Abstract—An increasing number of embedded devices of all sorts (sensors, mobile phones, cameras, smart meters, traffic lights, home appliances etc.) are now capable of communicating and sharing data over the Internet. 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 with end users through an open, REST-based API. Such a data-exchange can enhance our understanding of the world around us and offer valuable insights for tackling a wide range of issues—from global ones like sustainable resource management to local ones like improving rush-hour traffic flow. The design and implementation of a service like this raises several questions: What are the right data abstractions? How should one balance ease of sharing with privacy concerns? What are effective mechanisms for searching, visualizing and analyzing data? How can one facilitate data-centric collaboration and the composition of loosely-coupled “mashups” between sensors and actuators (e.g. a humidity sensor from one vendor controlling a sprinkler system from another). This paper describes the design choices we made in addressing many of these questions and the rationale behind them. We also provide a brief survey of other comparable projects and evaluate them against a set of common criteria.

I. INTRODUCTION

The Internet is no longer just a network of computers. It has evolved into a network of devices of all types and sizes, all connected, all communicating and sharing information: smart phones, cameras, cars, toys, medical instruments, home appliances, even trees with embedded sensors. Tiny wireless sensors, in particular, promise to fuel a significant expansion of the Internet with their rising adoption in multiple application domains including, but not limited to, intelligent agriculture, proactive health care, asset tracking, environmental monitoring, security surveillance, data center management and social networking. These sensors can generate an enormous volume of very small bits of data such as temperature, humidity, GPS position, acceleration, energy consumption, etc. The real value of this data lies in its analysis which can lead to significant insights that enhance the health of our planet and its populations.

The Sun SPOT [1], developed at Sun Labs, is an example of a versatile, general purpose sensor and actuator platform. Since its public availability in early 2007, more than twenty thousand of these devices have been put to use in different applications by students, researchers and hobbyists around the world. A large number of these applications require facilities for collecting, storing, searching, sharing and analyzing sensor data. These requirements aren’t unique to Sun SPOTs. There are several commercial offerings that attempt to address this need for specific application domains. For example, Sentilla [2] and SynapSense [3] provide wireless instrumentation solutions for data center monitoring and energy management including data collection, storage, analysis, alerts and reporting tools. Similarly, Johnson Controls [4] (a maker of HVAC and building management systems), Echelon [5] (maker of the Meterus smart energy meter) and Fitbit [6] (which makes the fitbit fitness and sleep tracker) all offer custom software for post-processing data from their devices and (in some cases) for managing their equipment controllers. 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.

Figure 1. Examples of Internet-enabled devices include (clockwise from top left) home appliances, mobile phones, energy monitors, sensor platforms, weather stations, fitness/medical devices, and traffic sensors and lights. Our vision with Sensor.Network is to create a data exchange platform with an open API that facilitates the composition of new services incorporating heterogeneous devices from different manufacturers.

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. Such an open data-exchange is attractive for several additional reasons:

1) It enables investigation into correlations between sensor data from multiple disparate sources. For example, the RunKeeper iPhone application [7], provides runners with useful information such as distance, time, pace and path traveled using sensors built into the iPhone. 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.

2) In many scientific communities we’ve interacted with, lots of data still (sadly) sits on individual laptops as scattered text files or spread sheets. In many cases, original data from related studies is hard or impossible to find and only papers with interpretations of the raw data are available. By facilitating data sharing, an open data exchange makes it possible for a scientist to access raw data from someone else’s experiment and draw new conclusions.

3) It enables collaborative classification (e.g. with tagging), annotation, editing (e.g. to discard data from a miscalibrated sensor), analysis and visualization of data using the web as a common platform. In place of static images published in technical journals, one can enable online forums where teams of scientists can experiment with and discuss different types of visualizations for that same data set. Furthermore, for long running experiments, we can create live plots—plots that are redrawn by retrieving the latest data from the open data exchange whenever the web page containing the visualization is rendered in a user’s browser.

4) It spares domain experts the effort of setting up the IT environment required to store their sensor data reliably, managing access controls and performance tuning a compute infrastructure. Instead, the barrier is reduced to learning the API and tools exposed by the open data exchange.

The design of an open data exchange must address several critical issues. Data formats, abstractions, APIs and data insertion/retrieval models (e.g. push v/s pull, polling v/s alerts) need to be flexible enough to meet the requirements of many varied domains yet simple enough for users not trained in computer science. Users must be able to exercise flexible and fine-grained control over how their sensor data is shared (e.g., read-only, time-delayed, low-fidelity) and with whom (owner-only, specific individuals) and when and where alerts are sent. This requires support for authentica-

We are working closely with several customers and have used their feedback to fine tune the design of Sensor.Network over multiple iterations. These customers include: (i) the United States Geological Survey (USGS) which has launched an initiative [8] to convert 15,100 acres of commercial salt ponds at the south end of San Francisco Bay to a mix of tidal marsh, mudflat and other wetland habitats, (ii) Vodafone which is using our service for storing, analyzing and visualizing energy consumption data from a SmartMetering pilot project, (iii) Conservation through Research Education and Action (CREA) [9], a non-profit organization, which is using Sun SPOTs to monitor abiotic environmental variables in the Cocobolo Nature Reserve, Panama, and (iv) a host of hobbyists with data from experiments ranging from bicycle rides to environmental monitoring of tomato plants to a high altitude weather balloon launch into near space [10]. We feel that our experience could be useful to other efforts [11]–[15] we are aware of that share similar goals.

The rest of this paper is organized as follows: Section II delves deeper into the design choices we faced and the decisions that guided our implementation. Section III is a brief survey of comparable efforts to build open data exchanges and Section IV summarizes our contributions and future work we plan to undertake.

II. SENSOR.NETWORK DESIGN

This section discusses the key issues we faced in architecting Sensor.Network and the rationale behind the design choices we made. These issues include choosing good data abstractions and Application Programming Interfaces (APIs), building effective security mechanisms and tools for data analysis and visualization.

A. The Datastream Abstraction

At the core of the Sensor.Network architecture is the notion of a datastream. A datastream refers to a time-series of sensor values that are sampled together. Each value has a name, a type, units, and an optional valid range associated with it. For example, the datastream “My location” may have three sensor values: latitude, longitude and altitude, all of type float, with the first two measured in degrees, with a valid range of -90 to +90 and -180 to +180, respectively and the last in meters.

The datastream abstraction decouples the physical sensor from the high-level phenomenon of interest to the end user. Consider a user interested in measuring the light exposure of an outdoor plant using Sun SPOTs. Let’s say the user defines a “My plant’s light exposure” datastream and starts inserting light readings (as described in section II-B2) from a
Figure 2. Metadata associated with a datastream includes: name, description, tags, media type and URI, category, location, sampling period, access permissions and the name, type, units for each sensor value.

Sun SPOT having unique identifier 0014.4F01.0000.01AB. If, at a later point, the user replaces that Sun SPOT with another having identifier 0014.4F01.0000.020B (e.g. if the original is broken or needs to be moved indoors for charging) the datastream abstraction allows us to stop feeding the light readings from '1AB' and instead start feeding readings from '20B' into the same datastream. Since higher level operations like visualization are performed on a datastream rather than the sequence of readings from a specific sensor, such replacement does not cause any interruption to the phenomenon being tracked.

The above example illustrates the case of a datastream that is fed by multiple sensor nodes at different times. Conversely, a sensor node might concurrently feed multiple datastreams. For example, a portable auto navigation unit can be feeding sensor readings into both “My location” and “My car’s location” datastreams while I’m driving. Later, if someone else drives the car, GPS readings from the navigation unit can continue to feed “My car’s location” (but not “My location”).

Note that some datastreams (e.g. “My location”) are not associated with a fixed location but others (e.g. “My plant’s light exposure”) are. Similarly, some might have sensor readings coming in periodically and others might be aperiodic. Besides the name and sensor values, the following additional pieces of information (many of which are optional) are associated with each datastream:

- A brief description, e.g. “Environmental readings from the tomato plant.”
- A media URI (e.g. image or video) with additional information about the datastream.
- A primary category, e.g. energy, health and fitness, environment, etc. Such categorization helps with organization and in building communities around particular interests.
- Tags which have proven to be an effective search aid on popular services like Flickr and YouTube.
- Location for use in a geographical view or search of a datastream, if the datastream is not ‘mobile’.
- Sampling period for periodic datastreams. This information is used by the system to recognize when sensor data collection is experiencing unexpected interruptions.
- Access permissions which determine what operations are allowed for different classes of users.
- A numeric identifier which is unique across the Sensor.Network system. While other pieces of information are editable and specified by a user, this identifier is immutable and assigned by the system when the datastream is first registered.

B. REST-based API

Sensor.Network supports creating, editing and deleting datastreams, inserting and retrieving sensor data, search, alerts and visualization. This functionality is available both via form-based interaction in a web-browser and programmatically. We anticipate the former method to be used rarely—in cases where the user is just getting started and needs a quick and easy way to explore the service or when dealing with a sensor device incapable of communication, e.g. a user with a very simple garden thermometer could input daily readings manually from a browser. Programmatic access is provided in the form of a well documented REST-based [16] API. The choice of RESTful interfaces makes it easy to support a number of data formats (e.g. CSV, XML and JSON) over the same web protocols (HTTP, HTTPS). The API is directly usable by a broad range of clients using widely available tools like cURL [17], Wget [18] or standard libraries for generating HTTP(S) requests in many languages (e.g., C, Java, Perl). Sensor.Network is agnostic to how sensor data is inserted via the REST API: either directly in the case of HTTP-capable sensor nodes (e.g. phones) or via a gateway. We also provide sample code to exercise our API and complete demo applications as additional aids for new users.

A typical sequence of actions for interacting with Sensor.Network is presented next. We use cURL for illustration and skip over many of the details which are available in our API documentation [19]. Most actions require authentication (described in Section II-D1), indicated by the use of an API key below.
1) Creating/Editing a datastream: Datastreams are defined using an XML descriptor that includes name, tags, location (if any), values, etc. as shown in Figure 3. To create a datastream, this description is sent using an HTTP POST as shown below:

```bash
```

This returns a “Created (201)” HTTP response with the location field in the header set to the URL for the newly created datastream. If the user already has a datastream with the specified name, a “See other (303)” response is returned with the location field set to the URL of the preexisting datastream. The edit operation is similar but replaces the POST with a PUT on the URL returned by the original create operation.

2) Inserting data: Sensor data is inserted into a datastream by posting a sample data descriptor (see Figure 4) in either XML or CSV format on the datastream URL returned by the create operation. This descriptor contains:

- A sensor node identifier string which identifies the sensor node contributing this sensor data (recall that a datastream may be fed by multiple sensor nodes at different times).
- A timestamp indicating when the data was sampled at the sensor node. The timestamp uses XML datetime format [20] to accommodate different timezones.
- As many data values of the appropriate type and in the same order as specified during datastream creation. Each value is checked against any previously specified validity range.

```bash
```

3) Retrieving data: Data can be retrieved from Sensor.Network in multiple formats including CSV, JSON, XML and plain HTML to accommodate the needs of various applications. We also support retrieving data for a specific time period and even the retrieval of some number of most recent samples.

```bash
```

In addition to the “pull” model described above, our design also supports “alerts” to enable “mashup” services built from heterogeneous sensors and actuators. For example, a user can specify a boolean expression of sensor values in a datastream. Whenever sample data is inserted that causes the expression to be true, the system can generate either an email, SMS or an HTTP POST to a user-specified URL (aka webhook [21]).

Data insertion and retrieval are the most frequently used operations. Preliminary tests indicate that a single data sample insertion takes around 30ms\(^1\) and bulk insertion via CSV files takes under 1ms/sample. Data retrieval is almost always a bulk operation and takes roughly 2ms/sample independent of the format—HTML, CSV, JSON or XML.\(^2\)

C. Search and Organization

Search is yet another important component for many of the application scenarios we envision. Both the hobbyist looking for moisture readings to control his sprinkler system

\(^1\)This includes TCP connection setup, HTTP operations, sanity checks on input values and database insertion.

\(^2\)We feel that retrieval can be made much faster by devoting some effort to performance tuning.

---

Figure 4. Example data descriptor in (a) XML, and (b) CSV format.

Many of our current and potential users have legacy data in the form of CSV files and we provide a mechanism to bulk-upload multiple samples in a single operation. We do require that users pre-format their CSV file to conform to a specific order (timestamp, sensor node identifier, sensor values) so the system can parse the samples correctly.

```bash
```

"http://sensor.network.com/rest/resources/datastreams/id/data"

---

Figure 3. Datastream.xml: An example of a minimal datastream descriptor.
and the runner looking for temperature and humidity readings for her run need the ability to find datastreams meeting specific criteria (e.g. based on sensor type and location). We provide a mechanism for organizing datastreams into some broad categories (described in Section II-A) and this list is still evolving. However, we strongly feel that a flexible search mechanism is much more important and have designed the system to support searching based on datastream name, description, tags, owner, location and even value names and units.

D. Security Mechanisms

Sharing is a central idea for our service and some might see security mechanisms for access control and privacy as being at odds with it. However, our experience suggests that users are more willing to store their data on Sensor.Network if they can control how it is shared and with whom.

1) Authentication: As illustrated in Section II-B, most operations on Sensor.Network require the invoking entity to authenticate itself. Operations initiated in a browser use username/password and those initiated programmatically use an API key. The API key is a base-64 encoded string which serves as an authorization token and must be passed in the HTTP request header as part of the “X-SensorNetworkAPIKey” field. Each user is assigned a unique system-generated API key that identifies them. A user may request the generation of additional API keys to delegate authority for limited operations or for specific time periods to other entities without having to share their password.

Alerts pose another interesting authentication issue. Some services [14] allow the creation of alerts that can result in the system sending data to a target (e.g. a URL) chosen arbitrarily by the alert creator. We see this as a security issue that can lead to “alert spamming”. Our design only permits alerts to verified targets—email addresses, phone numbers or URLs under control of the alert creator. A user’s control over their associated email address is established at the time of account creation. New users must request an account by email and the system sends back a unique, hard-to-guess “invitation code” which the user must input on the registration web page. Other targets (phone numbers, URLs) can be similarly verified: by sending a nonce via SMS and requiring the alert creator to input it at the website or by displaying a nonce at the website and having the alert creator set things up so this nonce can be retrieved via an HTTP GET at a URL.

2) Authorization: The owner of a datastream can choose which users are authorized for what operations. Access control is implemented using a UNIX-like model of permissions based on user classes and access types. There are three built-in user classes for each datastream: (i) owner (ii) registered users of Sensor.Network, and (iii) everyone. In addition, each user can create arbitrary groups of users and authorize different groups for different operations. Besides data insertion and retrieval, we are investigating other forms of access to sample data e.g. low-fidelity or time-delayed access.

3) Confidentiality: Encryption is essential to guarantee that authentication credentials like passwords and API keys or sensor data are not exposed to eavesdropping by unauthorized entities. The use of a REST-based API makes it easy to layer the entire protocol interaction over HTTPS.

E. Tools for Visualization and Analysis

Figure 5. Valuable insights can be derived from a rich set of visualization tools. Here we illustrate two of the supported visualizations (scatter plot and location trace) for the datastream from a century bike ride. The top plot correlates cadence to speed and, as expected, shows a positive correlation. The different slopes correspond to different gears. Parts of the ride where the rider was able to coast downhill (zero cadence but non-zero speed) are also easy to spot. The location trace lets the user replay location changes of an object at different speeds. Closely coupled composite visualizations, e.g., being able to identify the speed/cadence data point in the scatter plot corresponding to a specific location in the bottom visualization, can further enhance data analysis.

The real value of an open data exchange lies in the insights it provides for turning data into actionable information. Tools for visualization, analysis and collaboration are therefore essential. Our service currently offers several different visualizations including line plots, scatter plots, bar graphs, location traces, etc. These plots are interactive (e.g. line plots

---

3There were many attempts in the early days of the web to organize and categorize websites (e.g. by Yahoo! [22]) but as the information on the web exploded, those attempts were quickly abandoned in favor of better search mechanisms.

4The UNIX model is more limiting in comparison. For instance, it is not possible to have a group with “read only” permissions and a different group with “write only” permissions on the same file.
support pan-and-zoom, location traces support a time slider as in Figure 5) and can be embedded in web-pages, blogs etc. not hosted at Sensor.Network (see [10]). In addition, we also offer views that capture essential information about groups of datastreams. For example, a dashboard view is useful for conveying recent activity and access permissions while a map view is useful for geographical search. The ability to dynamically generate tag clouds for a specific set of datastreams (e.g., recently active or belonging to a specific geographical area) is an interesting means for studying popular categories and spotting trends.

The ability to plug in different analysis engines, e.g. one that processes a time-series of GPS and accelerometer readings to deduce activity (walking, running, driving), is another important requirement. We expect many of these modules to be domain specific and our users are excited about the possibility of sharing and reusing such modules within their scientific communities.

### III. Related Work

This section briefly surveys other projects similar to ours and the comparison is summarized in Table I. Our evaluation is based on publicly available information. These projects may very well have the vision of implementing additional functionality that isn’t currently exposed to their users.

Open data exchanges came about as a result of Wireless Sensor Network (WSN) research groups investigating ways to collect sensor data and manage it easily via ubiquitous clients like web browsers. One of the earliest such initiatives is SensorBase [11], [12] by the Center For Embedded Networked Sensing (CENS) at UCLA. It uses a relational database table as its data abstraction and an SQL-centric API. Users can group related tables into “projects”. This has the benefit of being very general but is too low-level for domain experts without a computer science background.

For example, a scientist trying to correlate sensor readings from two different projects, would need to understand and implement SQL joins. SensorBase seems to have some the most advanced notification mechanisms that the user can set on a per-table basis based on the satisfaction of multiple conditions.

The Sensorpedia [13] project at Oak Ridge National Laboratory (ORNL), the SenseWeb [15] project at Microsoft and Nokia’s SensorPlanet [23] are all examples of similar initiatives within industrial or government, i.e. non-academic, research organizations. There is very little public information available for SensorPlanet. SenseWeb is the most mature project amongst these three. It offers a platform on which other applications can be built. One such application is SensorMap [24] which matches up sensor data from SenseWeb on a geographical map interface. However, SenseWeb’s use of SOAP [25] makes it more heavy-weight compared to a REST-based service and this complexity raises the barrier to entry for potential users. Sensorpedia is the newest of these three services with the stated aim of utilizing Web 2.0 social networking principles for organizing and providing access to sensor network related data sets. Users can publish and subscribe to sensor feeds using Atom. The service does not appear to offer any data archiving which makes it unsuitable for users (e.g. hobbyists) that, for lack of ability or resources, do not wish to set up their own data store. They appear to have under development an interesting spreadsheet-inspired visual programming tool for mixing sensor data from multiple sources but this functionality is currently not publicly available nor is there any information on the security model or visualization support.

Pachube [14] stands out among all the services we looked at because it is the only one backed by venture capital funding. Its vision of facilitating different sensor and actuator devices to connect easily with each other is very similar to

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data Abstraction</strong></td>
<td>Datastreams</td>
<td>Database table</td>
<td>Atom Feeds</td>
<td>I/O Feeds (similar to datastream)</td>
<td>Web service representing sensor node</td>
</tr>
<tr>
<td><strong>Open API</strong></td>
<td>RESTful</td>
<td>SOAP</td>
<td>RESTful</td>
<td>RESTful</td>
<td>SOAP</td>
</tr>
<tr>
<td><strong>Data archival</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Unknown</td>
</tr>
<tr>
<td><strong>Data formats</strong></td>
<td>XML, CSV, JSON</td>
<td>XML, CSV, JSON</td>
<td>Atom, GeoRSS</td>
<td>EEML, CSV, JSON, RSS, Atom</td>
<td>Text, Excel</td>
</tr>
<tr>
<td><strong>Data uploading</strong></td>
<td>Push</td>
<td>Push</td>
<td>Not Applicable</td>
<td>Push, Pull</td>
<td>Unknown</td>
</tr>
<tr>
<td><strong>Authorization</strong></td>
<td>Rich</td>
<td>Fine-grained</td>
<td>Unknown</td>
<td>Password &amp; API Key</td>
<td>Password</td>
</tr>
<tr>
<td><strong>Search by</strong></td>
<td>Name, Description, Tags, Owner, Value names, Value units</td>
<td>Project name, Table name, Tags</td>
<td>Feed Name, Feed Description, Tags</td>
<td>Feed name, Feed description, Tags</td>
<td>Sensor name, Owner</td>
</tr>
<tr>
<td><strong>Graphical visualization</strong></td>
<td>Rich, interactive, embeddable plots, Dashboard &amp; Map views</td>
<td>Static line plots, Map view</td>
<td>Map view, others unknown</td>
<td>Line plots, Map view</td>
<td>Application provided</td>
</tr>
<tr>
<td><strong>Alerts/Notifications</strong></td>
<td>Compound expressions</td>
<td>Compound expressions</td>
<td>Unknown</td>
<td>Simple expressions (insecure*)</td>
<td>None</td>
</tr>
</tbody>
</table>

*Notifications can be set on any target URL without proving ownership or control over the URL.
ours. When we were first starting out, we evaluated Pachube in its early days and noticed several shortcomings that prompted us to pursue an independent effort. While some of those shortcomings have been addressed subsequently, such as support for “pushing” data into Pachube (the pull model doesn’t work for sensor nodes behind firewalls), others still remain. Pachube’s EEML [26], the data format used to insert and retrieve sensor data, mixes sensor data and metadata and causing metadata to be repeated unnecessarily with individual data samples. The security model doesn’t appear to be well developed: all data is viewable by anyone, even without an account on Pachube, and the alert mechanism is prone to “spamming” as described in Section II-D.

IV. CONCLUSION AND FUTURE WORK

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 service makes it easy for scientists and hobbyists to share and analyze sensor data and compose loosely-coupled mashups between sensors and actuators. We have arrived at a robust and flexible design based on feedback from our users over multiple design iterations. The paper presented the key aspects of building such a service and the design choices we made, with an intent to benefit other projects with similar goals. We also provided a brief survey comparing these other projects.

For the near future, we plan to fill in some gaps between the design presented in Section II and our current implementation, e.g., finishing up the implementation of alerts, supporting delegated authorization and composite/link visualizations. Longer term, we are investigating support for reprogramming and remote management of devices. Our experiments with SPOTWeb [27] and Yggdrasil [28] represent an early exploration in this direction. We are also investigating integrating statistical packages like R [29] as pluggable analysis modules, visual programming environments [30] for interactively creating analysis workflows, and visualization tools that support collaborative annotation and editing of data. We also plan to study other data storage and processing models (e.g., Hadoop [31]) that may offer a scalability advantage over relational databases by moving computation closer to data storage.

V. ACKNOWLEDGMENTS

We would like to thank Ron Goldman, Randy Smith and the anonymous referees for their feedback on an earlier draft of this paper.

REFERENCES

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