

# Chapter 7

## A Scalable Sensor Middleware for Social End-User Programming

Salvador Faria and Vassilis Kostakos

**Abstract** A substantial amount of research has focused on developing sensor middleware targeted at various research communities such as networking and context awareness. This chapter presents SAWA, a sensor middleware based on Sensor Andrew aimed at social end-user programming. SAWA is designed to collect, present, share, and act on sensor data. First, it allows novice users to deploy a multitude of both physical and virtual sensors and actuators (e.g. temperature, light, unread email count, friend's status on Facebook, SMS, Tweet) and to aggregate this data in a central server. Users are able to access an online portal to visualize and explore their recorded data. In addition, they can request and share access to other users' sensor streams. Finally, they can create actions that are driven by sensor data—both physical and virtual, and both their own or any of their friends'. In addition to describing SAWA's architecture, this chapter presents case studies where this middleware was used. It is shown that in addition to being robust and scalable, SAWA opens up a series of new applications by allowing users to program sensors and actuators in a shared social environment.

### 7.1 Introduction

Sensors, or transducers, are devices that measure a physical quantity and convert it into a signal that can be read by an observer or by an instrument [8]. With big advances in processor technologies and wireless communications, sensor networks and home automation systems have been growing in recent years [5]. In these systems, many sensors distributed in an area collect various physical data such as temperature, humidity, motion, and light [4].

---

S. Faria (✉)

Madeira Interactive Technologies Institute, University of Madeira, Madeira, Portugal  
e-mail: [salvador.faria@m-iti.org](mailto:salvador.faria@m-iti.org)

V. Kostakos

Department of Computer Science and Engineering, University of Oulu, Oulu, Finland  
e-mail: [vassilis@ee.oulu.fi](mailto:vassilis@ee.oulu.fi)

Researchers have previously developed many sensor-networking applications and middleware, but they are typically isolated, small-scale and short-lived experiments [10]. This highlights the necessity for interconnecting various sensors, both physical and virtual, in a scalable manner and using open protocols and technology.

This need is further exaggerated because with digital technology information available anywhere and anytime about the physical world has a potential value, and 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 [2]. With increasing penetration of embedded sensors in networked devices, such as GPS sensor in mobile phones, it is now possible to create applications that take into account the current state of the real world [3]. 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.

Furthermore, collected data from sensors is normally difficult 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 to manage the collection of data [2]. Therefore, the sensing platform should help scientists to leverage computational power to simulate, visualize, manipulate, predict and gain intuition about monitored phenomenon [6]. Visual representation of data can be done in many ways, the most common are evolution 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 intention is to build a system to facilitate the use and sharing of sensor data by providing a set of functionalities and a uniform access to these devices. The system allows users to network any sensing device by “registering” sensors and actuators, enables users to monitor their devices and define who has access to the recorded information; present data to users in form of charts thus giving a better interpretation and meaning to data; longitudinally record sensor data enabling monitoring; and allow for user policies to define sensor conditions and respective actions.

## 7.2 Related Work

There are different methods for retrieving data from sensors, continuous, event-oriented, query-oriented and hybrid. The method to use depends on network design and resources constraints, like power and limited communication. In continuous methods, 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, applications are responsible for defining which events are of interest and then querying sensors (e.g. “SELECT sensors WHERE temperature > 40 AND C02 > 15”). Finally the hybrid method is when more than one of the other three methods are used [9].

Online sensor-sharing services like Pachube and SensorPedia have the objective of applying the social networking 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. Recently Patchube started to offer triggers, allowing users to have URL event notification and experimental SMS notifications. The Patchube web service has the major advantages of an easy to use interface, a big community and a set of third party applications. A drawback is that Patchube relies on Representational State Transfer (REST) paradigm and not on a real-time transfer paradigm, limited support for actuators and basic triggers.

Our system is similar to Patchube, in the way it has similar objectives, allowing users to easily share sensor data and visualize them. Besides these objectives, our system allows a high scalability, extensibility, security and privacy in the middleware. Our system allows multiple entities to subscribe to sensor data in a push method, define user access type and groups, encryption and other relevant XMPP Pub-Sub features. It also provides a web interface, where 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 of our application it is possible to directly call commands in the actuators and create advanced policies, which can use different types of rules and actions.

### 7.2.1 *Sensor Andrew*

Sensor Andrew is a scalable campus-wide sensor middleware, developed with the objectives of supporting ubiquitous large-scale monitoring and infrastructure control [10]. It was developed by Carnegie Mellon University in order to integrate multiple systems, and designed to be extensible, easy to use, and secure while maintaining privacy. Its reliance on XMPP allows application developers to be able to transmit sensor data with no need to re-invent lower-level interfaces.

Sensor Andrew's objectives made it ideal for our needs:

- 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.

A drawback of Sensor Andrew, however, is that it remains an expert tool, with no visual interface and all configurations taking place via scripts and options files in the command prompt. Therefore, a substantial amount of our work focused on adding an interface and visualization component to Sensor Andrew.

### 7.2.2 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 [1].

XMPP is used by many types of applications, instant messaging, multi-party chat, voice and video calls, collaborations, lightweight middleware and content dissemination [7], also expanded to the domain of message-oriented middleware. Built to be extendible, the protocol has been extended with many features, including the Publisher-Subscriber paradigm. According to [11], by 2003 it was estimated that software using XMPP was installed in hundreds of server across the Internet and was used by ten millions of people. Most notably, services like Google chat and Facebook chat rely on this technology.

XMPP has more than 150 published extensions, 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.

## 7.3 System Overview

Our system makes it possible for users to network physical and virtual sensors, and create policies to act upon the values of the sensors. For instance, it is possible to create a policy to alert a person via SMS when a particular value, e.g. of gas concentration, is detected. The system has potential for real-time uses, for non real-time, and 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 use of television, computers, and lights turned on without any person present in the home.

Furthermore, using GPS technology it is possible to create policies to detect when a device enters or exits a geographical area. Considering a bus updating its GPS position every minute, a policy can be created to trigger a notification when a bus is detected in special area such as a bus stop. Given the built-in sharing capabilities of the system, the owner of the “bus sensor” (conceivably the bus operator) could choose to share the sensor readings with any member of the public. Subsequently, interested users could construct their own notification based on the bus position, as they see fit.

Finally, 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.

### 7.3.1 User Interface

Our system provides user with instructions on how to network and connect their sensors to our infrastructure. This is possible in a number of ways, both using XMPP and HTTP GET. Here we do not describe this part of the process, but rather focus on the web front of our application.

The main interface (Fig. 7.1) consists of 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 substantially change the menu or interface layout. From the first page the user is able to graphically browse her sensor networks and sensor networks that she has been granted access to. Sensor networks are groups of sensors, usually functioning as a group. Sensor networks are presented in the form of a mashup using Google maps. It is worth noting that sensor networks contain multiple sensors, and an individual sensor may contain multiple “variables” as in the case of an accelerometer that contains 3 variables for  $X$ ,  $Y$  and  $Z$ .

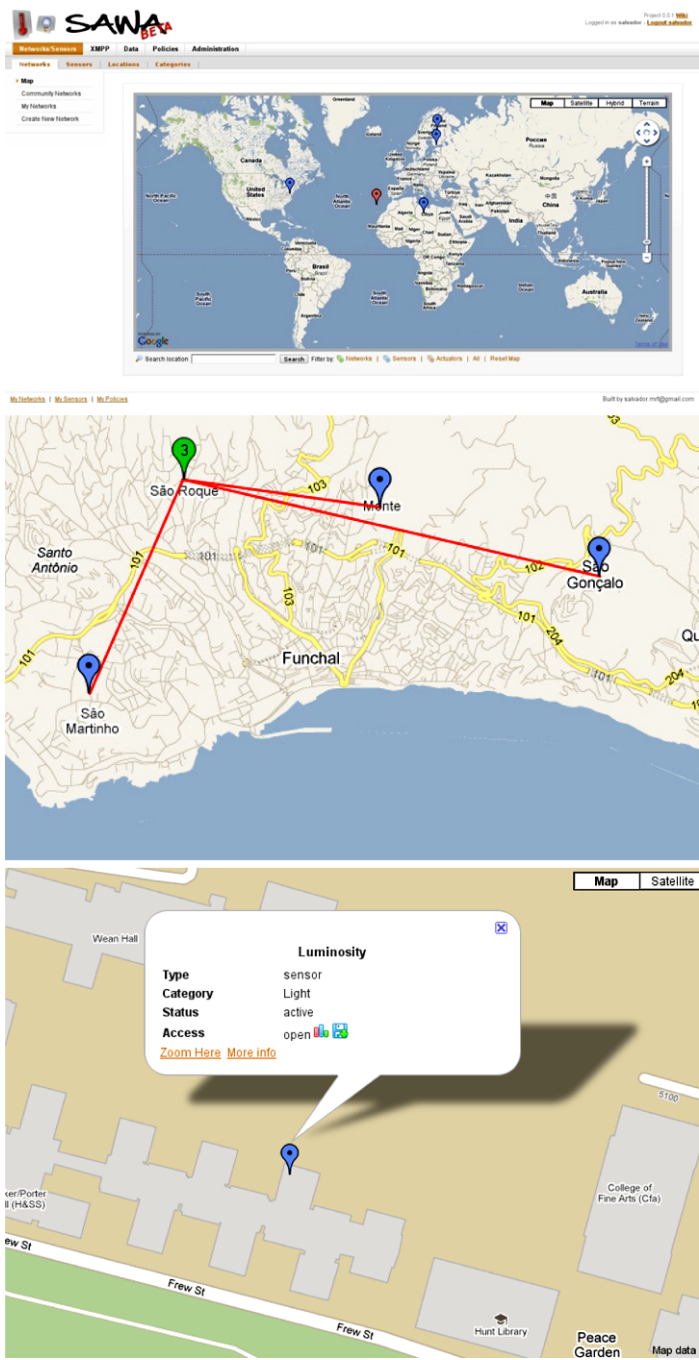
The user is also able to create a list of sensor networks they have access to, as shown in Fig. 7.2. Here, each sensor network has an ID, a name and address, a number of associated sensors, a control model (open allow access to anyone, whitelist gives access to authorized users), and an owner.

Each sensor network can be probed for further information. In this case, a variety of details are shown about the network, including possible screenshots uploaded by other users (see Fig. 7.3), a detailed list of the sensors in this network, as well as the type of each sensor and its current recording state.

More information for each sensor can be shown on a separate page (see Fig. 7.4). This detailed contains the basic sensor information, an image gallery and the sensor variables list. Certain sensors, such as an accelerometer, may have multiple “variables”: in the case of the accelerometer these are  $X$ ,  $Y$ , and  $Z$ . In other cases, such as in the figure below, it is convenient to create aggregations of sensors, such as a “refrigerator sensor”. In this case, we show a “sensor” that has a series of value relating to temperature, door status, gas levels, and whether any human is nearby.

In addition to sensors, sensor networks may also contain actuators. These are treated in a very similar way as sensors, except from the fact that they cannot record information and they have functions that can be called. Each actuator has a detailed page where all its attributes and functions are shown (see Fig. 7.5). In addition, the functions can be directly called for purposes of testing and daily usage. New functions can be added to actuators, but these functions are symbolic and must by interpreted appropriately by the actuator hardware. This most likely requires consultation of the hardware instructions and the device’s specifications.

Given a set of sensors and actuators, a user can specify a policy (see Fig. 7.6). Policies consist of sensor values (it is noted that these sensors may not be owned by the owner herself but may be shared or public), actions, and a notification interval. The notification interval is the least amount of between subsequent executions of the actions. This feature avoids the continuous notifications when a policy is constantly matched, i.e. we do not want to receive an SMS notification every second. A policy



**Fig. 7.1** Main Interface, showing an overview of available sensor networks (*top*), a summary of a particular network's structure (*middle*), and details about particular sensors (*bottom*)

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	

Fig. 7.2 Detail view of sensor networks

Network info

About Network

Name sal\_home

Owner salvador




Access open

Members 0

Location Estreito De Câmara, Portugal

Network Gallery

Upload image  No file chosen



[Remove](#)

[Remove](#)

[Remove](#)

Sensors and Actuators



















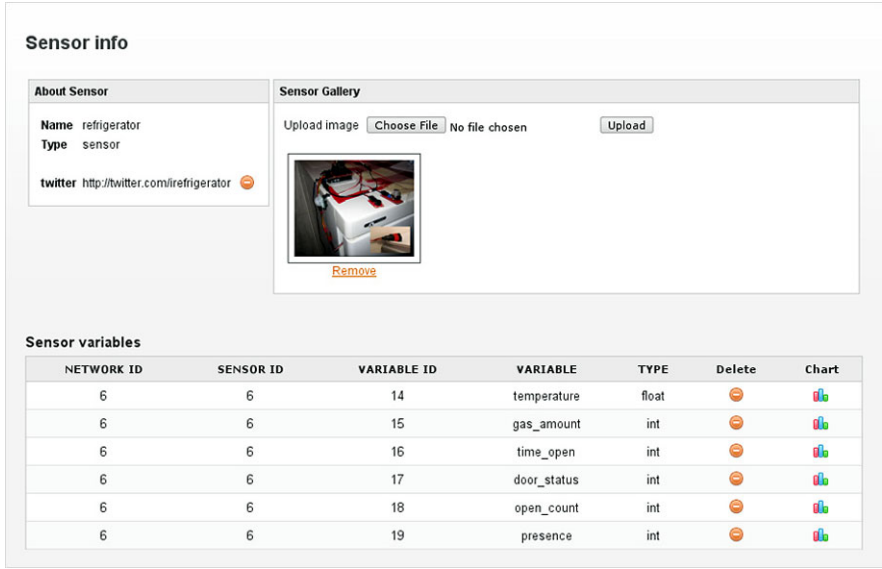
SENSOR ID	SENSOR NAME	TYPE	CATEGORY	USERNAME	Info	Record	Chart
6	 refrigerator	sensor	Light	salvador			
16	 led_red	actuator	Light	salvador		--	--
17	 speaker	actuator	Sound	salvador		--	--
18	 s_room	sensor	Temperature	salvador			
20	 power_outlet	actuator	Energy	salvador		--	--
24	 Luminosity	sensor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

Fig. 7.3 Detailed view of a particular sensor network. This includes a list of all the sensors in the network along with their current state

can have multiple rules which describe conditions under which policies should be fired. When a policy is fired, the actions are executed.

In Fig. 7.6 we created a policy named “gas\_alert”, the selected notification type is SMS, in the message field, and we use tags ({value}), to be replaced in the actual SMS. This policy contains only one rule, the defined rule uses gas\_value variable, the operator ‘>’ and the match value 100. The effect of this policy is to alert via SMS when there is a gas leak (gas\_value > 100).

It is noted that rules may also use dates and times, while for variables of type string the operators “is”, “is not”, “starts with”, “ends with”, “contains” and “does



**Fig. 7.4** The page showing detailed information about a particular sensor. In this case, the sensor has multiple variables

not contain” can be used. In addition, SQL-Like rules may specify date intervals during which a particular value may exist or not.

Finally, users may choose to enable recording on a particular sensor, and subsequently visualize the collected data (see Fig. 7.7). Multiple chart types are supported, and multiple variables can be combined in a single chart. The time span of their chart, as well as their granularity, can be dynamically changed.

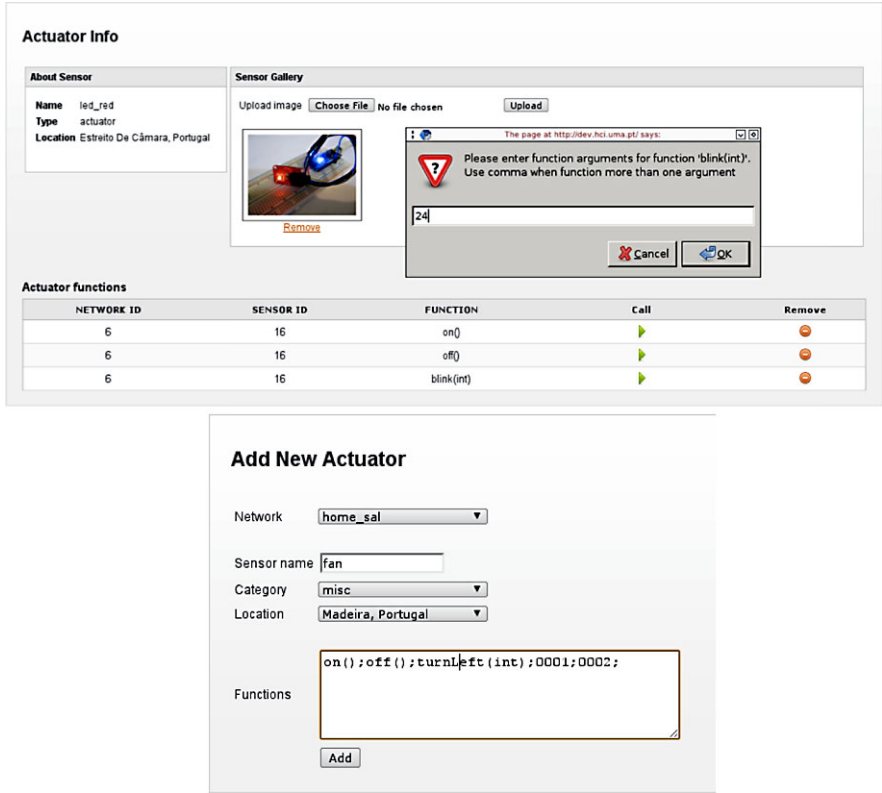
**7.3.2 High Level Architecture**

The system consists of a number of components (see Fig. 7.8):

- *Web application*—The component responsible for presenting all information to users, and enabling users to manage their devices, associated options, and sharing preferences.
- *Datarecorder*—The service responsible for storing sensor data.
- *Actionchecker*—The service responsible for matching sensor input to existing policies, and trigger actions when a policy matches.
- *Scheduler*—The service responsible for handling new recording requests and new policies, and coordinating the datarecorder and actionchecker services.

Our system can be divided in two parts: (i) the middleware, composed by the XMPP server with publisher-subscriber support, which is responsible for transport-





**Fig. 7.5** *Top*: Detailed information for a particular actuator. In this case the actuator is an LED. In addition to details about the actuator, its functions are also shown and they can be called. Furthermore, new functions can be added. *Bottom*: adding a new actuator allows for the definition of functions. These functions are symbolic, and must be interpreted appropriately by the actuator hardware

ing and handling all sensor data, and (ii) the other part which consists of the web application and services.

The architecture style of our system is a passive shared repository, allowing for a loose coupling between system components. All interactions are made through the 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 the figure shows, the services also use the repository to modify, read and create new entries. The datarecorder and actionchecker services 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.

Finally, the system allows HTTP POST requests to be used to update sensor values. This is a very flexible way of accepting sensor values, and makes it easier



Fig. 7.6 A policy to alert via SMS when there is a gas leak (gas\_value > 100)

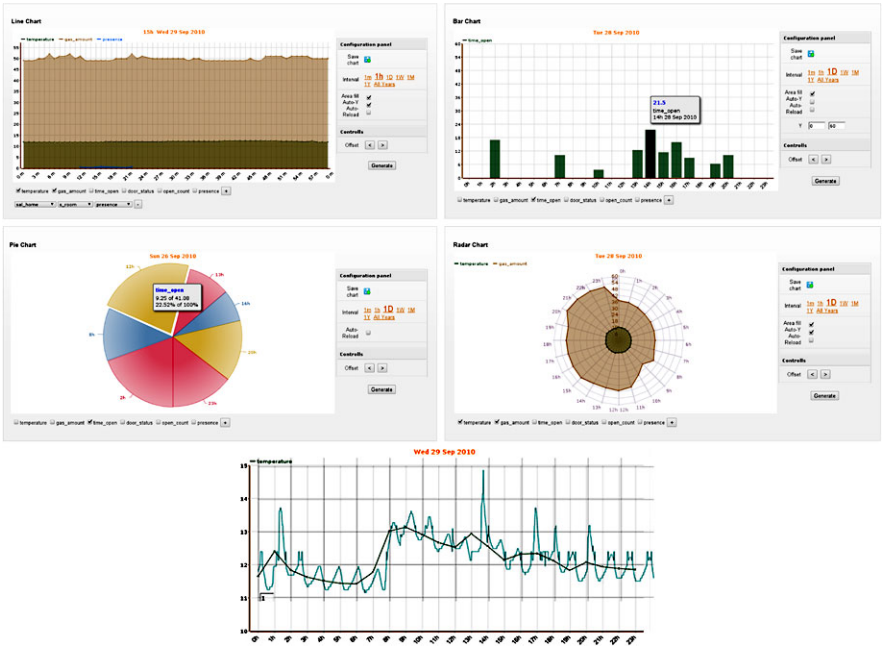
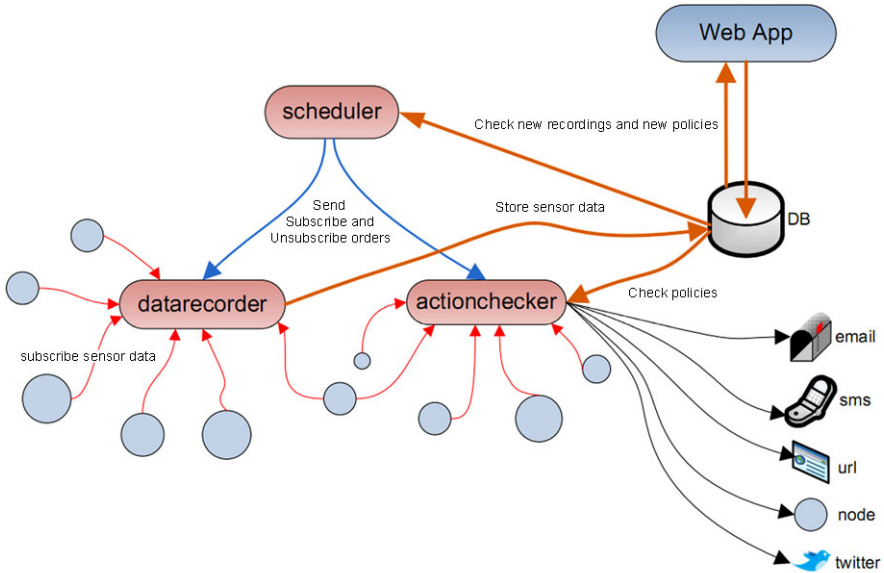


Fig. 7.7 Dynamically generated charts allow users to visualize historic data for recorded sensors

for our system to work with embedded systems that may be incapable of generating XMPP messages. In this case, an HTTP wrapper is used to accept values for a particular sensor: the sensor id, the value, and a pre-shared secret. The wrapper then interprets this input, identifies from which sensor it is generated, and generates an internal XMPP message to update the sensor value. This solution ensures that sensor values can be updated easily, while at the same time the authorization and security model of XMPP is maintained.



**Fig. 7.8** High-level architecture

### 7.3.3 Plugins

The advantages of using plug-ins are well known, they are easy to deploy, they are efficient, 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. The system provides a number of plugins that can be triggered as part of policies.

**Email.** Email is a communication method and is used in almost every notification system. The plugin allows static and dynamic text to be generated and sent to a recipient.

**URL.** Efficient notifications can take the form of URL calling, a simple and basic notification that can be very useful in communicating between two unknown systems. This plugin relies on the HTTP GET method to call a pre-defined URL with or without parameters. One way in which this plugin can help bring together diverse systems is, for example, by calling a foreign systems' API by accessing a REST API: <http://example.com/lights/set/33/off>.

**Functions.** This plug-in enables the publication of events using XMPP nodes. In particular, it allows the publication of commands to actuators, such as "activate", "turn\_off", "increase", or any pre-defined command. In sensor gateways, when a 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 commu-

nication protocol. Optionally the sensor gateway may report the new status of the actuator, such as “running” or “active”.

**SMS.** The plugin allows the transmission of arbitrary static and dynamic text to a mobile device capable of receiving SMSs. With the spread of mobile communications, SMS is a good mechanism to alert people with urgency, and to send sensor data from remote areas.

**Twitter.** Twitter is a social networking and micro blogging service that enables its users to send and read messages known as tweets (Wikimedia Foundation 2010). We created a plug-in for sending posts to this social network. Following the official tutorial we developed the twitter plug-in that uses HTTP GET to create a new twitter post.

## 7.4 Case Studies

### 7.4.1 *iMailbox*

The motivation for this use case was to build a system that notifies the user via SMS whenever new physical mail arrives in the post box. To implement this idea we used an Arduino board with a door sensor. The physical sensors detect the status of the door (open, closed), and also detects when a mail is added to the mailbox.

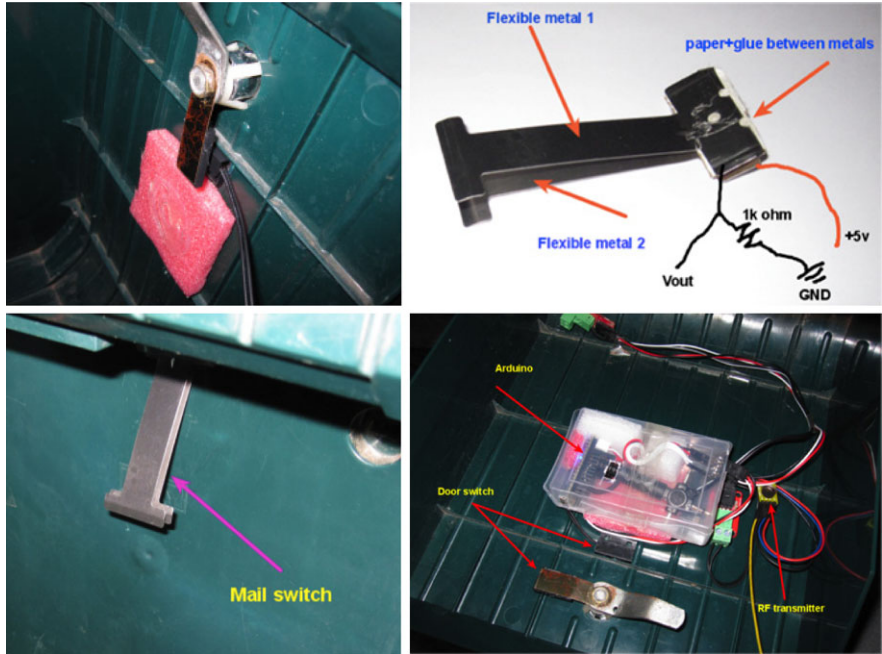
To detect when a mailbox door is opened, a cheap magnetic door switch was used. This sensor consists of two parts, the magnet and the magnet detector. The magnet was attached to the rotating lock mechanism and the magnetic sensor was glued to mail box, as shown in Fig. 7.9. To detect the presence of mail we used a sensor that acts like a button: when new mail is inserted the metal plates touch and close the circuit.

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 mailbox slit. In tests using physical envelopes the sensor detected the insertion of all mail.

Once the hardware was configured, we were able to use our online platform to achieve the desired notification behavior. In the web application we registered a “mail sensor”, and we registered two variables for it: *new\_mail* (set to 1 when new mail is present) and *door\_status* (set to 1 when the door is opened).

Using the appropriate network, sensor and variable IDs, our Arduino board transmitted values using HTTP POST to a server using a simple protocol as follows: *message\_number#sensor\_id#variable\_id#variable\_value*. The server re-broadcasts the received information using XMPP to our system.

Next, a policy was created such that whenever the variable *new\_mail* becomes “1” an SMS action is triggered with the message “New mail arrived!”. In addition, a notification was created such that whenever the mailbox door was opened a further SMS notification was sent (see Fig. 7.10).



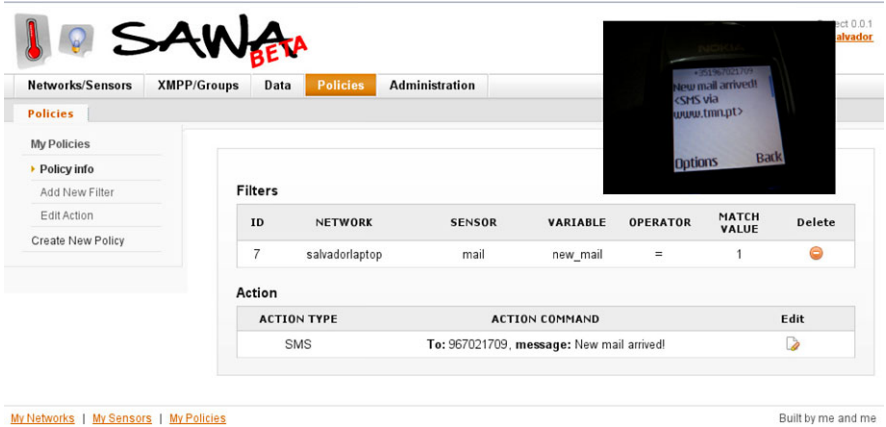
**Fig. 7.9** *Top left:* Magnetic door sensor. *Top right:* metal plates for detecting mail. *Bottom left:* metal plates installed in the mailbox. *Bottom right:* final installation

### 7.4.2 iRefrigerator

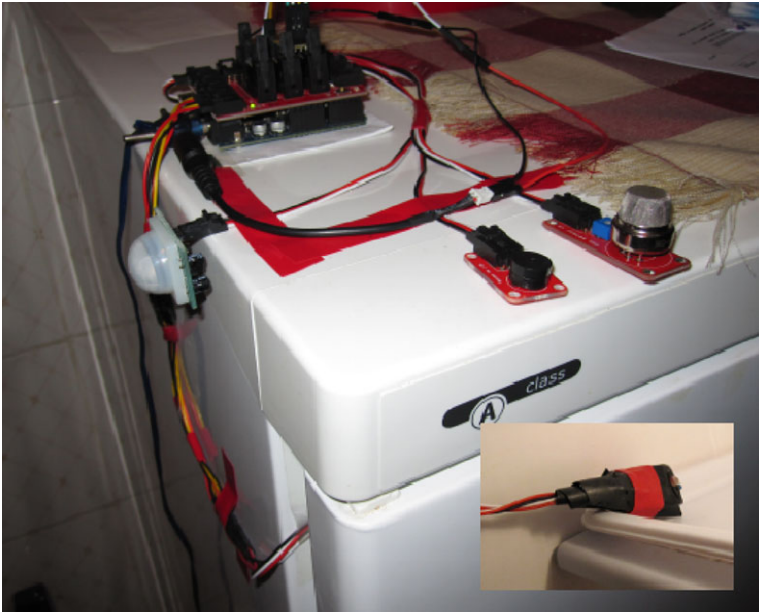
The motivation to monitor a refrigerator was to assess the impact of opening the door on energy consumption. To accomplish this, we used a temperature sensor, a photo resistor sensor, a gas sensor to detect butane gas leaks, and a presence sensor to monitor people’s presence in the kitchen (see Fig. 7.11).

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 photo resistor was used. 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 the light intensity sensor a thermoresistor was attached to measure temperature. The gas sensor, which is highly sensitive to gases, was used to monitor the presence of butane and propane gas in the kitchen. Finally, a Buzzer that emits a sound alert when the gas sensor detects high values was installed. Similar to the iMailbox, we used an Arduino board platform, a RF transmitter module, and the same protocol and procedure for sending sensor values.

The chart in Fig. 7.12 represents the data from four variables within one hour (10 h). The darkest brown line is the refrigerator temperature, the light brown represents the gas sensor data, and the line with magenta color represents humans’ presence in that hour. The blue line shows the duration for which the refrigerator

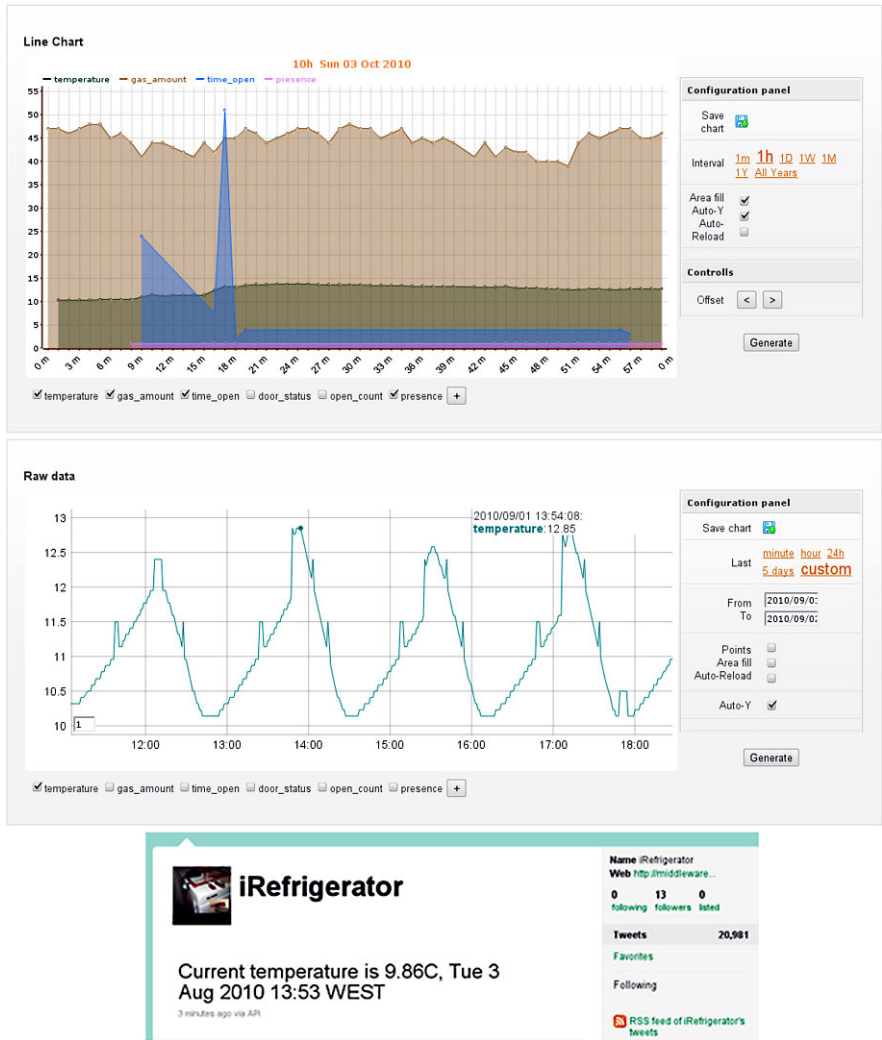


**Fig. 7.10** Policy information for the iMailbox case study (*inset*: a received SMS as a result of new physical mail arriving in the mailbox)



**Fig. 7.11** iRefrigerator sensors

door was opened, and it is noted that this line only contains 7 data points (ideally it should be rendered as a bar chart, but combining multiple chart types is not technically possible at this point). In addition, a Twitter update was generated once per hour to announce the iRefrigerator’s temperature.



**Fig. 7.12** *Top*: one-hour data recorded by the iRefrigerator (x-axis: time, y-axis: value). *Middle*: high granularity data for the refrigerator temperature (y-axis) over time (x-axis). *Bottom*: an automated twitter post every hour indicated the iRefrigerator’s temperature

## 7.5 Discussion and Conclusion

This chapter has presented a sensor middleware that allows users to network, record and share sensor data. The chapter has described the system itself, and has presented two case studies where the middleware was used. Both the iMailbox and iRefrigerator case studies exemplify the technical capabilities of our system, however we wish to point out the fact that they also exemplify the potential of the system’s sharing capabilities.



The key aspect of our system is the ability for users to share the sensors they own with other users, and the ability for users to create rules and actions based on sensors that they do not actually own. For instance, in the case of the iRefrigeratore, one member of the family could technically own the “refrigerator sensor”, but could choose to share that sensor with other members of the family. The rest of the family could make use of this sensor in different ways. One person may be interested in charting the use of the kitchen area during the day, while another person may want to receive an alert if the door has been opened for more than 300 seconds.

Similarly, in the case of the iMailbox, the owner may wish to share the sensor with her secretary such that she could pick up the mail whenever it arrives, and also share it with security such that they can monitor whether the door is opened outside working hours (indicating potential theft). In this case, the security themselves could decide for the most appropriate automated action rather than for the user creating an email or SMS alert intended for security. For instance, security personnel may have a physical buzzer in their control room which they can actuate should the iMailbox’s door be opened in the middle of the night—this is something that neither the user nor security can do on their own but is made possible by sharing access to sensors.

Another advantage of the system is that it is flexible enough to allow for virtual sensors to be utilized. Such virtual sensors may include calendar information, system information such as file space, unread emails, and so on. A particularly interesting type of sensor is GPS: we are currently working on a prototype system whereby a mobile phone constantly reports its GPS location as a “location-sensor”. The user is then able to create arbitrary rules online triggered by the phone’s location. In addition, the user may choose to share this sensor data with others. This would allow, for example, for parents to be notified whenever their child leaves work, whenever the husband is near a supermarket, or whenever two people are within certain distance of each other. In combination with the ability to post live updates on Twitter, the system provides a wide array of possibilities in terms of location sharing.

**Acknowledgements** This work is funded by the Portuguese Foundation for Science and Technology (FCT) grants CMU-PT/HuMach/0004/2008 (SINAIS) and CMU-PT/SE/0028/2008 (Web Security and Privacy).

## References

1. Barrett, K. May 2009. <http://fyi.oreilly.com/2009/05/what-can-you-do-with-xmpp.html>.
2. Dickerson, R. F., et al. (2008). MetroNet: case study for collaborative data sharing on the world wide web. In *2008 international conference on information processing in sensor networks (IPSN 2008)* (pp. 557–558).
3. Elahi, B. M., Romer, K., Ostermaier, B., Fahrmaier, M., & Kellerer W. (2009). Sensor ranking: A primitive for efficient content-based sensor search. In *Proceedings of the 2009 international conference on information processing in sensor networks*, Washington (pp. 217–228).
4. Fletcher, B. (2006). *XMPP & cross domain collaborative information environment*. Power-Point Slides. August 2006.
5. Newbury, N. (2008). <http://www.frost.com/prod/servlet/market-insight-top.pag?docid=118964127>. 22 January 2008.



6. Hammoudeh, M., Newman, R., Mount, S., & Dennett C. (2009). A combined inductive and deductive sense data extraction and visualisation service. In *Proceedings of the 2009 international conference on pervasive services*, London (pp. 159–168).
7. <http://xmpp.org/about-xmpp/>. January 2010.
8. Kenniche, H., & Ravelomananana, V. (2010). Random geometric graphs as model of wireless sensor networks. In *The 2nd international conference on computer and automation engineering (ICCAE)*, 6 February 2010 (pp. 103–107).
9. Ribeiro, A., Silva, F., Freitas, L., Costa, J., & Frances, C. (2005). SensorBus: a middleware model for wireless sensor networks. In *Proceedings of the 3rd international IFIP/ACM Latin American conference on networking*, Cali, Columbia (pp. 1–9).
10. Rowe, A., et al. (2008). *Sensor Andrew: Large-scale campus-wide* (Technical Report). Carnegie Mellon University, Pittsburgh.
11. XMPP Software Foundation. <http://xmpp.org/xsf/press/2003-09-22.shtml>. September 2003.