Sensor.Network: An Open Data Exchange for the Web of Things

Vipul Gupta, Arshan Poursohi, Poornaprajna Udupi

Sun Microsystems Laboratories, Menlo Park, California, USA
{*vipul.gupta, arshan.poursohi, poornaprajna.udupi*}*@sun.com*

***Abstract*—Tiny, wireless, sensors embedded in a large number of Internet-capable devices–smart phones, cameras, cars, toys, medical instruments, home appliances and energy meters–can generate an enormous volume of small bits of data such as temperature and humidity readings, GPS co-ordinates and energy usage. The real value of this data lies in its analysis, which can lead to significant insights and actions that enhance the health of our planet and its populations. We have developed a web-based infrastructure called Sensor.Network for storing, sharing, searching, visualizing and analyzing data from heterogeneous devices and facilitating easy interaction amongst devices and end users through an open, REST-based API.**

This demonstration starts off with a brief walkthrough of the Sensor.Network portal as we describe its data abstractions, APIs and different *views*, *e.g. dashboard view* for monitoring the health of different sensor datastreams and their access permissions and *map view* for geographical search. Next we show the platform's visualization capabilities, in particular *LivePlots*–interactive plots of different types which can be embedded into web-based applications hosted elsewhere and are rendered using the latest data from our service each time that external webpage is reloaded. We discuss data-centric collaborations and social interactions made possible by features such as tagging, searching and sharing with fine-grained privacy controls. We demonstrate *notifications* that are automatically generated when sensor data meeting user-specified criteria is inserted. These notifications allow the composition of "mashups" of heterogeneous sensors and actuators (*e.g.* a humidity sensor from one vendor controlling a sprinkler system from another). Finally, we present our early explorations into reprogramming and remote management of devices.

I. Introduction

Tiny wireless sensors promise to fuel a significant expansion of the Internet. Such sensors can be found in all kinds of devices, an increasing number of which are becoming Internet-capable—smart phones, cameras, cars, toys, medical instruments, home appliances, energy meters, traffic loops, *etc.* They are being adopted in several application domains including, but not limited to, intelligent agriculture, proactive health care, asset tracking, environmental monitoring, security surveillance, data center management and social networking. A large number of these applications require facilities for collecting, storing, searching, sharing and analyzing sensor data. There are several commercial offerings that attempt to address this need for specific application domains [1]–[5]. However, in all of these cases, there is a tight coupling between the devices and the back-end services and tools. It is not easy to interface devices from one manufacturer with the services and tools from another, thereby making them *closed* systems.

With Sensor.Network, our vision is to create a *data exchange* platform with an *openly published* Application Programming Interface (API) that eliminates this tight coupling and allows the composition of new services incorporating heterogenous sensors and actuators from different manufacturers and potentially owned by different entities. By storing, searching, sharing and analyzing sensor data from multiple disparate sources, Sensor.Network also enables investigation into correlations that would otherwise be missed. For example, the RunKeeper [6] iPhone application and its associated web portal use the GPS sensor built into the iPhone to help runners keep track of their runs. If this data were managed using an open data exchange, a runner could potentially correlate these measurements with data from temperature and humidity sensors in the area for greater insight when evaluating her own performance.[1] The next few sections present a high-level overview of Sensor.Network and a description of our proposed demonstration. Additional details on its design and architecture are available in [7].

II. Using Sensor.Network

This section briefly describes our data abstractions, Application Programming Interface (API) and a typical user interaction.

A. The Datastream Abstraction

At the core of the Sensor.Network architecture is the notion of a *datastream.* It refers to a time-series of sensor values that are sampled together. The datastream abstraction decouples the physical sensor from the high-level phenomenon of interest to the end user and the same datastream may be fed by different sensors at different times. Metadata associated with a datastream includes: name, description, tags, URI, category, location, sampling period, access permissions and the name, type and units for each sensor value.

[1]The iPhone does not have temperature or humidity sensors.

978-1-4244-5328-3/10/$26.00 ©2010 IEEE

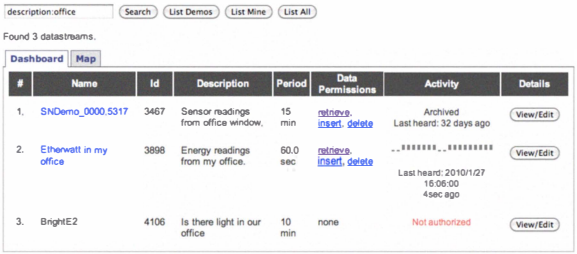

Figure 1. The dashboard view provides at-a-glance information, such as permission settings and recent activity for datastreams. We see three datastreams with "office" in their descriptions: the first one has no recent sample data, the second one has missed four readings in the last 20 minutes, and the user is not authorized to view data in the third.

Figure 2. The map view identifies the geographical location of a datastream. Clicking on a datastream's marker pops up a tabbed window containing different types of associated information. One of these tabs shows simple line plots for recent sensor data.

As an example, a datastream named "John's location" may have three values: latitude, longitude and altitude, all of type float, with the first two measured in degrees, with a range of -90 to +90 and -180 to +180, respectively and the last in meters. This datastream may be fed by a GPS sensor on a car navigator when John is driving and from a phone, capable of radio signal triangulation, when he is walking.

B. Application Programming Interface (API)

The Sensor.Network service is located at http://sensor.network.com and offers its users the ability to create, edit and delete datastreams, insert and retrieve sensor data and generate notifications and visualizations. Users can exercise flexible, fine-grained control over how their sensor data is shared (*e.g.*, read-only, low-fidelity) and with whom (owner-only, specific individuals). Sensor.Network enables collaborative classification (*e.g.*, with tagging), annotation, editing (*e.g.*, to discard readings from a miscalibrated sensor), analysis and visualization of data. All this functionality is available through form-based interaction in a web-browser and programmatically using a REST-based API which may be exercised via clients like cURL [8] and Wget [9] or standard libraries for generating HTTP(S) requests in many languages (*e.g.*, C, Java, Perl). We also provide sample code and complete demo applications as additional aids for new users. Detailed documentation on the API is accessible at our website [10].

C. Typical interaction with Sensor.Network

A typical interaction with Sensor.Network begins with the user obtaining an API key by registering for an account. The API key serves as an authorization token and must be passed in the HTTP request header for all operations requiring authentication. The user can then create a datastream by invoking the HTTP POST method on the URL http://sensor.network.com/rest/resources/datastreams with the HTTP body containing meta information associated with the datastream. If the operation is successful, the status code in HTTP response header is "201" (Created) and the "Location" field contains the URL for the newly created datastream. The URI ends in an integer value, a *datastreamid*, that uniquely identifies the new datastream. Inserting sample data requires an HTTP POST on the URL http://sensor.network.com/rest/resources/datastreams/*datastreamid*/data. Retrieving data involves an HTTP GET on the URL http://sensor.network.com/rest/resources/datastreams/*datastreamid*/data with additional query parameters for specifying the number of data samples, the start and end times for which to retrieve data and the output format (CSV, JSON, XML).

Sensor.Network is agnostic to how sensors and actuators connect to it: either directly, in the case of HTTP-capable devices (*e.g.* phones), or via a gateway.

III. Demonstration

Our demonstration setup consists of several Sun SPOTs[2] [11], an energy monitoring device called the *etherwatt*, and a GPS-equipped phone all sending sensor data into Sensor.Network.

We first take our audience on a walkthrough of Sensor.Network and show different views associated with datastreams (see Figures 1 and 2).

Sensor.Network supports several different types of visualizations *e.g.,* scatter plots, motion charts, location traces, *etc.* These can be easily shared and embedded in external web pages, blogs and forums (*i.e.*, they need not be hosted at sensor.network.com), by including JavaScript code snippets generated by our service (see Figure 3). These visualizations are *interactive*–many support pan-and-zoom–allowing the user to analyze the data at arbitrary granularity. They are also *live*–whenever the hosting page is reloaded, the latest sensor data is retrieved from Sensor.Network and displayed. We show several examples of customers using such *LivePlots* (see Figure 4) and how these visualizations are affected by privacy settings on the underlying data, rendering differently based on a viewer's access rights.

[2]This is a versatile, Java-programmable, sensor and actuator platform developed at Sun Labs.

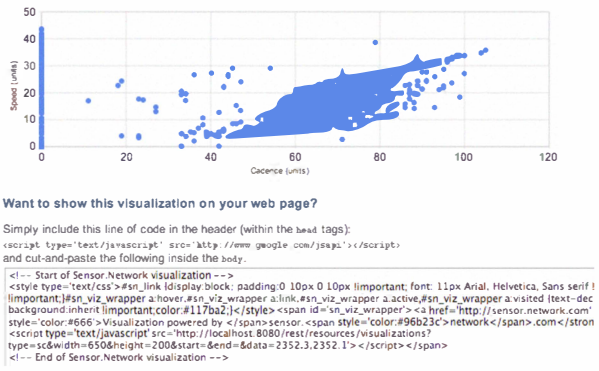

Figure 3. A scatter plot along with the corresponding JavaScript code snippet that can be used to embed this visualization into external web applications

Our service also supports "notifications". A user can specify a boolean expression on sensor values and, whenever sample data is inserted that satisfies that expression, the system can generate either an email, SMS or an HTTP POST to a user-chosen URL (aka *webhook* [12]). In our demo, we show several examples of such notifications:

1) sudden changes in the accelerometer readings on one device causing a warning light to flash and buzzer to sound on another device.
2) GPS readings that fall within user-chosen geographical bounds triggering an email, and
3) energy readings that exceed a user-specified threshold causing a non-critical electric device (a "christmas light" in our demo) to be turned off

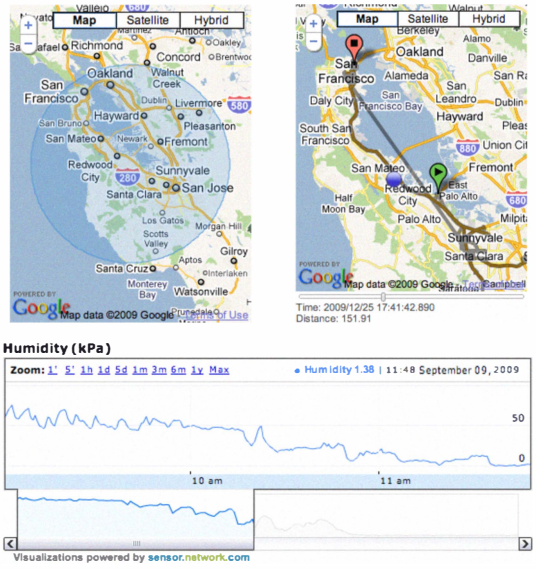

Figure 4. Example *LivePlots* embedded at [13], [14]. The location trace (top) can show either low- (left) or high-fidelity (right) information depending on privacy settings. The line plot below illustrates pan-and-zoom.

It is easy to see the practical implications of these examples in application areas ranging from proactive healthcare to asset tracking to home and industrial automation.

In many scenarios, scientists need to tweak the sensor data collection system and individual sensor nodes in response to collected data, *e.g.,* alter sampling frequency. This points to the need for remote management, even remote reprogramming, of sensor devices. The last part of the demo shows our early explorations in this direction [13].

IV. Conclusion

We have developed a web-based service called Sensor.Network that facilitates a heterogeneous mix of devices to interact with one another and with end users through an open REST-based API. The combination of an open, easy-to-use API and fine-grained access control mechanisms offers multiple benefits: (i) Makes it easy for scientists and hobbyists to share and collaboratively analyze sensor data while freeing them from the effort of setting up the necessary infrastructure. (ii) Enables investigation into correlations between multiple disparate sources of sensor data that would otherwise be missed. (iii) Makes it feasible to compose loosely-coupled mashups between sensors and actuators from potentially different manufacturers and belonging to different owners.

Acknowledgment

The authors would like to thank Ron Goldman for his feedback on an earlier draft of this paper.

References

[1] Sentilla. http://www.sentilla.com/

[2] SynapSense. http://www.synapsense.com/

[3] Johnson Controls. http://www.johnsoncontrols.com/

[4] Echelon Corp. http://www.echelon.com/

[5] Fitbit. http://www.fitbit.com/

[6] RunKeeper. http://www.runkeeper.com/

[7] V. Gupta, P. Udupi, and A. Poursohi, "Early lessons from building Sensor.Network: an open data exchange for the web of things," in *Proceedings of PerCom 2010*, Mar 2010.

[8] D. Stenberg. cURL and libcurl. http://curl.haxx.se/

[9] GNU Wget. http://www.gnu.org/software/wget/

[10] Sensor.Network API. http://sensor.network.com/rest/api/

[11] Project Sun SPOT. http://www.sunspotworld.com/

[12] WebHooks. http://www.webhooks.org/

[13] V. Gupta. SPOTWeb. http://blogs.sun.com/vipul/entry/sun_spots_spotweb_and_sensor

[14] G. Klein *et al.* High altitude weather balloon launch into near space. http://hibal.org/missions/apteryx/