again. By the chart, in average, it takes 44 minutes to temperature increase and half of these 44 minutes to cool down.

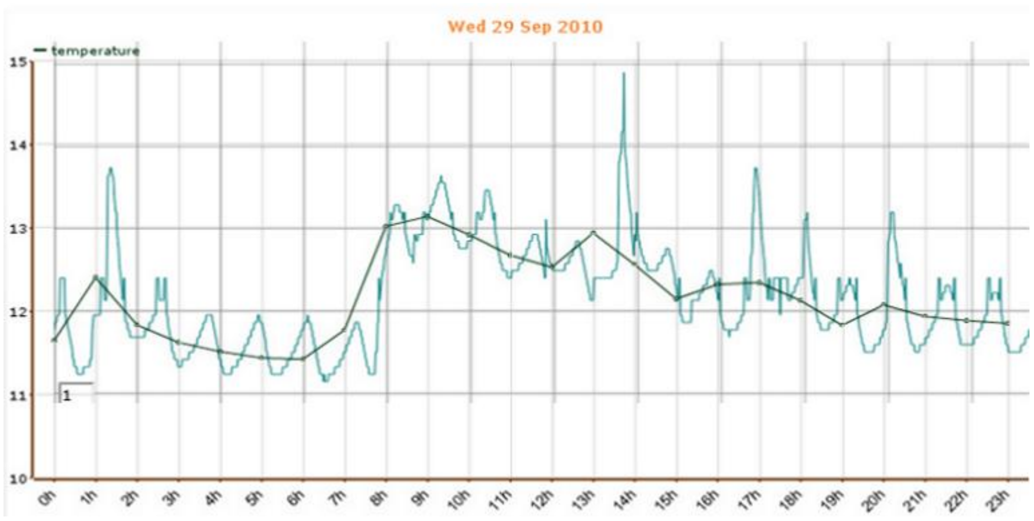


Figure 31 – Aggregate and Raw data

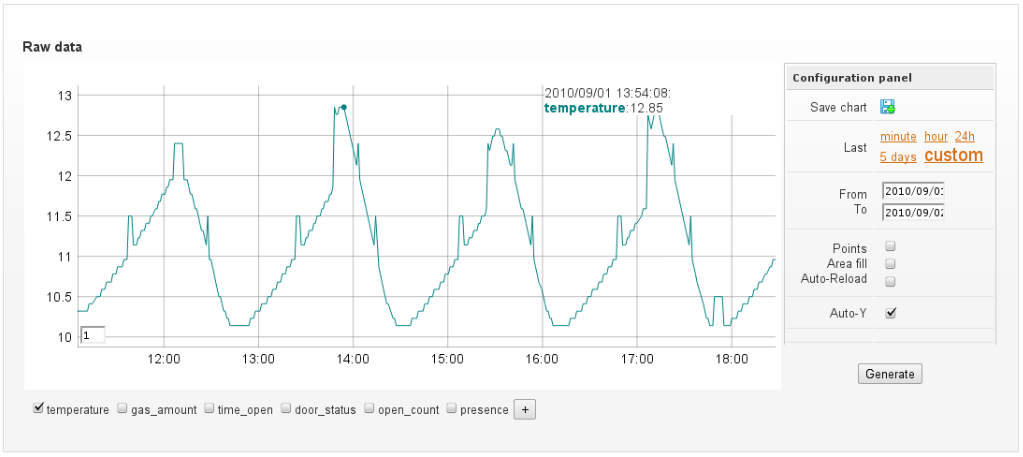
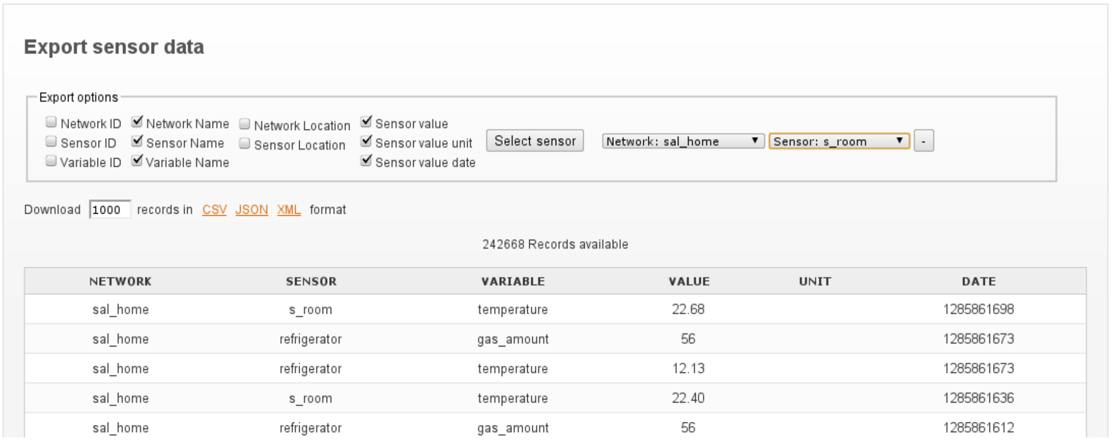


Figure 32 – Zooming in Raw Data

3.4.2.5.2 Export Data

In Figure 33 is shown the export panel. The export panel allows the user to select a sensor from a network, select the export fields such as id's and names. The users can select the number of records to export and a file format.



Export sensor data

Export options

☐ Network ID [✓ Network Name] ☐ Network Location [✓ Sensor value]
☐ Sensor ID [✓ Sensor Name] ☐ Sensor Location [✓ Sensor value unit]
☐ Variable ID [✓ Variable Name] [✓ Sensor value date]

Select sensor | Network: sal_home | Sensor: s_room | -

Download 1000 records in CSV JSON XML format

242668 Records available

NETWORK	SENSOR	VARIABLE	VALUE	UNIT	DATE
sal_home	s_room	temperature	22.68		1285861698
sal_home	refrigerator	gas_amount	56		1285861673
sal_home	refrigerator	temperature	12.13		1285861673
sal_home	s_room	temperature	22.40		1285861636
sal_home	refrigerator	gas_amount	56		1285861612

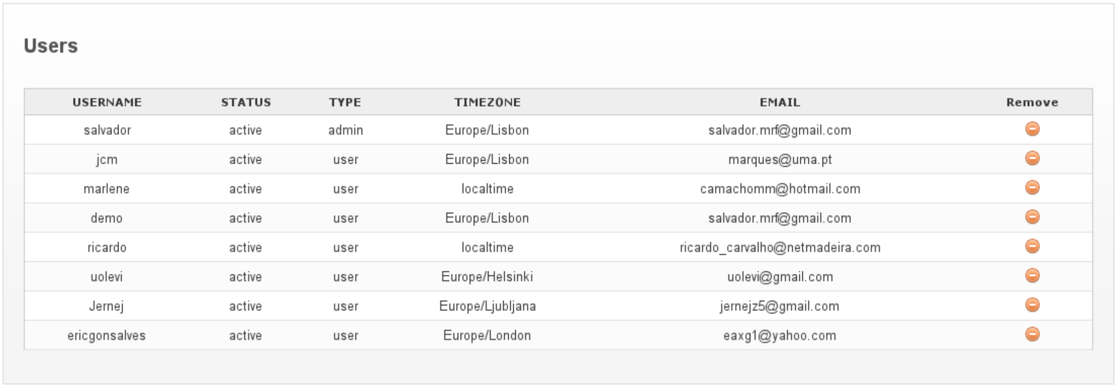
Figure 33 - Export Panel

3.4.2.6 Administration

The platform administrators have a set of monitoring pages, this pages enables the administrators to track users, services status and platform configuration.

3.4.2.6.1 Users

The user list contains all users in the platform, their details and an option to delete the user.



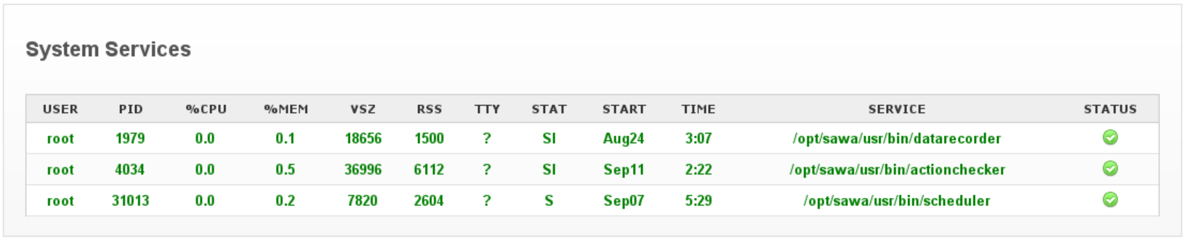
Users

USERNAME	STATUS	TYPE	TIMEZONE	EMAIL	Remove
salvador	active	admin	Europe/Lisbon	salvador.mrf@gmail.com	⊖
jcm	active	user	Europe/Lisbon	marques@uma.pt	⊖
marlene	active	user	localtime	camachomm@hotmail.com	⊖
demo	active	user	Europe/Lisbon	salvador.mrf@gmail.com	⊖
ricardo	active	user	localtime	ricardo_carvalho@netmadeira.com	⊖
uolevi	active	user	Europe/Helsinki	uolevi@gmail.com	⊖
Jernej	active	user	Europe/Ljubljana	jernejz5@gmail.com	⊖
ericgonsalves	active	user	Europe/London	eaxg1@yahoo.com	⊖

Figure 34 - Platform Users

3.4.2.6.2 Services

The services page, allows administrators to monitor the platform services, the list in Figure 35 contains the process status for each service (status, memory consumption, cpu usage, etc).



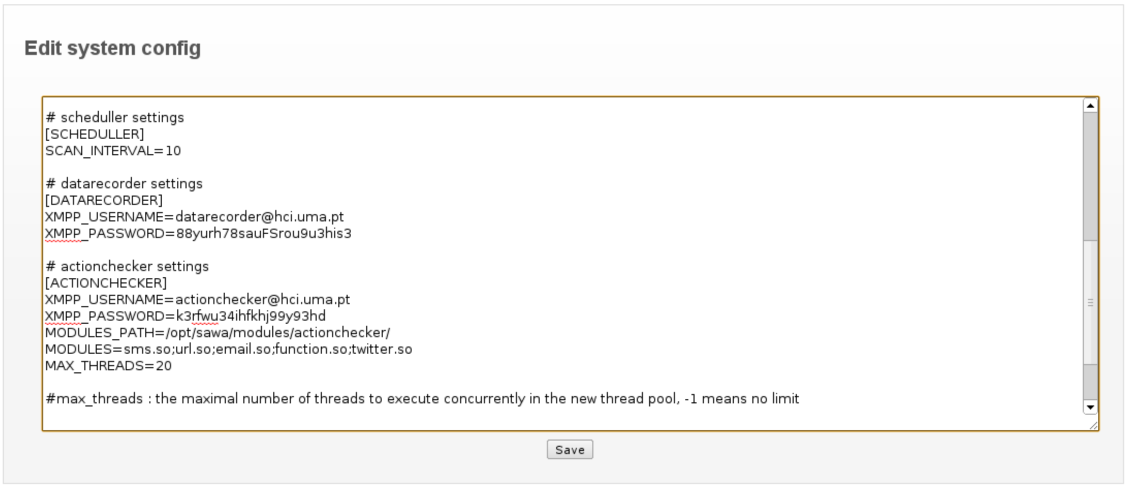
System Services

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	SERVICE	STATUS
root	1979	0.0	0.1	18656	1500	?	Sl	Aug24	3:07	/opt/sawa/usr/bin/datarecorder	✓
root	4034	0.0	0.5	36996	6112	?	Sl	Sep11	2:22	/opt/sawa/usr/bin/actionchecker	✓
root	31013	0.0	0.2	7820	2604	?	S	Sep07	5:29	/opt/sawa/usr/bin/scheduler	✓

Figure 35 - Services Status

3.4.2.6.3 Edit Configuration

The platform configuration file can be edited online by administrators. The configuration file contains all settings and credentials for services to be able to connect to XMPP server and database server.



Edit system config

```
# scheduller settings
[SCHEDULLER]
SCAN_INTERVAL=10

# datarecorder settings
[DATARECORDER]
XMPP_USERNAME=datarecorder@hci.uma.pt
XMPP_PASSWORD=88yurh78sauFSrou9u3his3

# actionchecker settings
[ACTIONCHECKER]
XMPP_USERNAME=actionchecker@hci.uma.pt
XMPP_PASSWORD=k3rfwu34ihfkhj99y93hd
MODULES_PATH=/opt/sawa/modules/actionchecker/
MODULES=sms.so;url.so;email.so;function.so;twitter.so
MAX_THREADS=20

#max_threads : the maximal number of threads to execute concurrently in the new thread pool, -1 means no limit
```

Figure 36 - Platform Configuration

3.4.3 Tools

Sensor Andrew project has developed a set of command line tools, each tool is responsible for execute an operation. They were useful for managing nodes from command line, and useful to be used by other applications. We have improved and integrated these command line tools into a single tool, which provides the most important functions and contains a configuration file to avoid the use of user credentials in command line.

sox_tools – Is a utility tool that uses SOX library and allows users to manage XMPP nodes, affiliations and subscriptions. Also allows publishing sensor data, listen, listen for actuator commands and publishing actuator commands.

sawa_tools - This tool is stripped version of sox_tools, and is used by the web application to retrieve data from XMPP and to make XMPP requests.

3.4.3.1 sox_tools

The first problem in using the Sensor Andrew command line tools was the need of specifying every time the user credentials and server settings, in order to execute each tool. When we developed new command line tools, such as "get_affiliations", which retrieves the affiliations of a user, most of the code was the same and all tools followed the same structure (parse user input, connect, execute operation and report result).

We developed *sox_tools*, a specific client to interact with our system. This tool joins all command line tools functions, and uses a configuration file, with server settings and user credentials.

```
./sox_tools
Available options:

        --user                      Select user from config

        listen                      Listens for pubsub events
        commands            Listens for commands and call appropriate action in actuator
        publish                     Publish to a pubsub node
        publish_cmd                 Publish a command to a pubsub node
        last_item                   Get last item from a pubsub node

        create                      Creates a pubsub node
        subscribe                   Subscribe to a pubsub node
        unsubscribe                 Unsubscribe from a pubsub node
        add_member                  Add a member to a pubsub node
        subscriptions               List pubsub subscriptions
        affiliations        Lists pubsub affiliations
        delete                      Deletes a pubsub node

        create_user                 Creates a XMPP user
        delete_user                 Deletes a XMPP user
```

The sox_tools configuration is in user home, normally in *POSIX* systems at "~/.sox_tools.cfg" path. Next, we explain each configuration option and the less common functions in sox_tools.

```
# this file in your home ~/.sox_tools.cfg

[XMPP]
PORT=10223
HOST=hci.uma.pt
PUBSUB_SERVER_NAME=pubsub.hci.uma.pt
TIMEOUT=10

[DEFAULT_USER]
JID=salvador@hci.uma.pt/laptop-user
PASSWORD=*********

[sapo]
JID=sapo@hci.uma.pt/soxtools
PASSWORD=********

[COMMANDS]
9.21.on()=echo -n "up#8#*" > /dev/ttyUSB0
9.21.off()=echo -n "down#8#*" > /dev/ttyUSB0
9.21.blink(int)=echo -n "up#8#{1}#*" > /dev/ttyUSB0
```

The configuration file contains four groups, the "XMPP", "DEFAULT_USER", "sapo" and "commands" group. The XMPP group contains the server settings and timeout value, which sets the number of seconds a request should be completed. If the request is not finished on time, the program will return a negative value.

We can specify more than one user in configuration file, allowing us to use the tool with different users, without changing the configuration file every time. When no user is specified (using *--user* option), the default user is assumed.

In sox_tools we have the "commands" option, being this option experimental and built for demonstration purposes. This option makes the tool listen for actuator commands and then execute the specified command from configuration file.

As example, supposing we are receiving the command "off()", to be called in actuator with id 21 from network 9. The tool will look for "9.21.off()" and execute what is in front of equal sign. Function arguments are also supported, in the third line of commands group, there is "{1}", this tag is replaced by the first argument.

3.4.3.2 sawa_tools

Before starting to develop the web application we looked for available PHP libraries to interact with XMPP server. We have found *XMPPHP*[17] and *JAXL*[18] projects, which allows interacting with XMPP servers, but they do not support XMPP Pub-Sub extension. However we found an *XMPPHP* project extension that supports Pub-Sub called "SIXTIES".

After testing and exploring *SIXTIES*[19] project, we have found a few problems, first the non support for XMPP administration extension (which supports user management) and the lack of documentation. If we choose to use the SIXTIES library, we would have to implement the Administration extension support and the Sensor Andrew sensor schema. For time constraints and because we were using the SOX library to develop tools/services, we selected to use the command line tools in web application instead of this PHP library.

At the beginning of the construction of web application, the sox command line tools were used to retrieve data and to make requests to XMPP server. In order to make a better output and easier parsing in web application, custom tools were developed. By the end, we had more than 20 command line tools, which included most of Sensor Andrew command line tools, our new created tools and adapted tools.

After the development of sox_tools, an adapted version was built to be used by the web application to replace all those command line tools. In the sawa_tool, the *listen* and *commands* options were stripped, the output was changed to be easier to parse, and server settings are read from the platform configuration file (/opt/sawa/etc/sawa.cfg). The sawa_tools, is no more than a replacement of an XMPP API, this tool is used by the web application to: create and remove users, create and remove nodes, retrieve user subscriptions/affiliations, unsubscribe, retrieve nodes, and publish actuator commands.

[17] http://code.google.com/p/xmpphp
[18] http://code.google.com/p/jaxl
[19] https://labo.clochix.net/projects/show/sixties

```
./sawa_tools
Available options:

        --user X --pass Y           Uses the specified user X with password Y

        publish                     Publish to a pubsub node
        publish_cmd                 Publish a command to a pubsub node
        last_item                   Get last item from a pubsub node

        create                      Creates a pubsub node
        subscribe                   Subscribe to a pubsub node
        unsubscribe                 Unsubscribe from a pubsub node
        add_member                  Add a member to a pubsub node
        subscriptions               List pubsub subscriptions
        affiliations                Lists pubsub affiliations
        delete                      Deletes a pubsub node

        create_user                 Creates a XMPP user
        delete_user                 Deletes a XMPP user
        nodes                       List all event nodes on server
```

Some servers may allow users to create an account by their own, other may not. We use an admin user to create and delete user accounts. In platform configuration file, a new group was added, the admin group, which contains an admin user in XMPP.

```
[ADMIN]
XMPP_USERNAME=sawa@genbox
XMPP_PASSWORD=***********
```

3.4.4 HTTP POST upload

For low resource devices, unable to use XMPP connections, we support publishing via HTTP. The following Bash script exemplifies how to publish using HTTP method. The server accepts 3 parameters, the sensor key (KEY), which grants publishing permissions; the sensor parameter id (VAR), which identifies the parameter of the sensor; and the parameter value (DATA).

```
#!/bin/bash

SERVER_URL='http://dev.hci.uma.pt/sawa/xmpp/publish'

KEY='5596dd8ee347131'
TEMP_VAR_ID=6
DATA=33

curl -X POST -d "KEY=$KEY&VAR=$TEMP_VAR_ID&DATA=$DATA" $SERVER_URL
```

The Figure 37 shows the sequence of publishing via HTTP, acting as proxy, and due to more one intermediary the time for publishing is higher. The web application when receives an HTTP POST request, checks if there is any sensor with the given key, when the sensor exists, the system retrieves sensor information from database (owner credentials, network and sensor details) and then uses the decrypted user credentials along with other information to publish via XMPP.

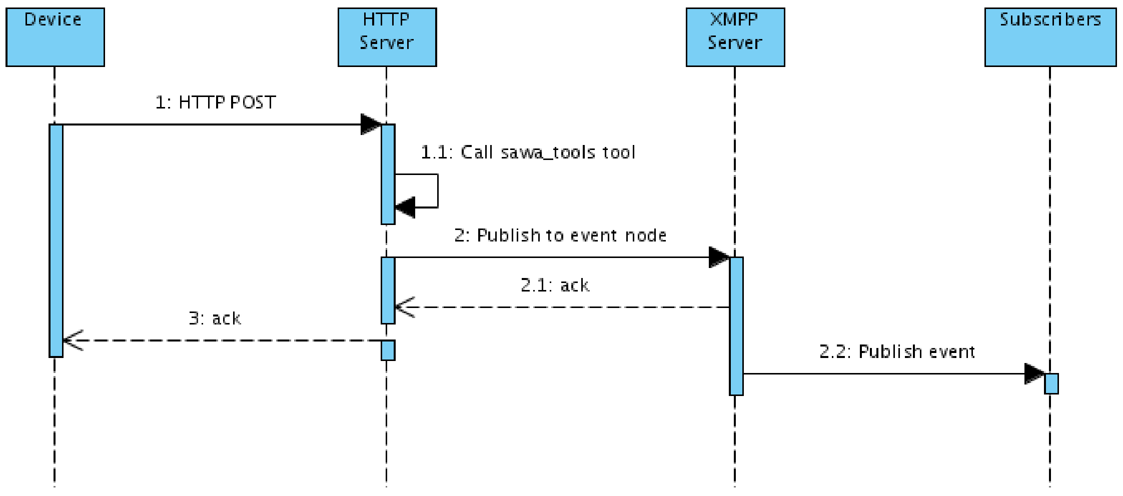


Figure 37 - HTTP POST Publishing

3.4.5 Platform

Instead of using system folders, the platform files reside in one folder "*/opt/sawa*". The main reason for this option was the difficulty to track all files and to manage the dispersed files. With this option, the platform folder can be easily removed, transported and replicated.

Path	File/Directory	Description
/opt/sawa/etc	sawa.cfg	Platform configuration file
/opt/sawa/modules	actionchecker/email.so actionchecker/function.so actionchecker/sms.so actionchecker/twitter.so actionchecker/url.so	Email plug-in XMPP publish event plug-in SMS plug-in Twitter plug-in URL plug-in
/opt/sawa/usr/bin	actionchecker datarecorder scheduler sox_tools sawa_tools	Policy matching service Data recording service Helper service Command line tool for interacting with system Utility tool, used by web application
/opt/sawa/usr/lib	libsox.so	SOX library
/opt/sawa/usr/include	soxlib.h sawa_plugin.h	SOX library header file actionchecker plug-in header file
/opt/sawa/web	non-www/application (1) non-www/system www/assets (1)/config/config.php (1)/config/email.php (1)/config/database.php	Web application files PHP Framework files All accessible files (JavaScript, images, CSS) Web application configuration Email connection settings Database connection settings

Table 8 - Platform File Structure

Currently the platform depends of path "*/opt/sawa*", but with some tweaks, we can use any path. The platform configuration file contains the database settings, xmpp connection settings and services settings, such as XMPP users and passwords.

The web folder, contains "*www*" and "*non-www*" folders, the "*www*" folder, contains all readable files like CSS, JavaScript, images, and others files. Inside "*non-www*" is two main folders, the "*system*" folder that is the PHP framework and contains all framework files. The "*application*" folder contains the developed web application with the PHP framework.

The *include* path, contains the development headers, to be able to develop application with SOX library, we included SOX library header (soxlib.h). For developing actionchecker plug-ins, the header file sawa_plugin.h is also needed.

3.4.5.1 Database

The database is composed by 31 tables, 7 of them virtual (database views). The database includes certain features that the web application currently does not support, like policy groups, requests, posts and comments. For a better understanding, we now describe each table individually.

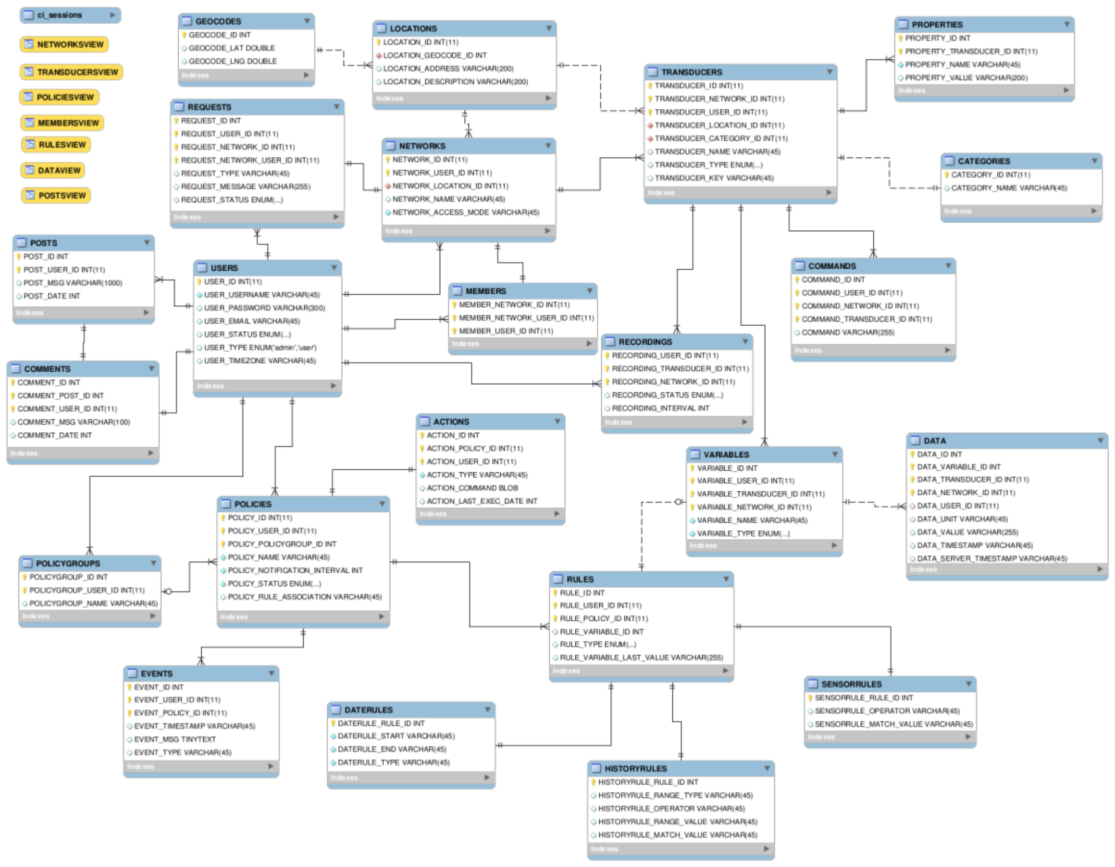


Figure 38 - Database Model

GEOCODES – This table stores the geographical reference of locations.

LOCATIONS – This table stores the location of sensors and networks.

USERS – This table stores information about users, either normal or admin users.

NETWORKS - This table stores sensor network records.

MEMBERS – This table stores the members of a network.

TRANSDUCERS – This table stores information about sensors and actuators.

REQUESTS – This table stores information about user request, which can be simple messages between users.

PROPERTIES – This table stores dynamic properties of transducers.

POSTS – This table stores user posts or news in news wall.

COMMENTS – This table stores comments to posts.

RECORDINGS – This table stores the recordings request by authorized users.

CATEGORIES – This table stores categories for sensors.

COMMANDS – This table stores actuator commands.

POLICYGROUPS – This table stores groups of policies.

POLICIES – This table stores information about policies.

ACTIONS – This table stores information about actions.

EVENTS – This table stores the log of the execution of one action.

RULES – This table stores information about rules.

VARIAVLES – This table stores sensor parameters.

DATA – This table stores sensor parameters values.

DATERULES – This table is a specialization of rules and stores information about date rules.

HISTORYRULES – This table is a specialization of rules and stores information about history rules.

SENSORRULES – This table is a specialization of rules and stores information about sensor rules.

ci_sessions – This table is used by the PHP framework to store user session data.

The database views (represented in yellow in Figure 38), are the result of queries that join multiple tables, reducing the amount of code to retrieve information from database, and also are used to hide sensitive information. For example the *POSTSVIEW* table uses data from two tables: from USERS table, the name and id of the author that created the post, and from POSTS table, the number of comments for each post.

3.4.5.2 Web Application Files and Views

The developed web application contains 14 controllers, 10 models and 7 different views layouts. We now give a brief description of the developed code and the views.

Controllers:

actuator.php – this controller contains *Create Retrieve Update Delete* (CRUD) functions and functions to call commands in actuators.

category.php – this controller basically contains CRUD functions.

dashboard.php – is an experimental controller to enable user custom panels (with sensors, actuators and graph widgets).

data.php – the data controller contains CRUD functions, and functions to export and retrieve sensor data for charts (e.g., average).

gallery.php – this controller is responsible for adding and managing images in sensors and networks galleries.

location.php - this controller basically contains CRUD functions.

main.php - this controller contains the login, logout and register functions and is the first controller to accept user connections.

network.php – this controller has the functions to manage the user networks.

policy.php – this controller has CRUD functions for policies, rules and actions.

sensor.php – this controller has CRUD functions to add sensors and retrieve sensor information.

system.php – this controller contains functions to monitor the platform services, users, and to edit the platform configuration.

user.php – this controller contains CRUD functions related to users.

wall.php – is an experimental controller to enable user news and user posts, enabling users to comment when a new user adds a resource in the system, or when any user shares a post.

xmpp.php – this controller is responsible accepting HTTP POST requests with sensor data to publish via XMPP.

Models:

actuator_model.php **category_model.php** **data_model.php** **location_model.php** **network_model.php** **policy_model.php** **sensor_model.php** **user_model.php** **wall_model.php**	These models mostly contain CRUD functions. They accept an array of options and then use CodeIgniter Active Record to create dynamic SQL queries.
gallery_model.php	The gallery model is different from other models, since it does not interact with database but with the file system. This controller is responsible for managing the gallery images on the system.

Views:

In the views we have mostly *phtml* files, these file are composed by HTML, PHP and some JavaScript code. These files are executed using the supervised data coming from controllers, and then the output is sent to the user web browser to be rendered. Our web application uses layouts, enabling multiple output formats such as HTML, PDF, RSS.

ajax – the *phtml* files in this layout folder, mostly contain operation outputs (success and errors messages) for AJAX requests.

csv – this layout folder contain views that are used to generate CSV files for exporting and for Dygraphs chart component.

default – when no layout is specified the default is assumed. This folder is a HTML layout; it contains all views rendered to the user's web browser.

json – in this layout, all views are rendered in JSON format. For example all networks and sensors lists are requested in this format to be used in chain dropdown boxes.

ofcjson – this layout contain views dedicated to generate OFC2 configuration files.

rss - this layout contains views for generating RSS files.

xml – this layout contains views that export sensor data in XML format.

3.4.5.3 SOXLIB modifications

The sox library implements the essential extension features (administration and pub-sub). But for creating more advanced applications a few changes was needed.

The library provided a function to retrieve subscriptions, but not for affiliations. We added a function in the library to retrieve affiliations, with the affiliation list the users can see what nodes they own and the user role to a node (owner, member, publisher, etc).

The original SOXLIB supported subscribing and unsubscribing, but it did not have support for unsubscribing from multi-subscribe. A user may subscribe to the same node several times with or without different subscription options, for these situations the subscription id must be used to unsubscribe from a particular subscription. We did a small modification on the library to support unsubscribing from a specific node with a subscription id.

As we intended to create system services, the services would have to handle problems when the connection goes down. SOX library had a function to handle this situation, but it only reported the cause of the problem. We needed to handle the problems in the service (e.g. when internet connection goes down), to fix this we cloned the function that starts the XMPP connection and we added one more argument. This argument points to a handler function, to be executed when the connection has problems. Thus the services we developed handle the connection problems by their own, being possible to reconnect after a XMPP connection problem.

3.4.5.4 Dygraphs modification

The Dygraphs chart component accepts the sensor values in CSV format, with date in first column and the next columns for sensor values.

```
Date,                    temperature,   gas_amount
2010/10/09  20:17:47,    10.78,         50.12
2010/10/09  20:18:49,    10.87,         50.15
```

For a successful integration of this chart component in the platform, the support for multiple data series was required. Meanwhile the Dygraphs had an assumption that became specific problem, if a data series did not have any value for the specific time, it was assumed as a gap. However in our system, the data is received at any time and with different intervals, making the chart to display only dots (gap before and after). To solve this issue we modified the Dygraphs source code to support the representation of data series with different time intervals.

4 Case Studies

During the project development, we implemented basic cases, such as monitoring a room door. When the platform has become more developed, we started to setup uses cases to test the platform and to demonstrate the platform potentials.

4.1 Mail Box

The mail box case of study was the first to be implemented. The idea for this use case consisted on alerting the user when a mail arrives to the physical mail box. Using a notification for new mails, avoids people walking to mail box and verify if there is mail or not.

The plan to concretize this idea was to use an *Arduino*[20] board, with a mail detector, a door sensor and a simple RF transmitter module. The reasons to use the *Arduino* board were the ease of use, fast deployment, affordable and easy interfacing.

To detect when a mail box door is open, a magnetic door switch was used, the sensor is composed by two parts, the magnet and the magnet detector. The magnet was attached to the rotating lock mechanism and the magnetic sensor was glued to mail box, as shown in Figure 39.

To transmit data from mail box to the sensor gateway, we used a radio frequency (433 MHz) transmitter module. These modules are cheap and very easy to use, and can be found in many applications to do simple remote control.

[20] http://www.arduino.cc

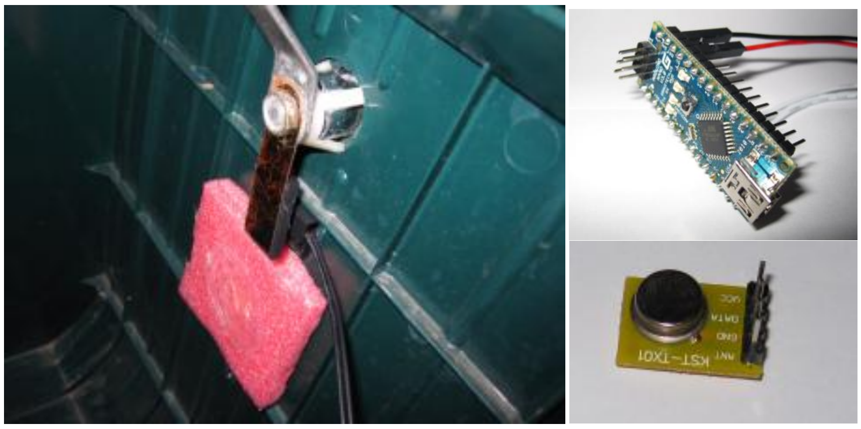


Figure 39- Magnetic Door Sensor, RF Module and Arduino

After testing a distance sensor for detecting new mails entering the mail box, the detection rate was very low (4 in 10), we decided to build a mail detector. This sensor acts like a button, when a new mail is inserted; the metal plates touch, making a closed circuit.

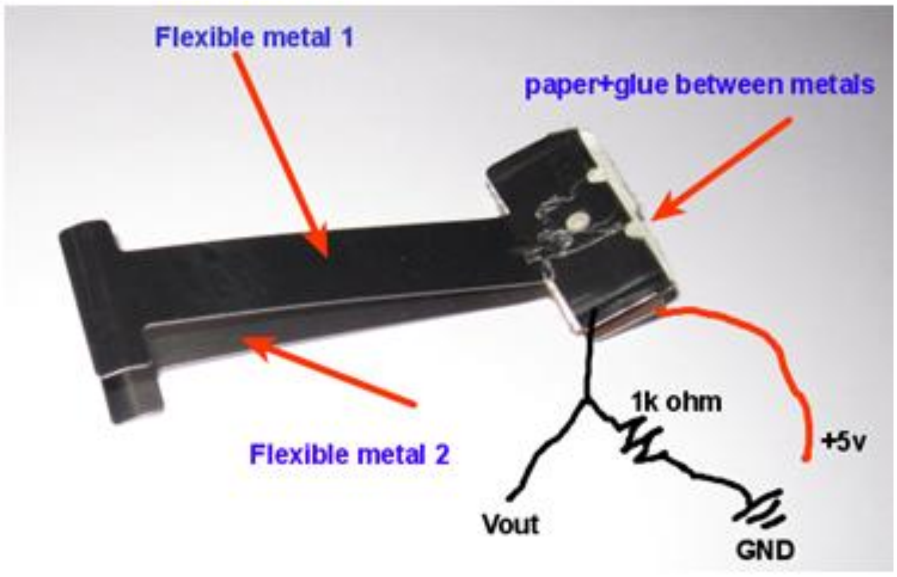


Figure 40 - Mail Detector

In the installation, we used an external power supply unit, but a battery or a solar panel can be used. The mail sensor was attached to the box, a few centimeters above mail box slit. In effectuated tests (using a regular letter), the sensor detected the insertion of all mails.

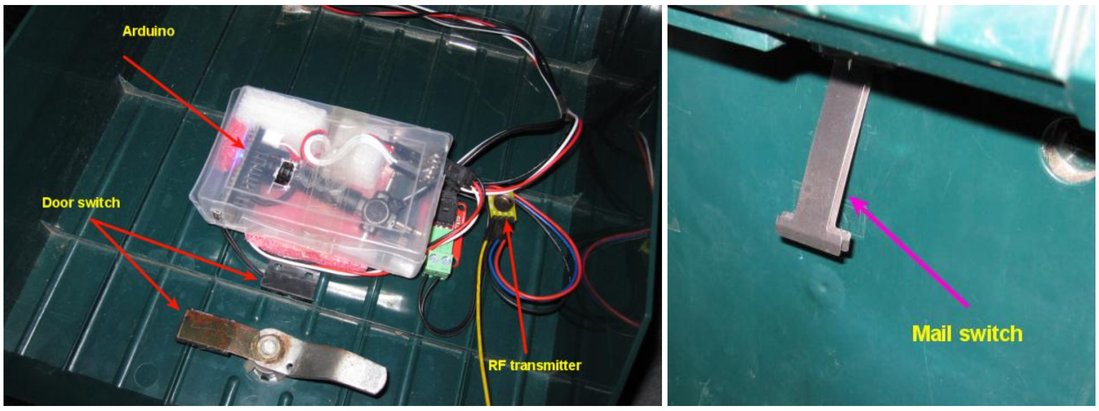


Figure 41 – Installed Components

In web application we registered a new sensor in the network, and we registered two variables for the mail sensor, *new_mail* and *door_status*, the *new_mail* variable is updated to server when a new mail is detected, and the *door_status* is updated to server when the mail door is closed or open.

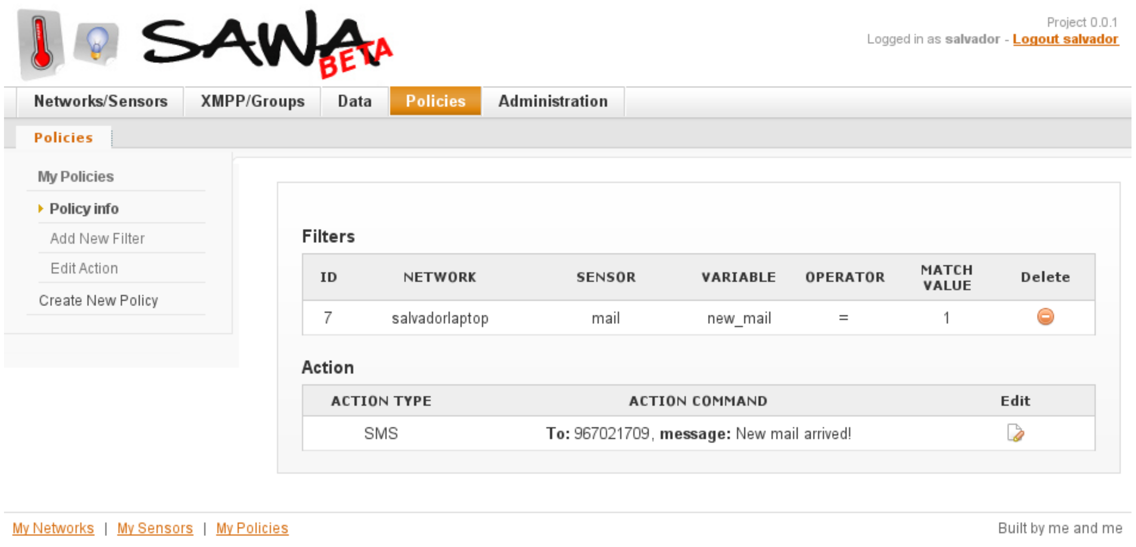
The local sensor network was built, using an *Arduino* with an RF receiver module and with an Ethernet shield. This gateway is resource limited, and only can do simple HTTP POST and GET operations. The message protocol used between sensors and the RF gateway was defined as fallow.

message_number#sensor_id#variable_id#variable_value

The tested RF module works well for small distances in clean air, since we used these modules indoor, the thickness of walls affect the transmission reliability. In order to minimize the effects of distance and the objects between the transmitters and the receiver, the transmitters were programmed to send five times each message. The message number is used to sensor base know if the message has been accepted before or not.

The sensor id identifies a sensor in the network, and the variable id is the id of the sensor parameter. When a message is sent from transmitter, the gateway (receiver) parses the message and makes an HTTP POST call to the web application, which then publishes via XMPP.

In the web application a policy was created, as shown in Figure 42 with a rule that applies to the *new_mail* variable from mail sensor in the *salvadorlaptop* network. This policy matches when the uploaded *new_mail* value is "1", and the SMS action is triggered with the message "*New mail arrived!*".



Filters

ID	NETWORK	SENSOR	VARIABLE	OPERATOR	MATCH VALUE	Delete
7	salvadorlaptop	mail	new_mail	=	1	⊖

Action

ACTION TYPE	ACTION COMMAND	Edit
SMS	**To:** 967021709, **message:** New mail arrived!	✎

Figure 42 - Policy Details

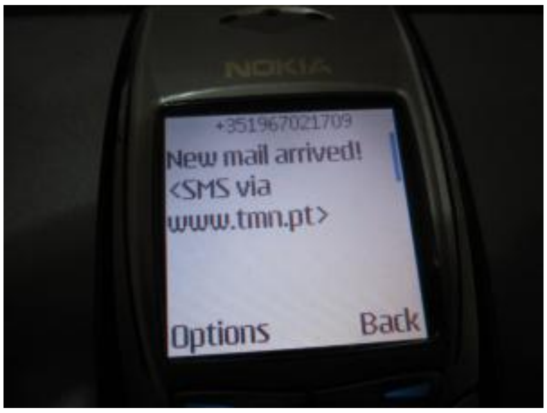


Figure 43 - SMS Notification Message

With this experience we have learned some lessons. Hardware choices are very important, because the physical world limits in various conditions. In this experience, the distance, rain, energy, mail box material, and others constraints must be taken into account. But in the end, the experience was useful, with a simple policy, we got notifications when new mails arrived, and we could see the mail history in charts. And in future we could develop a custom mail sensor node, much smaller and with better hardware, to do the same as this experience.

4.2 Refrigerator

The idea to monitor a refrigerator was to test the system and possible monitor the impact of opening the door in energy consumption. To accomplish this idea, we ordered a temperature sensor, a photo-resistor sensor and a non-invasive current transformer (CT) sensor. The *CT* sensor required external circuit and our knowledge was very limited in that field, we skipped the energy monitoring, and we added a gas sensor to detect butane gas leaks, and a presence sensor to monitor presence in the kitchen.

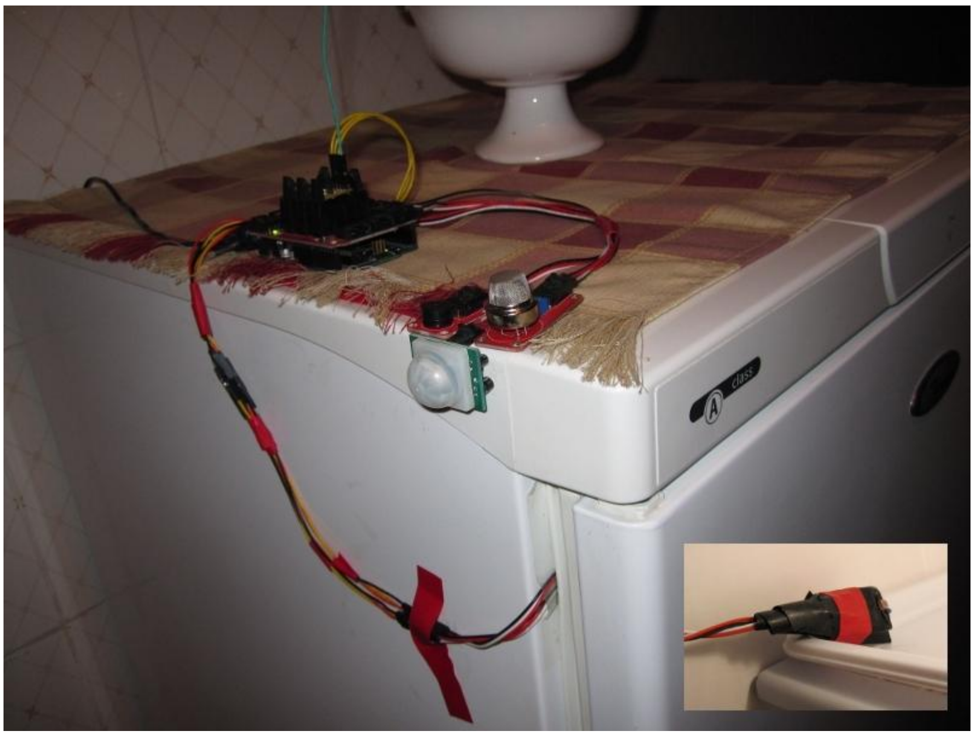


Figure 44 - Refrigerator Sensors

Every refrigerator has a light which is turned on when the door opens, and is turned off when the door is closed. To detect when the door was open, a photoresistor sensor was used, a photoresistor can be defined as "a resistor whose resistance decreases with increasing incident light intensity" [38], by other words, it measures the light intensity. When the door is closed (no light present) the sensor will report "0" and when the light intensity is more than zero, means the door is open.

Along with light intensity sensor, was attached a thermistor, which is a resistor whose resistance varies with temperature [39], to monitor the refrigerator temperature. The gas sensor, is high sensitive to gases, and was used to monitor the presence of gases in the kitchen, more particular the butane or propane gas.

The buzzer as shown in Figure 44 emits a sound alert when the gas sensor detects high values. The installation was very easy to achieve, like in mail box experience, we used an *Arduino* board platform, a RF transmitter module, and the same protocol for sending messages.

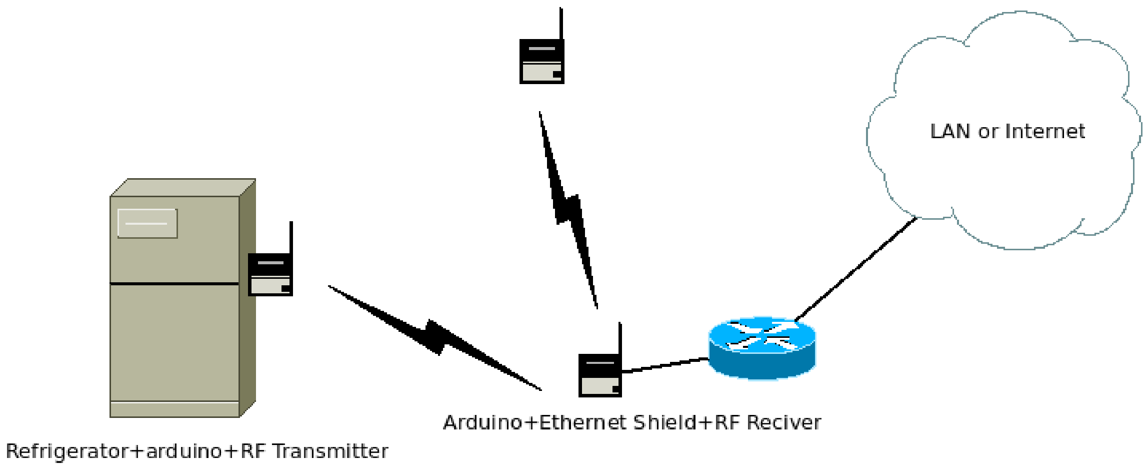
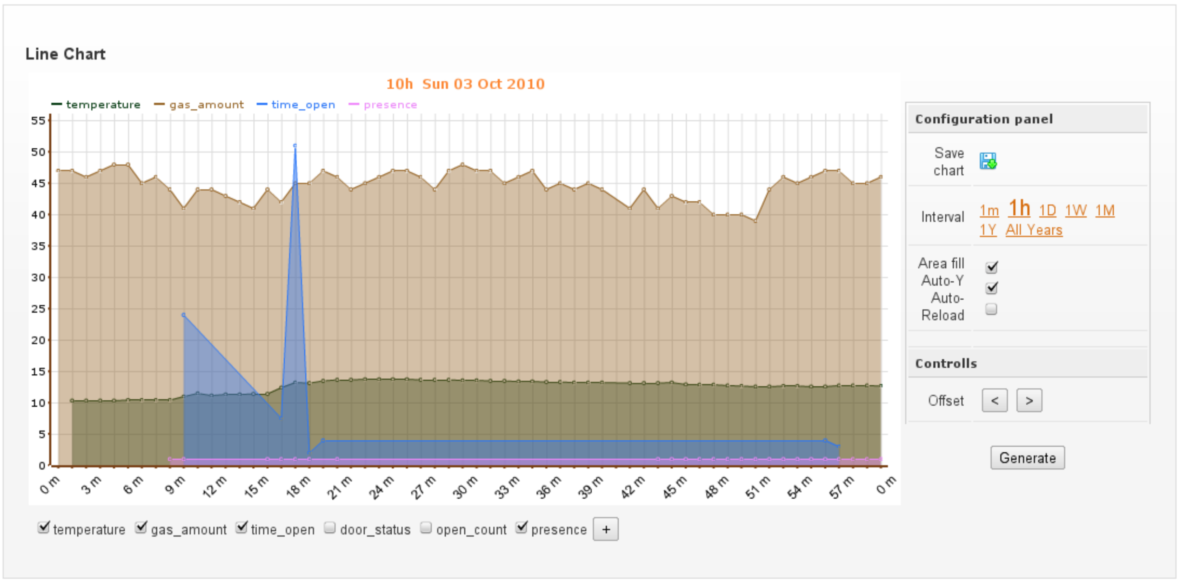

LAN or Internet
Arduino+Ethernet Shield+RF Reciver
Refrigerator+arduino+RF Transmitter

Figure 45 - Sensor Network



Configuration panel

Save chart

Interval 1m **1h** 1D 1W 1M 1Y All Years

Area fill ✓
Auto-Y ✓
Auto-Reload ☐

Controlls

Offset < >

Generate

✓ temperature ✓ gas_amount ✓ time_open ☐ door_status ☐ open_count ✓ presence +

Figure 46 - Recorded Sensor Data

The chart from Figure 46 represents the data from four variables within one hour (10h). The blue line contains seven points, with the time in x-axis and the number of seconds in y-axis. The darkest brown line is the refrigerator temperature and the light brown represents the gas sensor data. Finally the line with magenta color represents presence in that hour.

The graph in Figure 47 represents in seconds the average time, which the refrigerator door was open. On Figure 48 is represented the average number of times the refrigerator door was open by hour.

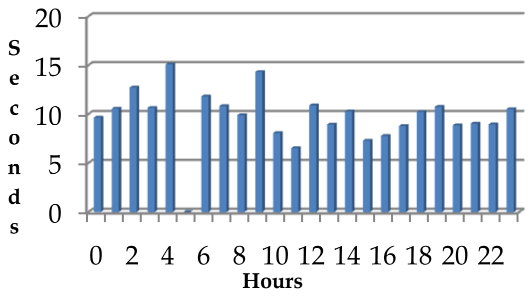


Figure 47 – Door open average time

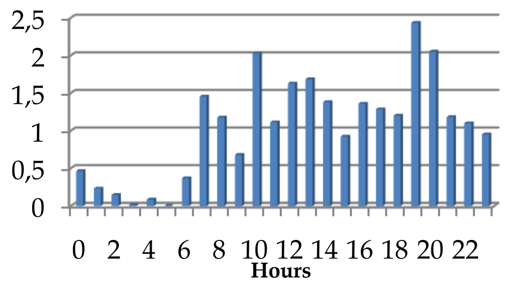


Figure 48 - Average open count by hour

The presence graph, in Figure 49, shows the presence count average for each hour in the kitchen.

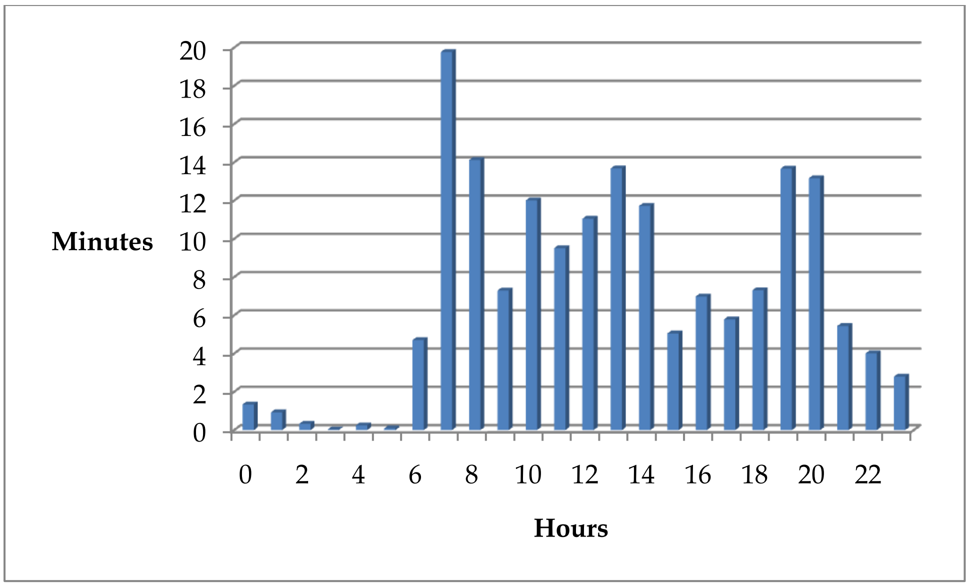


Figure 49 - Average Presence in Kitchen

The Chart in Figure 50 shows the average presence in the kitchen, the door open count frequency, and the average door open time for each hour.

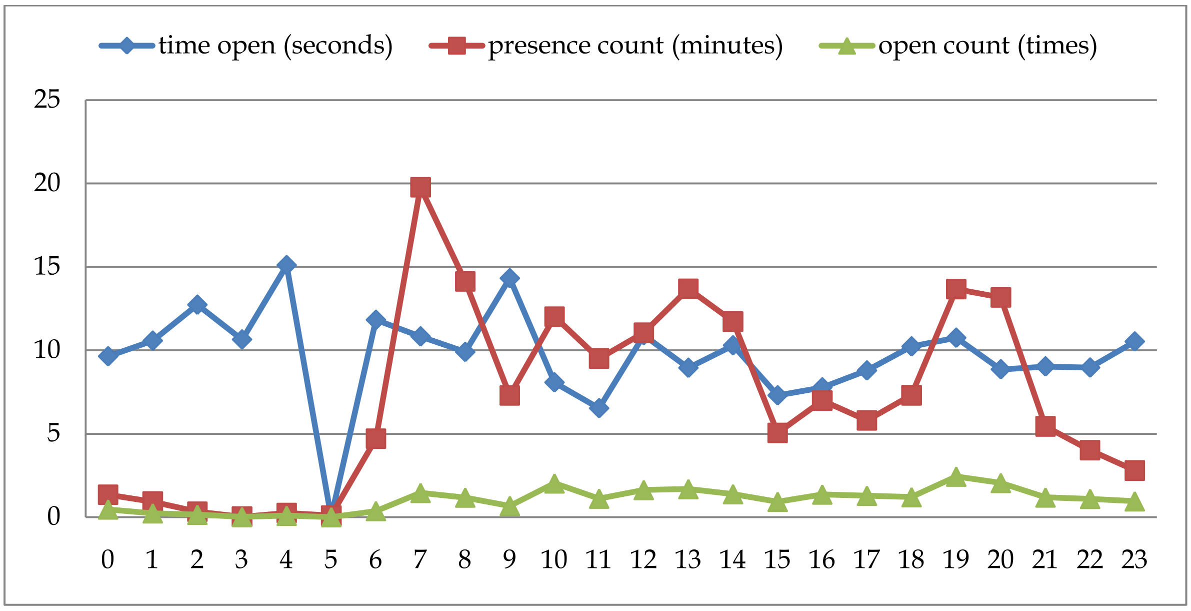


Figure 50 - Presence and Refrigerator

4.3 Wine Monitoring

4.3.1 Objectives

The objective of wine monitoring use case was to gain scientific understanding of the vinification process of Madeira wine. Was intent to use sensors to study various parameters of the vinification process, as well use actuators to control the environment.

Figure 51 - Wine Tank

Figure 52 - Wine Laboratory

4.3.2 Selecting Sensors

The first step was to search for related projects, and in overall we find the basic temperature measurements and commercial projects that monitor few parameters (e.g, CO2). The next step consisted in search of sensors, and we set a few requirements to select the sensors:

Easy interfacing – the sensor should be easy to interconnect with microcontrollers, and not require external circuits.

No maintenance – the sensor should operate for long time without maintenance.

Affordable – the price of sensor should be relatively affordable, considering the possibility of extending to multiple wine tanks.

The following tables 8 and 9, resumes the reasons to select or not the various types of sensors.

Reasonable – The sensor seemed easy to interface and use.

Expensive - The system or sensor is expensive.

Specific HW and SW – The parameter can be measured with a special lab machine, which requires specific hardware and software.

No sensor - No available standalone sensor for sale.

Maintenance – The system or sensor requires human intervention, and/or chemical products.

Parameters	Order	Use
pH	Yes	(+) Seems reasonable
Temperature	Yes	(+) Reasonable
Color	No	(-) Expensive (-) Specific HW and SW (-) No Sensor
Dissolved Oxygen	No	(-) Expensive (-) Specific HW and SW (-) Maintenance
Volatile acidity	No	(-) No sensor
Total acidity	No	(-) No sensor
Oxidation	No	(-) No sensor
Ethanol	No	(-) Expensive (-) Specific HW and SW (-) Maintenance
Sulfur Dioxide	No	(-) No sensor
Specific gravity	No	(-) Expensive (-) Specific HW and SW (-) No sensor

Table 9 - Liquid Measurable Parameters

Parameters	Order	Use
Relative Humidity	Yes	(+) Reasonable
Carbon Dioxide	Yes	(+) Reasonable
Monoxide Carbon	Yes	(+) Reasonable (-) Less important in wine
Alcohol	Yes	(+) Reasonable

Table 10 - Gas Measurable Parameters

4.3.3 Sensor testing and connections

After the arrival of the ordered sensors, the testing phase started as described next.

Alcohol sensor – The MQ-3 gas sensor, has a high sensitivity to alcohol and small sensitivity for benzene, and is common used in alcohol checkers and breathalyzers [40]. To use and test this sensor, we had do use a variable resistor as explained in datasheet. The calibration of the sensor is made through the potentiometer (variable resistor) and the sensor must have at least more than 24h of preheat time.

CO2 module - The CO2 module, uses the MG811 CO2 Gas Sensor, in order to use the MG811 sensor, would be necessary a complex external circuit to amplify the sensor output. The CO2 module implements the required circuit and has the MG811 CO2 Gas Sensor. The CO2 sensor has a good CO2 sensitivity and selectivity for typical applications, which include, air quality control, fermentation process control and room CO2 concentration detection [41].

The CO2 module has a potentiometer for alarm calibration and requires at least five minutes of preheating time. In order to use this module, a pin had to be soldered in sensor amplified output.

PH probe – The PH probe, has a measurement range from 0 to 14 pH units, and has a fast response time. This pH probe contains an amplifier circuit to amplify the signal from electrode. Most available pH probes require external circuit or hardware to make readings. We specially ordered this probe because of the amplifier circuit, making possible an easy interfacing with the microcontroller.

Temperature sensor – the temperature sensor is a thermistor with a steel head, being appropriate for temperature measurements in liquids.

LM34 temperature sensor – LM34 is a high precision Fahrenheit temperature sensor, this sensor is an integrated circuit, whose output voltage is linearly proportional to the Fahrenheit temperature [42]. The calibration for this sensor was not necessary, and no complex wiring was needed.

Relative humidity sensor – The HIH-4030 relative humidity sensor, has a near linear analog output and has fast response times. Common applications are refrigeration equipment, HVAC systems, metrology and others more. No special effort was necessary for use this sensor, since it has a simple interface.

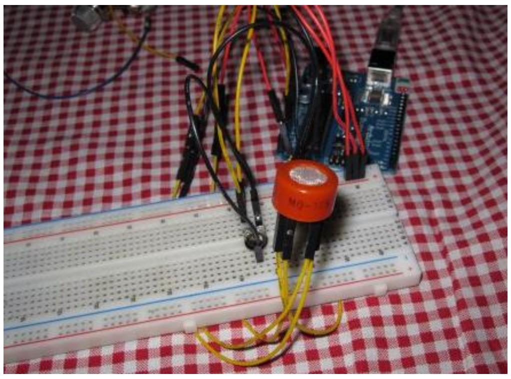


Figure 53 - Alcohol Sensor

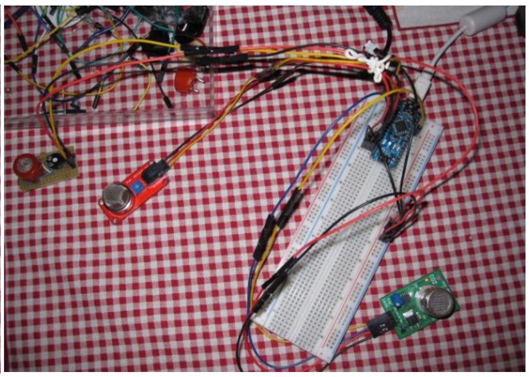


Figure 54 - Gas Sensors

The Figure 55 schematizes the wiring from the sensors to the microcontroller.

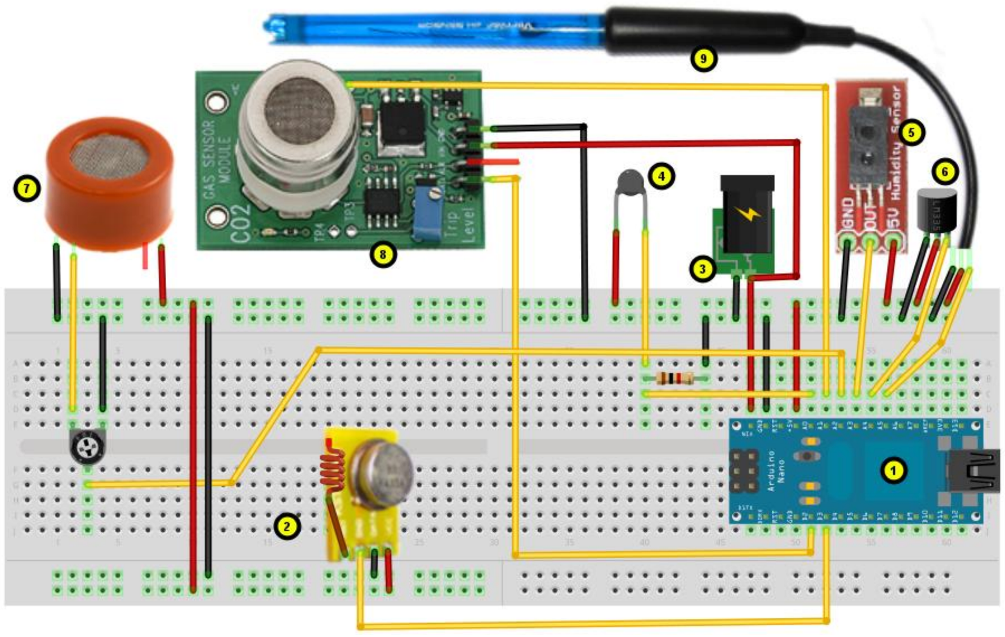


Figure 55 - Wiring Scheme

1 Arduino Nano	**4** Thermistor	**7** Alcohol Sensor
2 RF Module	**5** Relative Humidity Sensor	**8** CO2 Module
3 Direct Current Jack	**6** Temperature Sensor	**9** PH Probe

4.3.4 Probe Construction

Four attempts were made to integrate all these sensors, components and wires, into one single probe. In the first attempt, we used a stainless tube with 3 cm of diameter. The first difficulties were the drill process to make holes for gas sensors, and the second problem was the limited space and the difficult of moving the cables.

In second attempt, a common plastic tube was used, with more flexibility to make holes for sensors and with more length in diameter. The lack of space and difficulty for moving and wiring sensors inside the tube made us abandon this approach.

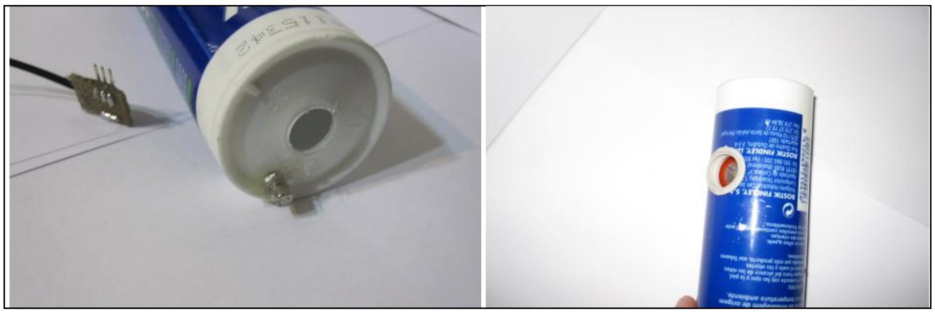


Figure 56 - Second attempt Probe

The third attempt consisted in having two modules; the liquid module would have the pH probe and the temperature sensor. And the other module would have the gas sensors and the microcontroller. The liquid module was filled with a heavy silicone, forcing it to submerge at certain level. The wiring to the *Arduino* platform was made with less difficulty than the previous experience, but there were difficulties in wiring everything. The problems started when the two modules were connect, the whole probe not kept equilibrated and the water level was too much close to the gas sensors.

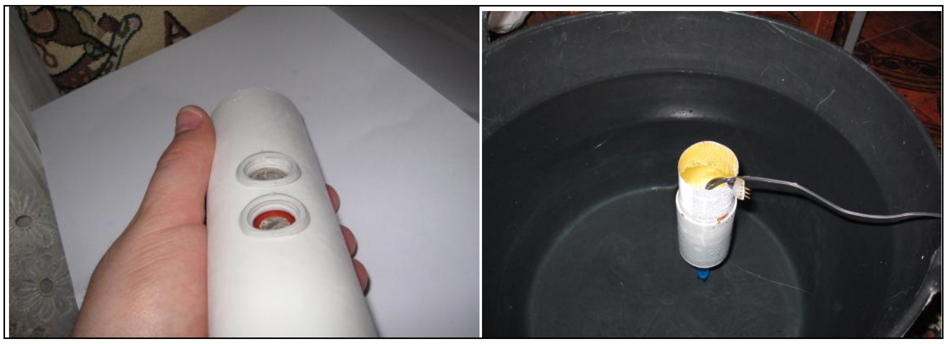


Figure 57 - Third attempt Probe Modules

From this attempt, was decided to split in two parts, the gas part where we would have the gas sensors and the microcontroller, and the part floating in the water.

In the construction of final solution, a plastic tube was used to hold the pH probe and temperature sensor. A heavy metal object was added to keep the module more equilibrated. To build the gas module, a rectangular plastic box from an old scanner was used. Using this material the manipulation and wiring was much easier to achieve.

In Figure 59, in the left side of the black box, is the alcohol sensor *PCB* (Printed Circuit Board) and under the same *PCB* is the humidity sensor with the temperature sensor. The green *PCB* is from CO2 module and on the right side of this module is a soldered mini distribution of ground and positive voltage. The distribution board was needed because *Arduino Nano* has only two ground pins and one pin for five volts.

In the right side of the box is *Arduino Nano*, with analog and digital pins connected to sensors; ground and voltage pins connected to the distribution *PCB*. *Arduino* is also connected to a DC jack, where receives energy from.

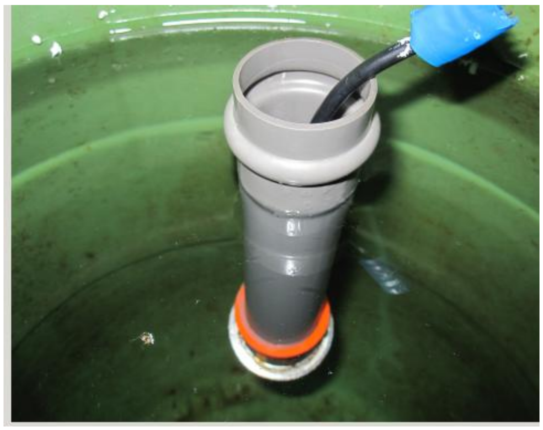


Figure 58 - Liquid module

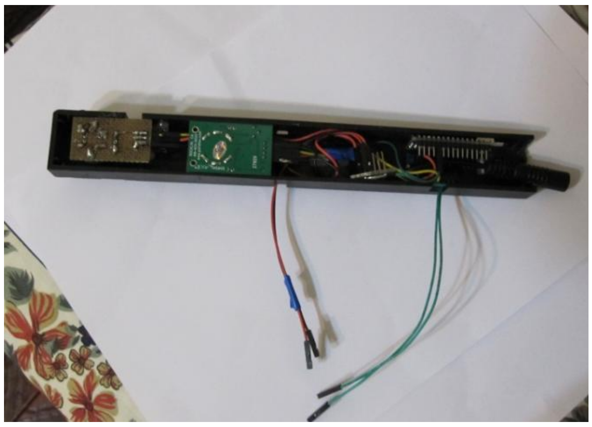


Figure 59 - Gas Sensors with Microcontroller

This construction had a problem, the temperature and humidity sensor was between the two gas sensors, these gas sensors contain heaters which increase the temperature of surrounding objects. The probe was installed in the wine tank, and after one month we did an improvement to fix the problem.

After the previous experience we came with a simpler solution, using modules, one for liquid sensors, and the other for gas sensors and to be attached to the top of the wine container. The upper module contains two parts; the top one contains the micro-controller, power connector and RF modules. The bottom part contains the c02, alcohol and relative humidity sensors.

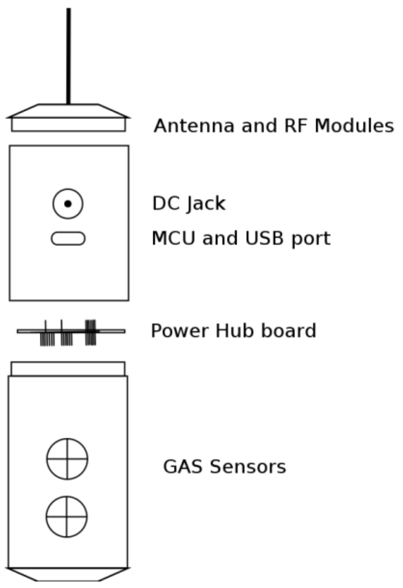


Figure 60 - Gas Module Parts

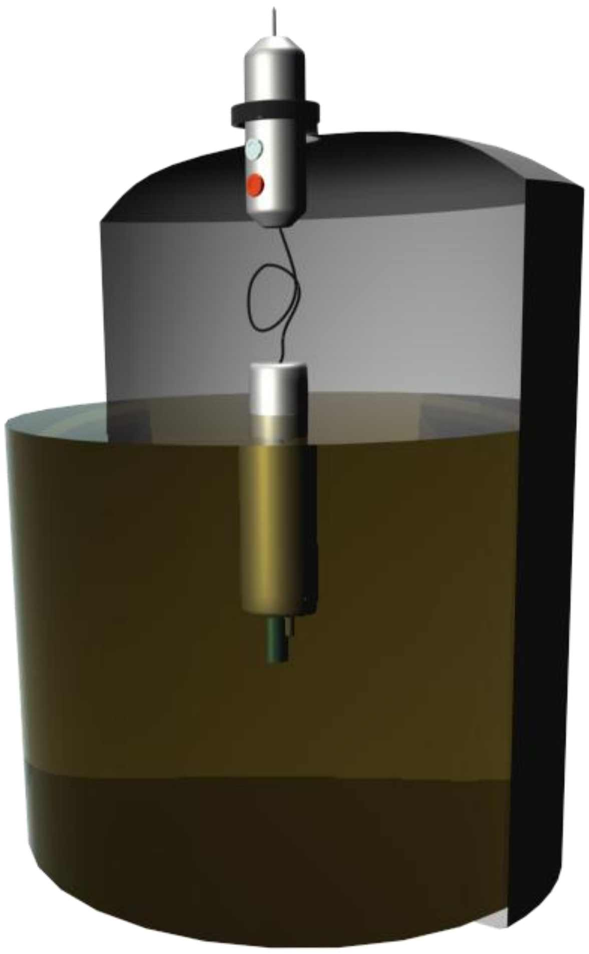


Figure 61 - Idealized Sensor Probe

To construct the gas module part, we cut in half a sealant silicone tube, and we used the tube cap to add the transmitter/receiver modules with an antenna (yellow wire). In the upper part of the gas module was added the *Arduino* board, with the *USB* port exposed to exterior, and we added a *DC* jack. In the tube part destined for gas sensors, was drilled three holes with the diameter to fit each sensor.

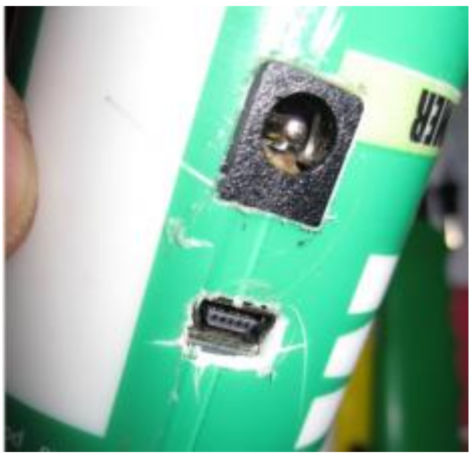


Figure 62 - USB and DC Jack

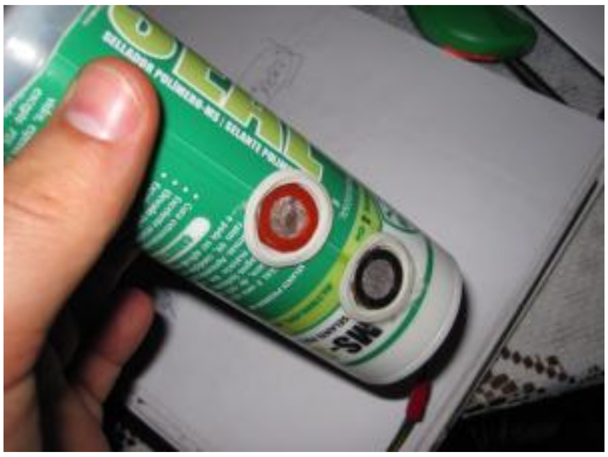


Figure 63 - Gas Sensors

The Alcohol and CO2 sensors were attached to the tube using tube sleeves and glued with hot glue. The power hub board was built to connect the sensor power pins and sensor output pins. The hub allows an easier plug of these two parts.

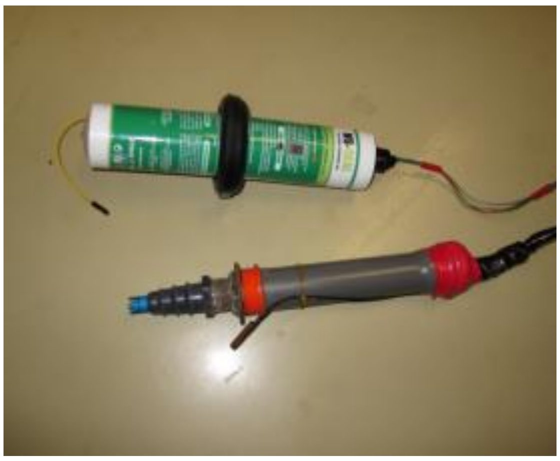


Figure 64 - Final Probe

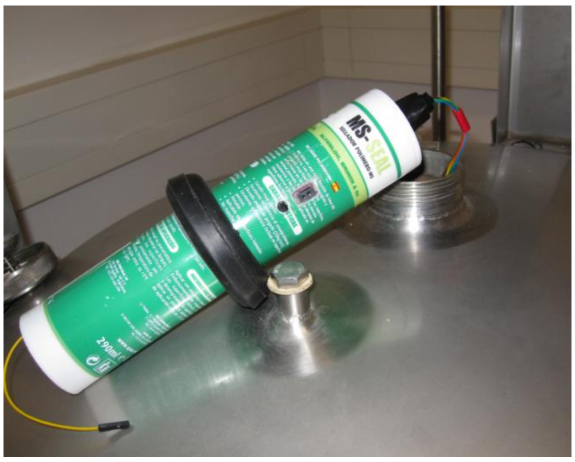


Figure 65 - Installation

4.3.5 Installation

After the construction of the probe, we used an elastic screw cover to hold the probe in the openning of wine tank, and also to avoid major temperature and gases changes. The installed sensor gateway was an Arduino board with an Ethernet shield, being responsible to connect with server, accept RF messages from the probe and then upload the sensor data via HTTP POST.

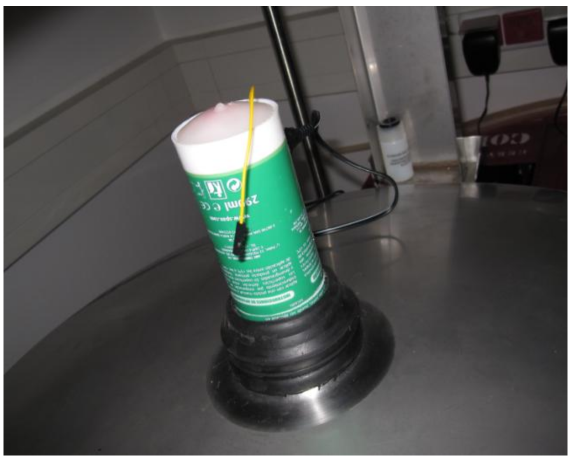


Figure 66 – Installed Probe

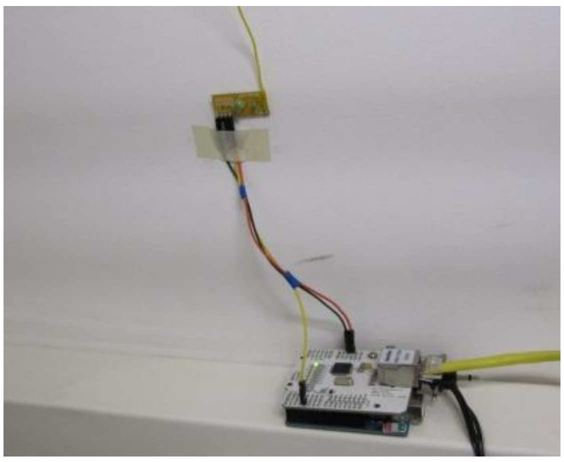


Figure 67 - Sensor Gateway

The sensor calibration consisted in using liquids with different pH values, for the pH probe. For gas sensors, we used the ambient, the sensor datasheet charts, and linear formulas to calculate the approximate sensor values.

4.3.6 Sensor Data

We have created a network in the system named “wine_lab_network” and created a sensor named “wine_probe_1”. We added 11 variables for the sensor, some variables contain raw values and some contain processed values. The advantage of uploading the raw value is the future possibility of correcting processed sensor values when the calibration or data processing was not done correctly.

ph – the processed value of ph.

ph_raw – the raw voltage (in mV) sensed from analog sensor output.

c02_raw – the raw voltage (in mV) sensed from sensor output.

co2_ppm – the processed value of co2_raw in parts per million (ppm).

c02_per – the processed value of co2_ppm in percentage.

temp_wine – the processed temperature measured in wine.

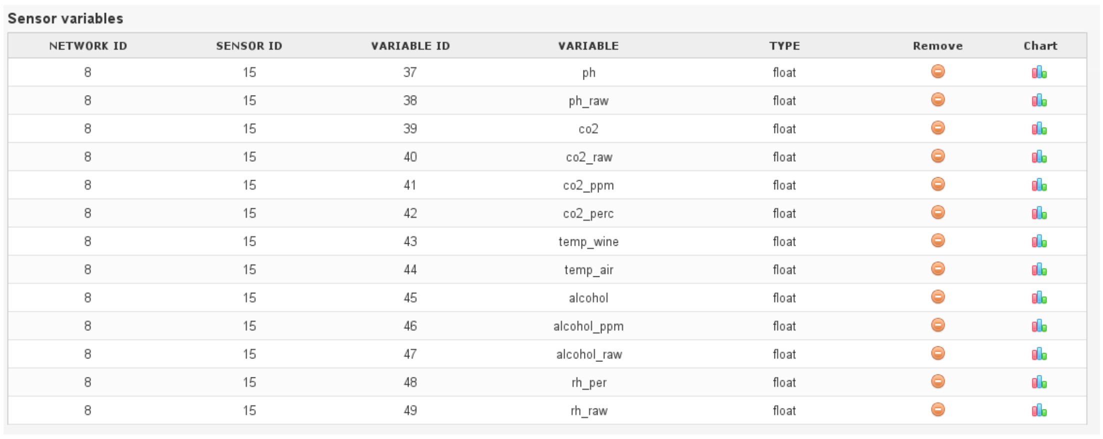
temp_air – the temperature measured in air.

alcohol_raw – the raw voltage (in mV) adquired from sensor output.

alchohol_ppm – the processed value of alcohol_raw in ppm units.

rh_raw – the raw voltage measured from relative humidity sensor.

rh_per – the relative humididy percentage (0% to 100%), processed from rh_raw and temp_air values.


Sensor variables

NETWORK ID	SENSOR ID	VARIABLE ID	VARIABLE	TYPE	Remove	Chart
8	15	37	ph	float	⊖	📊
8	15	38	ph_raw	float	⊖	📊
8	15	39	co2	float	⊖	📊
8	15	40	co2_raw	float	⊖	📊
8	15	41	co2_ppm	float	⊖	📊
8	15	42	co2_perc	float	⊖	📊
8	15	43	temp_wine	float	⊖	📊
8	15	44	temp_air	float	⊖	📊
8	15	45	alcohol	float	⊖	📊
8	15	46	alcohol_ppm	float	⊖	📊
8	15	47	alcohol_raw	float	⊖	📊
8	15	48	rh_per	float	⊖	📊
8	15	49	rh_raw	float	⊖	📊

Figure 68 - Sensor Variables

In Figure 69, is shown the available sensor data from the first nine days of July, the pH values maintain constant at 3.35 ph units, with some variation of 0.01 units. The temperature value also maintains constant; the temperate in the gas is in average 31 degrees Celsius and the temperature in the wine rounds the 30ºC.

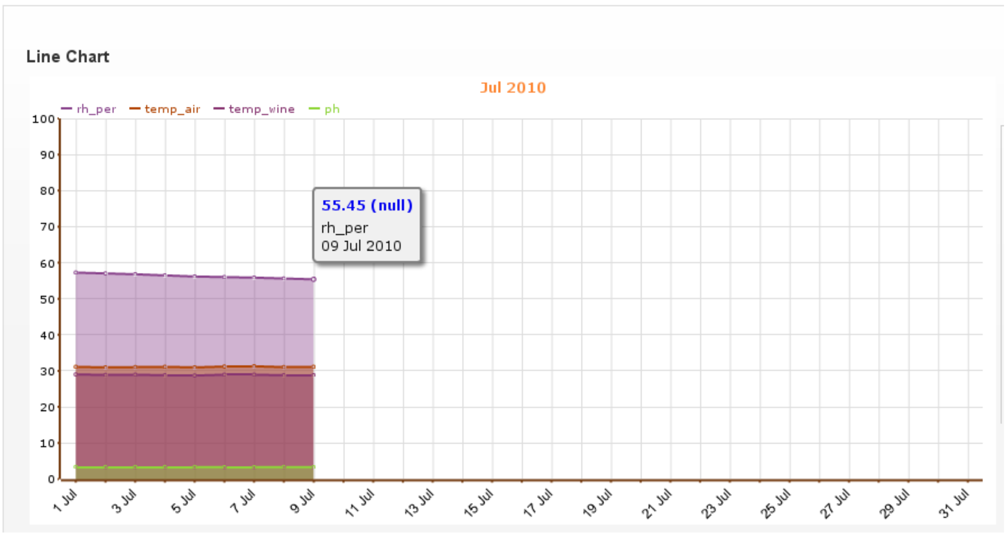


Figure 69 - Sensor Data from July

Drilling by day, we can observe by hour the same parameter values; the first aspect we can notice is the dropping of temperature and an increase between 18h and 19h. This process also happens with the refrigerator, but instead of cooling the system heats, to maintain the temperature constant at 30ºC.

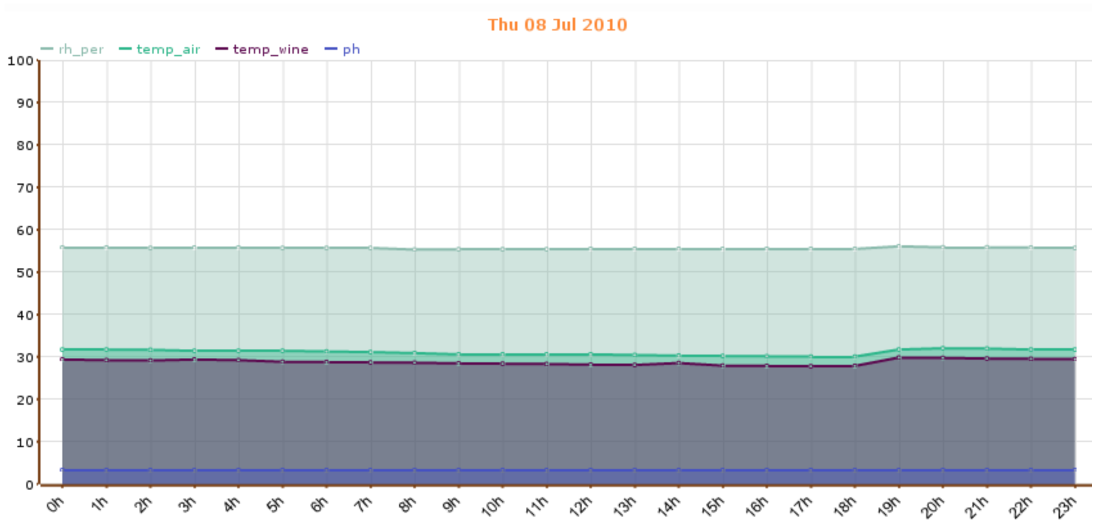


Figure 70 - Drilling Sensor Data by Day

4.3.7 Conclusion & Results

In this use case, we firstly focused in monitoring only, and we have installed the hardware just for the monitoring process. We have selected simple components like the RF modules and limited devices (*Arduino* in gateway). But the choice for the cheap and simple hardware penalized us. As example, the *RF* modules are limited to transmit 27 characters at once, do not have any transmission protocol, no encryption support and no data integrity mechanism. The use of a resource limited sensor gateway, do not allowed secure connections and support for actuation.

We faced with the problem of resistance of materials/construction in long term monitoring in harsh environments. A few weeks after prove improvement, the liquid module was flooded by wine, destroying the liquid module and circuits around.

During the writing of this thesis we found that two new interesting products. A company focused in wireless sensor networks, *libellium*[21], is selling sensor boards and gas sensor for their motes. Another relevant project in wine is *fermonitor*[22], which promotes a probe (currently in development) to measure interesting parameters (specific gravity, sugar, alcohol, ABX, pressure) for beer monitoring.

We suggest for the next improvements, the use of more reliable hardware in communication (e.g., *xbee*), and the use of a sensor gateway with resources and capable of establishing secure connections (e.g., desktop, laptop, mini-pc). For the probe construction we suggest the use of better materials (e.g., insulate materials), and the construction of custom parts. For the probe sensing components, we suggest the integration of more liquid sensors and gas sensors.

Despite all problems and difficulties, this use case was a starting point in wine monitoring, like any project development, we learn and improve with iterations. With more iteration we could develop a more reliable probe for wine monitoring, and for beer. This probe then could be used as tool for gaining

[21] http://www.libelium.com
[22] http://fermonitor.com

scientific understanding of Madeira wine fermentation process, and as a utility tool to know when the wine is ready to go to market.

5 Tests

We tested the system in order to check the performance and reliability under various conditions.

5.1 Reliability & Performance

The first test consisted in publishing the same sensor data for 1.000 times, using the two methods *XMPP* and *HTTP POST*. The objective was to measure the time to complete and the success rate. To achieve this, we developed a small program to connect to the *XMPP* server and publish to a sensor node 1.000 times.

We created two programs for testing each method, for *XMPP* method the program started by creating an *XMPP* connection, then a *SOX* message with sensor data was created, and finally a cycle to publish the message and report failures. For *HTTP POST* method, we created a small *C* program that used *libcurl POST* functions. The program started by creating a *POST* form request and then started requesting the server in a cycle.

Performance/Reliability		LAN		Internet (hci.uma.pt)	
Method	Tests	Time (s)	Success	Time (s)	Success
XMPP	1 000	23 sec	100%	86 sec	100%
HTTP POST		266 sec	100%	280 sec	100%

Table 11 - Publishing Results

The Table 11 shows the results effectuated in 7 October 2010, we did the same tests in a local area network and a server located in internet. The first aspect to notice is the success rate of 100%, making the system publishing very reliable. The times to publish in the two methods are smaller in LAN than internet server, and there is a huge difference in publishing time (at least 3 times faster) between both methods.

Performance/Reliability		Internet (hci.uma.pt)	
Notification type	Tests	Success rate	Average time (s)
Email	10	100%	2 sec
URL		100%	2 sec
SMS		100%	6 sec
Twitter		100%	4 sec
XMPP Event		100%	1 sec

Table 12 - Notification Results

The results in Table 12 show the performance and reliability of the notification mechanism. The average time of SMS and Twitter plugins depends of external service providers. The time was measured using a chronometer from the publish moment to the end of notification.

5.2 Storage

The users may define at what rate to upload sensor data, this selection is made in the sensor device, either by sending data at given rate or when an interesting event happens. The X represents the time needed to consume 1GB of space, for 12 different upload rates. A record is a sensor parameter value in database, as example, if we have 100 sensors in the system uploading 1000 records per minute (e.g., 100 sensors x 10 parameters), the database size will increase 1GB per week.

Time to fit one Gigabyte					
Upload Interval	Records		Days		Years
Second	1		126,928792		0,34775012
	10		12,6928792		0,03477501
	100		1,26928792		0,00347750
	1000		0,12692879		0,00034775
Minute	1		7615,72754		20,8650070
	10		761,572754		2,08650070
	100		76,1572754		0,20865007
	1000		7,61572754		0,02086501
Hour	1		456943,653		1251,90042
	10		45694,3653		125,190042
	100		4569,43653		12,5190042
	1000		456,943653		1,25190042

Table 13 - Upload Rates & Space Consumption

The database *DATA_VALUE* field in *DATA* table is of type *VARCHAR*, allowing storing sensor data up to 2000 characters. At the time of this test, we had 279.521 records in database, with total size of 26.1MiB (megabibytes), which gives an average of 98 bytes per record. The values in Table 13 are calculated with the calculated average record size in our database.

6 Conclusion & Future Work

6.1 Future Work

Security issues

Despite all security mechanism that the *PHP* framework has built-in and the security option provides by *SOX* library, attacks could be made. As example, the *Arduino* used in the uses cases, and is not capable of handling SSL connections, making possible to catch the sensor upload key in *HTTP POST* via packet sniffing. The actionchecker plugins that use the "*system()*" call and does not verifies the input string, are an flaw and may be exploited by intruders using commands injection, and then calling any command on the server.

Another security aspect is the definition of policies that creates loops; this problem is not as critical as the previous but can consume system resources unnecessarily. As example if a policy is defined as:

```
IF sensorX.temperature >0  THEN  PUBLISH  sensorX.temperature = 100
```

These situations may be difficult to detect and prevent, but in the next platform improvements this security problem should be taken into account.

The platform stores the user credentials in database, to connect with *XMPP* server and to execute operations. This represent another problem, if any intruder have access to the database, it is possible to decrypt the users password. To avoid this, the platform should implement more secure mechanisms like *oAuth*[23].

[23] http://oauth.net

Collection of nodes

Hierarchies are great for structuring and organizing elements, the *XMPP* server that we used does not have full support for the node collection extension. This extension permits hierarchies, where parent nodes are called collections nodes, and the children elements are called leaf nodes. With support for this extension in the *XMPP* server, our system could support hierarchy with a few changes in the web application and also on services.

Charts

Would be useful to compare different periods of time, as example, compare current month with the same month of previous year, as well any other interesting period of time. Also useful would be to have chart support for non numerical sensor parameters, this is, if a sensor reports "on" and "off" as values, the chart would have the possibility to map these string values as a numeric value defined by users. In this case a user could set "on" to be mapped as 1 and "off" as zero, being possible the representation of the sensor parameter in charts.

To create more interactivity with the users, the charts should allow users to define formulas to be represented on chart. As example, if we have the water consumption in a chart by liter, the user could add a new chart element to represent the cost of the water consumption.

Custom user panels would be interesting to users manage how they want to see the sensor charts and actuators control panels. As example the user could have more than one customized page (with chart, sensor and actuator widgets). The platform currently does not support user preferences in chart visualization (max, min, fill, etc), making the users reconfigure every time they access the chart page.

Social sharing

To add more “life” to the platform, would be interesting to add a public wall page where users could see the last events (new sensors added, new networks, etc). As example, when a user added a new sensor, the public wall would show that information along with sensor pictures, and everyone could comment and discuss.

Another possibility is to link the platform with social networks like *Facebook*. Then for example, the user could post automatically a chart of user weight over a month measured by a balance chair.

Policies

Another aspect is that policies are currently defined for a set of specific variables, but would be interesting to have policy templates, where the user could specify the sensors where the policy would apply.

6.2 Concluding Remarks

In this work, we presented a platform for real-time sensor data collection, visualization, and sharing. We demonstrate how we used a scalable, flexible and extensible real-time middleware to transport sensor data with security.

Also, we demonstrate with use cases that our system is useful for many applications, such as, data collecting data for statistics, data visualization and actuation based on policies.

In the practical use cases we demonstrated with a simple policy, in the mail experience, the platform potential in the real world. In the refrigerator use case we demonstrated that the platform is suitable for data collection for statistics and data analysis. With the wine experience, we give the first steps in building a wine monitoring probe that with more improvements can be used to collect data for analysis, used for actuation and for wine production control.

Bibliography

[1] Hichem Kenniche and Vlady Ravelomananana, "Random Geometric Graphs as model of Wireless Sensor Networks," *Computer and Automation Engineering (ICCAE), 2010 The 2nd International Conference on*, vol. 4, pp. 103-107, February 2010.

[2] James Truchard. (2008, Nov.) EETimes. [Online]. http://www.eetimes.com/design/industrial-control/4008153/In-2028-sensors-are-everywhere

[3] ThomasNet. ThomasNet. [Online]. http://www.thomasnet.com/about/sensors-73740607.html

[4] Wikimedia Foundation. (2010, November) Wikipedia - Transducer. [Online]. http://en.wikipedia.org/wiki/Transducer

[5] Frost & Sullivan. (2008, January) Frost & Sullivan. [Online]. http://www.frost.com/prod/servlet/market-insight-top.pag?docid=118964127

[6] Wire Service. (2007, September) SmartHouse. [Online]. http://www.smarthouse.com.au/Automation/General/W3G7L2H2

[7] Boyd Fletcher, "XMPP & Cross Domain Collaborative Information Environment (CDCIE) Overview For Net-Ready Sensors Summer Worksho," in *Net-Ready Sensors: The Way Forward*.

[8] Sensor Andrew. [Online]. http://sensor.andrew.cmu.edu/about/

[9] Robert F. Dickerson et al., "MetroNet: Case Study for Collaborative Data Sharing on the World Wide Web," in *2008 International Conference on Information Processing in Sensor Networks (ipsn 2008)*, 2008, pp. 557-558.

[10] B Maryam Elahi, Kay Romer, Benedikt Ostermaier, Michael Fahrmair, and Wolfgang Kellerer, "Sensor ranking: A primitive for efficient content-based sensor search," in *Proceedings of the 2009 International Conference on*

Information Processing in Sensor Networks, Washington, 2009, pp. 217-228.

[11] Mohammad Hammoudeh, Robert Newman, Sarah Mount, and Christopher Dennett, "A combined inductive and deductive sense data extraction and visualisation service," in *Proceedings of the 2009 international conference on Pervasive services*, London, 2009, pp. 159-168.

[12] Admilson Ribeiro, Fabio Silva, Lilian Freitas, Joao Costa, and Carlos Frances, "SensorBus: a middleware model for wireless sensor networks," in *Proceedings of the 3rd international IFIP/ACM Latin American conference on Networking*, Cali, Columbia, 2005, pp. 1-9.

[13] Ankit Tiwari, Prasanna Ballal, and Frank L. Lewis, "Energy-efficient wireless sensor network design and implementation for condition-based maintenance," *ACM Transactions on Sensor Networks*, vol. 3, no. 1, 2007.

[14] Lian Xing Zhang, "An Efficient Energy Adaptive Clustering LEACH in Wireless Sensor Network," *Key Engineering Materials*, vol. Advanced Measurement and Test X, pp. 510-515, 2010.

[15] Qinghua Wang, "Paradigms in the Research Community of Wireless Sensor Networks," Mid Sweden University, Sundsvall,.

[16] Muhammad Rashid and Mutarraf Mumtaz, "Remote Surveillance and Measurement," Halmstad University, Halmstad, Master Thesis IDE0836, October 30, 2008.

[17] Eduardo Souto et al., "Mires: a publish/subscribe middleware for sensor networks," *Personal and Ubiquitous Computing*, vol. 10, no. 1, pp. 37-44, December 2005.

[18] Diane Cook and Sajal Das, *Smart Environments: Technology, Protocols and Applications (Wiley Series on Parallel and Distributed Computin.*: Wiley-Interscience, September 2004.

[19] G. Michael Youngblood, Edwin O. Heierman, Lawrence B. Holder, and Diane J. Cook, "Automation intelligence for the smart environment," in

International Joint Conference On Artificial Intelligence, Edinburgh, 2005, pp. 1513-1514.

[20] Sajal Das and Diane Cook, "Designing and Modeling Smart Environments," in *Proceedings of the 2006 International Symposium on on World of Wireless, Mobile and Multimedia Networks*, Washington, DC, USA, 2006, pp. 490-494.

[21] Anthony Rowe et al., "Sensor Andrew: Large-Scale Campus-Wide," Carnegie Mellon University, Pittsburgh, Technical Report CMU-ECE-TR-08-11, 2008.

[22] Wikimedia Foundation. (2010, August) Wikipedia - GLib. [Online]. http://en.wikipedia.org/wiki/GLib

[23] Kathryn Barrett. (2009, May) O'Reilly. [Online]. http://fyi.oreilly.com/2009/05/what-can-you-do-with-xmpp.html

[24] (2010, January) XMPP Standards Foundation. [Online]. http://xmpp.org/about-xmpp/

[25] Wikimedia Foundation. (2010, October) Wikipedia - Extensible Messaging and Presence Protocol. [Online]. http://en.wikipedia.org/wiki/Extensible_Messaging_and_Presence_Protocol

[26] XMPP Software Foundation. (2003, September) Jabber Instant Messaging User Base Surpasses ICQ. [Online]. http://xmpp.org/xsf/press/2003-09-22.shtml

[27] Boyd Fletcher. (2006, August) XMPP & Cross Domain Collaborative Information Environment. PowerPoint Slides.

[28] Isode. (2007, July) Isode's Presence, Real Time Messaging and XMPP Strateg. Whitepaper.

[29] XMPP Software Foundation. (2010, August) XMPP Technologies Overview. [Online]. http://xmpp.org/about-xmpp/technology-overview/

[30] Peter Saint-Andre, Kevin Smith, and Remko Tronçon, *XMPP: The Definitive*

Guide.: O'Reilly Media, 2009.

[31] Iain Sadawo Shigeoka, *Instant messaging in Java: the Jabber protocols*. San Diego, CA: Manning Publications C, 2002.

[32] Wikimedia Foundation. (2010, September) Wikipedia - Publish/subscribe. [Online]. http://en.wikipedia.org/wiki/Publish/subscribe

[33] Umar Farooq, Eric Walter Parsons, and Shikharesh Majumdar, "Performance of publish/subscribe middleware in mobile wireless networks," *ACM SIGSOFT Software Engineering Notes*, vol. 29, no. 1, pp. 278-289, January 2004.

[34] Patrick Thomas Eugster, Pascal A Felber, Rachid Guerraoui, and Anne-Marie Kermarrec, "The many faces of publish/subscribe," *ACM Computing Surveys (CSUR)*, vol. 35, no. 2, pp. 114-131, June 2003.

[35] XMPP Standards Foundation. (2010, july) XEP-0060: Publish-Subscribe. [Online]. http://xmpp.org/extensions/xep-0060.html

[36] (2010, September) XEP-0248: PubSub Collection Nodes. [Online]. http://xmpp.org/extensions/xep-0248.html

[37] Wikimedia Foundation. (2010, October) Wikipedia - Twitter. [Online]. http://en.wikipedia.org/wiki/Twitter

[38] Wikimedia Foundation. (2010, October) Wikipedia - Photoresistor. [Online]. http://en.wikipedia.org/wiki/Photoresistor

[39] Wikimedia Foundation. (2010, October) Wikipedia - Thermistor. [Online]. http://en.wikipedia.org/wiki/Thermistor

[40] HANWEI ELETRONICS CO.,LTD. TECHNICAL DATA MQ-3 GAS SENSOR. Datasheet.

[41] Hanwei Electronics. MG811 CO2 Sensor. Datasheet.

[42] National Semiconductor. LM34 Precision Fahrenheit Temperature Sensors. Datasheet.

[43] Alessandro Vermeulen. (2008, November) Alessandro Vermeulen Blog. [Online]. http://alessandrovermeulen.me/2009/07/12/extending-the-v-in-mvc-revisited/

[44] Vitaly Friedman. (2007, October) Smashing Magazine. [Online]. http://www.smashingmagazine.com/2007/10/18/charts-and-graphs-modern-solutions/

[45] Jacob Gube. (2009, April) Six Revisions. [Online]. http://sixrevisions.com/flashactionscript/10-useful-flash-components-for-graphing-data/

[46] Wikimedia Foundation. (2010, November) Wikipedia - Google Maps. [Online]. en.wikipedia.org/wiki/Google_Maps#Google_Maps_API

[47] jQuery. (2010) jQuery. [Online]. http://jquery.com/

[48] Wikimedia Foundation. (2010, November) Wikipedia - Ajax. [Online]. http://en.wikipedia.org/wiki/Ajax_(programming)

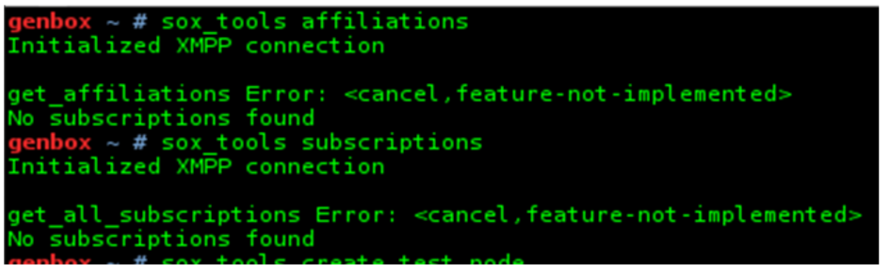
[49] json.org. JSON. [Online]. http://www.json.org/

Appendices

In this section we present the comparison between components which we selected to integrate in our system. Starting by comparing the XMPP server, then the PHP framework, then the chart component and finally we give a brief introduction in other technologies we used.

XMPP Server with Pub-Sub support

In Table 14 is compared three servers with support for Pub-Sub, all of them support the basic functionalities of Pub-Sub extension (XEP-060). Also they do not have implemented all protocol features. We started using *Openfire*[24] server, and after we tested *Ejabberd*[25] and *Tigase*[26] XMPP servers. We tested each server for the basic PubSub features (node management, subscriptions, affiliations, publishing, etc), and we found Openfire as the one with more support for PubSub features. We selected Openfire as our XMPP server for that reason but also for being the most easy to use and configure.



```
genbox ~ # sox_tools affiliations
Initialized XMPP connection

get_affiliations Error: <cancel,feature-not-implemented>
No subscriptions found
genbox ~ # sox_tools subscriptions
Initialized XMPP connection

get_all_subscriptions Error: <cancel,feature-not-implemented>
No subscriptions found
genbox ~ # sox_tools create test node
```

Figure 71 - Ejabberd Pub-Sub features not implemented

---

[24] http://www.igniterealtime.org/projects/openfire/
[25] http://www.ejabberd.im
[26] http://www.tigase.org

	Openfire	**Ejabberd**	**Tigase**
Pros	Web interface	Very scalable	Multi platform Installer
	Commercial Support	Live console	Java
	Easy configuration	Erlang	
	Java		
	Popular		
PubSub support			
Publish	Yes	Yes	Yes
Subscribe	Yes	Yes	Yes
unsubscribe	Yes	Yes	Yes
Create node	Yes	Yes	Yes
Delete node	Yes	Yes	Yes
Retrieve subscriptions	Yes	Not implemented	Not implemented
Retrieve affiliations	Yes	Not implemented	Not implemented
Retrieve last item	Yes	Not implemented	Not implemented
Add member	Yes	Yes	Yes
Node collections	Not implemented	Yes	Partially implemented

Table 14 - XMPP servers comparasion

PHP Framework

We had previous experience with PHP in developing web applications from scratch, but this was time consuming and slow to implement (eg: handle form validation, url redirection, security). In order to not repeat the same work we did in previous web applications, we decided to read about existing PHP frameworks.

Criteria to select a PHP framework were:

- Use of MVC
- Rapid development
- Small learning curve
- Good documentation

Advantages of using a PHP framework:

Time saving: development is faster, the amount of saved time can be up to 50% in most cases.

Reuse of code: many websites share the same features, these features are included.

Community assistance: there is a large community around a framework, which helps a lot in many situations (bugs, problems, getting help).

Libraries: all frameworks comes with a set of helpers/libraries, for example url, session, form validation, routing.

Disadvantages:

Learning curve: the time to learn to use a framework, some frameworks could take more time.

Memory foot print: Some frameworks have many associated libraries, making it slow.

There are various PHP Frameworks available today, *CodeIgniter*[27], *Cake PHP*[28], *Zend*[29], *Symphony*[30], each has a different coding convention. Most comparison tables only compare the features like, Ajax, Database, security, templating, caching, etc. The most popular frameworks are similar in the features, but only reading from user experience (user comments and posts), we could get some information about the learning curve, and other aspects.

CodeIgniter	**Symphony**	**Zend**
Ease-of-use	Used in enterprise world	Very popular
Lightweight	Code generation	Set of libraries
Performance and speed	Complete	Commercial support
Very good documentation	Mature	Large community

Table 15 - PHP Framework Comparison

We installed Symphony, Zend and CodeIgniter, from these three frameworks we found CodeIgniter simpler, with good documentation and with the smallest learning curve comparing with the other two. We made this choice based on our user experience, and in the small learning curve necessary and the ease-of-use.

Model-View-Controller System Extremely Light Weight PHP 4 Compatible Full Featured database classes Active Record Database Support Form and Data Validation Security and XSS Filtering Session Management	Email Sending Class File Uploading Class Image Library FTP Class Localization Pagination Data Encryption Benchmarking	Full Page Caching Error Logging Application Profiling Scaffolding Calendaring Class User Agent Class Zip Encoding Class Template Engine Class

Table 16 - CodeIgniter Features

[27] http://codeigniter.com
[28] http://cakephp.org
[29] http://www.zend.com
[30] http://www.symfony-project.org

CondeIgniter basically consists in two main folders, the system folder, that contains all framework files, and the application folder, like the name says, is where the applications are built.

Short description of application folder:

config – all configuration files are here, such as database connection settings, email, url settings.

controllers - this is where the program logic is programmed.

errors - Just defines what the various error pages are like, such as 404 error pages and database error pages.

helpers – collections of helper functions.

language – Used for dealing with more than one language.

libraries – Groups of related functions, such as charting component.

models – Used to represent objects, do all the interaction with the database.

Views – Where all html files go.

Layouts hack

During the development, we found a modification to select different layouts, as author explains "A layout is the representation of data in a specific format. So the layout of a page can be either, xhtml, RSS, PDF, PPT or any other data format" [43]. With the ability of selecting different layouts, only by adding the extension to the function, we increased the flexibility of our web application without changing controller's code.

Chart Component

In order to display data in charts, we searched for chart components to generate charts for our sensor data.

Our criteria to select a chart component were:

- Interactive (not a static image)
- Various types of charts (line, bar, pie)
- Open source
- PHP support

Most of charting components in web applications use CSS, JavaScript or Adobe Flash technologies. From these three technologies, the Flash charts were the ones with more quality (graphics and interactivity) and the ones with more chart types. There is many chart options around [44,45], in the end we selected three options to compare: *OpenFlash Chart 2*[31], *XML/SWF Charts*[32] and *FusionCharts*[33].

Open Flash Chart 2	**XML/SWF Charts**	**FusionCharts**
Open source	Free (with restrictions)	Commercial product
Json format	Xml format	High quality charts
Codeigniter plugin	Technical support	High flexibility
Popular		Technical support

Table 17- Chart component comparison

From these three options, FusionCharts was the better one, charts with more quality, high flexibility and many types of charts. Due to license restrictions in XML/SWF and FusionCharts, the Open Flash Charts 2 (OFC2) option gives more freedom in redistributing the platform without special attention to licenses. Besides being open source, the OFC2 was selected because it is popular and has many libraries for generating the charts.

[31] http://teethgrinder.co.uk/open-flash-chart-2
[32] http://www.maani.us/xml_charts
[33] http://www.fusioncharts.com

Chart types	Features	Libraries
Line Charts Bar Charts Horizontal Bar Chart Stacked Bar Chart Candle Chart Area Charts Pie Charts Scatter Charts Radar Charts	Tooltips Re-size charts Missing data Save charts as image	PHP Ruby Java .Net C Perl Python Coldfusion Google WebToolkit Smalltalk Pentaho

Table 18 - OFC2 features

Brief overview of *Open Flash Chart 2* components:

> **open-flash-chart.swf**: is a *SWF object*, wich receives a data file with a special format, and then renders into a chart.
>
> **Server library:** a server library is used to create the accepted format by SWF object.

The flash object is loaded by the client web browser. When the user makes a request to server, for example, update the chart. The server has the responsibility to get the data from database and then create the chart configuration file and send back to client browser. Finally when chart configuration is received, the flash object renders it.

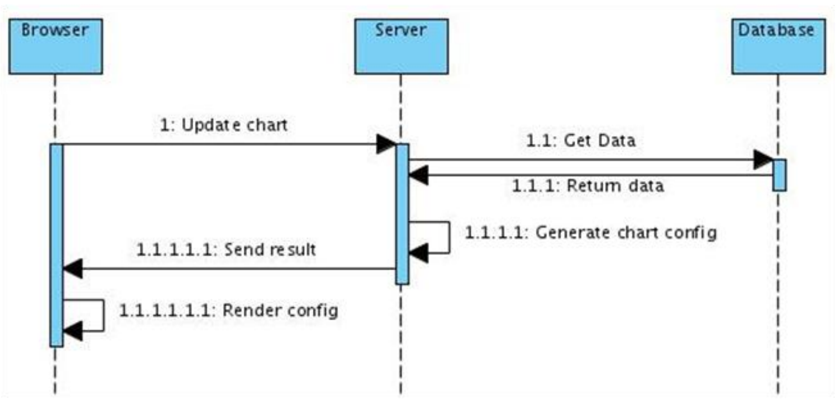


Figure 72 - Updating chart

Mapping Service

Google Maps is a web mapping service application and technology provided by *Google*. This technology enable us to navigate and search through the map of world, using different map views, and allowing to mark points of interest, which can be a single point, a line or a more complex shape.

Google Maps API[34] enables developers to integrate Google Maps into their web applications with their own data points of interest. Currently is a free service for non-commercial products [46].

The idea to use this popular service in our project was to represent geographically the networks and sensors.

- Represent networks
- Represent networks sensors
- Enable geographical search
- Show information about the networks
- Add networks and sensors in a geographic location

[34] http://code.google.com/apis/maps/index.html

Jquery

jQuery[35] is a fast and concise JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development [47].

Today any rich web application contains JavaScript. This technology enables most of the interactions between users and the web applications. jQuery is an easy to use and popular JavaScript library that we selected to use in our platform development.

Advantages of using jQuery:

- Event handling
- Animations
- Ajax calls (posts and gets)
- JSON support
- Fast development
- Very small
- Easy to use
- Popular

[35] http://jquery.com

AJAX

AJAX short for “Asynchronous JavaScript and XML”, is a technology based on the client side, allowing the construction of interactive web applications, with this technology it is possible to receive information from the server asynchronously without interfering with visual or behavior of the page web visualization [48].

Advantages of using this technology:

- refreshed dynamically
- refresh parts of web page instead of everything
- less data transmitted
- less overload on servers

JSON

JSON[36] or *JavaScript Object Notation* is a lightweight data format, easy for humans to read and write, is often used in data serialization and data transmission on networks, the most common application is in Ajax programming [49].

```
{
   "name": "John",
   "age": 25,
   "address": { "city": "New York", "state": "NY"}
}
```

This format makes concurrency with XML, but JSON is very used in web development mostly to transport data. Here are some advantages of using JSON over XML.

- JSON is simpler than XML
- JSON is faster
- Creates smaller files

[36] http://www.json.org