Sensor Data as a Service - A Federated Platform for Mobile Data-Centric Service Development and Sharing

[1]Jia Zhang, [1]Bob Iannucci, [1]Mark Hennessy, [1]Kaushik Gopal, [1]Sean Xiao, [1]Sumeet Kumar, [1]David Pfeffer, [1]Basmah Aljedia, [1]Yuan Ren, [1]Martin Griss, [1]Steven Rosenberg, [2]Jordan Cao, [3]Anthony Rowe
[1]Carnegie Mellon University – Silicon Valley, USA
[2]SAP, USA
[3]Department of Electrical Computing Engineering, Carnegie Mellon University, USA
jia.zhang@sv.cmu.edu, bob @sv.cmu.edu, martin.griss@sv.cmu.edu, steven.rosenberg@sv.cmu.edu, jordan.cao@sap.com, agr@ece.cmu.edu

***Abstract*—**The Internet of Things (IoT) offers the promise of integrating the digital world of the Internet with the physical world in which we live. But realizing this promise necessitates a systematic approach to integrating the sensors, actuators, and information on which they operate into the Internet we know today. This paper reports the design and development of an open community-oriented platform aiming to support *federated sensor data as a service*, featuring interoperability and reusability of heterogeneous sensor data and data services. The concepts of virtual sensors and virtual devices are identified as central autonomic units to model scalable and context-aware configurable/reconfigurable sensor data and services. The decoupling of the storage and management of sensor data and platform-oriented metadata enables the handling of both discrete and streaming sensor data. A cloud computing-empowered prototyping system has been established as a proof of concept to host smart community-oriented sensor data and services.

I. Introduction

With the advancement of the Internet of Things (IoT), the number of active web-connected devices is rapidly approaching 50B. These include static and mobile sensors and actuators planted in surrounding environments including buildings and spaces, as well as devices embedded in mobile smart phones and moving vehicles [1]. Such sensor-equipped connected devices, collaboratively, could help sense the world [2] and generate enormous amounts of potentially interesting and useful data [3, 4]. More importantly, such datasets present great opportunities to create unprecedented value-added services [5], for example, energy saving recommendations based on usage pattern recognition and exceptional situation prediction and detection for disaster management.

Realizing this promise, however, necessitates a systematic approach to integrating the sensors, actuators, and information on which they operate into the Internet we know today. Recent years have witnessed a significant amount of effort building various sensor networks. The first wave of sensor networks was characterized by a broad and diverse collection of mostly incompatible, purpose-built hardware. It surfaced some issues in wireless networking and allowed the building of 1000's-scale networks. Each such network was mostly homogeneous internally; however, there was little sharing of common elements from one network to the next. The second wave is motivated by the barriers that the first wave encountered. Making homogeneous networks scale to 10,000 and beyond compels the view that the stack must be divided up and that common functions should be shared rather than re-invented.

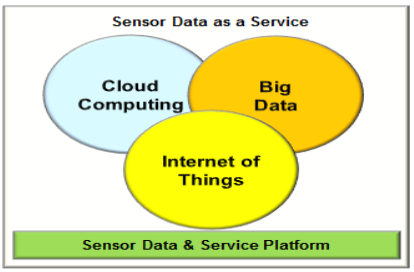

Fig. 1 Service-oriented sensor data & service platform.

With this motivation, a number of community-oriented platforms have been established to allow users to contribute and share sensor data in heterogonous data sources. Some tools are also developed to assist users in mashing up shared sensor data and developing value-added applications. Representative platforms include Microsoft SenseWeb [6], Global Sensor Networks (GSN) [7], SensorBase [8], IrisNet [9], and Semantic Sensor Web [10]. Such a peer production paradigm offers significant advantages such as large spatio-temporal coverage, on-demand resource sharing, amortized cost sharing over applications, and feasibility of developing new value-added sensor data services [6].

With these many sensor networks, their interoperability remains a challenge. Data stored at individual sensor networks are isolated and cannot be easily incorporated. As shown in Fig. 1, the advancement of cloud computing, big data and IoT calls for sensor data as a service. Therefore, one objective is to develop an architecture that addresses the fundamental issues of device semantics, heterogeneity, semi-connectedness, fault tolerance, clustering, low-power operation, and security [11, 12]. These devices may belong to public and private owners and may move constantly and may not remain available all the time. Inexpensive sensors may break; batteries may run down; and software may become obsolete. In addition, one must find relevant data sources and transform the datasets into unified, machine understandable forms. To properly understand data, its provenance as how it was created and gathered may have to

978-0-7695-5026-8/13 $26.00 © 2013 IEEE
DOI 10.1109/SCC.2013.34

be assessed. To process a federated set of data, one may leverage reusable data processing tools developed by others and compose them into an automated workflow (procedure). Throughout this data sharing and analysis process, privacy and confidentiality may have to be preserved as well [6].

It has been claimed that service-oriented computing would empower an infrastructure to provide a comprehensive solution to these challenges [11]. However, sensor data and services' reusability and interoperability among various middleware tools and platforms remains a challenge. Existing platforms and solutions do not provide system-level support for such key features.

Toward enabling sensor data as a service, this project aims to maximize the sharing and utility of available sensor data sources, data, and data processing tools, to enable greater sensor data services. Our strategy is to study the programmability [13] and fundamental data model of sensor networks, and develop a service-oriented platform to facilitate community-scale data services sharing and development. The platform will be made accessible to research groups and individuals in the global community who have original data and tools, and will provide direct assistance to potential contributors who will develop value-added data and tools.

This paper reports on this on-going project and our preliminary design and development of the platform prototype. Our contributions are three-fold. (1) The platform is centered on a data model that supports scalable and interoperable sensor data and services modeling. (2) Sensor data and platform-oriented metadata are decoupled and stored separately, to support sensor data federation and reuse. (3) A cloud computing-empowered system architecture further enhances the scalability of the platform.

Leveraging SOA, our presented platform provides a unified view of data and workflow provenance. The infrastructure directly improves the community-driven data-centric service provisioning capabilities for the smart space. The broader impact of our project lies in enabling more comprehensive data-oriented service development and sharing.

The remainder of the paper is organized as follows. In Section 2, we discuss related work. In Section 3, we analyze architectural decisions. In Section 4, we introduce our data model. In Section 5, we present the system architecture of the platform. In Section 6, we present our prototype implementation. In Section 7, we draw conclusions and discuss future work.

II. Related Work

At the most basic level, the ability to combine sensor data from disparate sources relies on a mechanism to readily identify equivalence classes–mapping raw readings from transducers of different manufacturers to correctly-translated, error-bounded readings of well-recognized physical quantities (*e.g.,* temperature, pressure). The IEEE 1451.4 Transducer Electronic Data Sheet (TEDS) standard [14] provides such a mechanism. TEDS-equipped sensor devices can self-identify and provide core information about a sensor–its manufacturer, version, serial number, transducer "type" (drawn from a set of well-known classes), and parameters for translating and interpreting raw readings.

Botts and Robin initiated the development of an XML-based sensor modeling language that grew into SensorML [15] which is under the oversight of the Open Geospatial Consortium (OGC). SensorML provides means to describe processes, including the process of translating measurement by sensors into data in common formats.

Madden, Franklin, Hellerstein and Wong created TinyDB [16] to demonstrate the concept of modeling sensor network data access as SQL-like queries and demonstrated the ability to push processing of sub-queries into the sensor network itself.

Mohamed and Al-Jaroodi [11] surveyed the requirements of service-oriented middleware for wireless sensor network and reviewed some representative approaches. In recent years, a number of efforts have established infrastructures and tools to enable acquisition, storage, processing and replication of distributed sensor data.

Internet-scale Resource-Intensive Sensor Network Services (IrisNet) [9] offers a set of generic functionalities and APIs to allow user to query and process distributed collections of high-bit-rate sensors. IrisNet adopts a two-tier architecture comprising sensing agents (SA) that collect and preprocess sensor data and organizing agents (OA) that store sensor data. On top of IrisNet, Chen et al. [17] proposed *X-Tree Programming*, a database-centric approach to programming over a large-scale of Internet-connected sensing devices. Microsoft SenseWeb [6] provides a Web 2.0 platform for users to upload and access sensor data streams from shared sensors across the Internet. SensorBase [8] built a centralized data storage and management platform that allows users to publish and share ("slog") sensor network data using a blog-like approach. It allows users to define some data types, project groups, and access control policies. SensorBase also allows users to query data sets based on geographic location, sensor type, date/time range, and other relevant fields.

On the other hand, Global Sensor Networks (GSN) [7] adopts a scalable peer-to-peer architectural model in favor of integrating heterogeneous sensor network technologies. GSN also supports the asynchronous publish/subscribe pattern in addition to synchronous queries.

Some researchers focus on building tools to support sensor data manipulation. Among them, the Desthino (Distributed Embedded Things Online) project aims to provide a practical set of software tools, such as Minos, to help users collect and store sensor data from heterogeneous distributed sensors [12]. Sensor Observation Service (SOS) is a standard Web service specification, defined by the Open Geospatial Consortium (OGC) Sensor Web Enablement (SWE) group, aiming to standardize the way of requesting, filtering, and retrieving sensors and sensor data to enhance sensor interoperability [18].

Some researchers explore how the Semantic Web can be integrated with a Sensor Web. For example, SemSOS [19] models the domain of sensors and sensor observations in a suite of ontologies. Based on the SemSOS ontologies, Semantic Sensor Web [10] was established where semantic annotations are added to comprising sensor data; and sensor observations can be reasoned through ontology models.

Some researchers study the scalability of sensor networks. For example, Liu et al. [1] studied a large operating sensor network system, GreenOrbs, and concluded that an event-based routing structure will gain more adaptability. Minos [12] proposes a simple 5-tuple data model (sensorType, node, nodeType, deployment, user). A concept of virtual sensor is introduced in GSN [7] to abstract sensor data as temporal streams of relational data, and to represent derived views or a combination of sensor data from different sources. SenseBox [20] introduced an autonomous computing unit encapsulating environment and REST APIs. In contrast to their data models, ours favor seamless integration of cyber world (functional units) and physical world (sensors) allowing dynamic composition (i.e., through the concept of device).

Three commercial sensor data platforms appear to be close to the goal of our project: TempoDB [21], Cosm, and SensorCloud. The three platforms share some common features. First, they all focus on time-series datasets. Second, the platforms concentrate on sensors with discrete readings instead of continuous media. Third, they all provide REST APIs to facilitate storing, retrieving, and querying time series data. The comparison of the unique features of the three platforms is summarized in Fig. 2.

TempoDB [21] is a database as a service aiming to store & analyze time series data from sensors, smart meters, and automotive telematics. It provides range rollups, bulk data upload, long range historical analysis and data summaries that generate statistical answers. Metadata tags and attributes in TempoDB allows the data series to be grouped and tagged. In-time data visualization is supported.

Cosm [22] is an online platform that allows users to upload sensor data and build applications based on the data. Both pull and push modes for sensor data are supported in multiple formats such as XML, JSON and CSV. An open

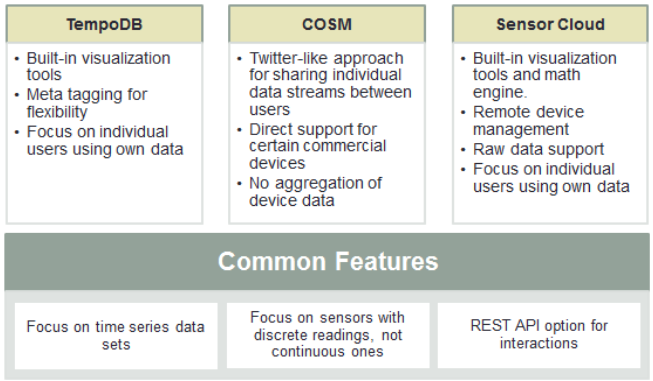

Fig. 2 Comparison of existing sensor platforms.

community has been established as well.

SensorCloud [23] is a time-series sensor data storage, visualization and management platform that leverages cloud computing technologies to provide data scalability, visualization, and user programmable analysis. Data API is equipped to allow third-party devices, sensors, or sensor networks to be linked in. Location-oriented query is supported. Data analysis applications can be applied through a data aggregation platform.

Compared to the existing sensor data and management platforms, our platform is centered on a data model that supports scalable and interoperable sensor data and services modeling. Not intending to build yet another sensor data and service repository in the increasingly crowded space, our platform is designed to leverage SOA to allow other repositories to be plugged into our platform serving in different roles, such as sensor data storage.

III. ARCHIECTURAL DECISIONS

We adopted the Architecture Tradeoff Analysis Method (ATAM) methodology [7], a systematic architectural analysis method, to study and evaluate the critical system requirements in a utility tree. Requirements are elicited from several domain experts who worked with the team throughout the project. As summarized in Table I, quality attributes are identified, associated with attribute concerns phrased by user requirements and scenarios listing key performance indicators. For example, as shown in Table I, finding the standard platform for the IoT must, fundamentally, face the hardest problem at the heart of the IoT: scalability. Based on the ATAM analysis, several architectural design decisions have been made: NoSQL database, SOA system, REST-based API, distributed processing, load balancing, and service health monitor.

NoSQL Database. The sensor repository needs to be flexible, allowing for various types of devices, sensors, and sensor readings. It should also allow "Free-form" metadata to be associated with readings. Thus, data should be stored in raw formats and rely on structured queries to synthesize the information on demand. Furthermore, streaming data sets should be supported in addition to discrete sensor readings. Therefore, we chose to adopt NoSQL database in the format of (key, value) pairs instead of table-oriented relational database.

SOA System. Our platform is obviously centered on a sensor data repository, and empowered by a number of service components. For example, to extract meaningful information, raw sensor data may have to be aggregated or derived. Thus, a data processing component is needed. Our platform will support extensibility and adaptability, and an SOA-guided layered architectural model is adopted.

REST-based API. According to the best practice from SOA applications, a REST-based API will be able to allow an open community to easily access our services and build new applications.

Table I. ATAM utility tree.

Quality Attribute	**Scalability**
Attribute Concerns	Supports different types of sensor data
Scenarios	• System shall accept all kinds of sensors send in data in all forms. • System shall verify same sensors sending different formats based on upgrades or revisions. • Data from different repositories shall be federated to yield a composite result.
Attribute Concerns	Facility to scale backend computing based on demand
Scenarios	• Additional servers shall be added for parallel computing of data-intensive calculations.
Attribute Concerns	Support scalable load
Scenarios	• System shall support incoming of huge volumes of sensor data.
Quality Attribute	**Reliability**
Attribute Concerns	Database service available when database node breaks down.
Scenarios	• Platform shall ensure data/data analysis operations reliable.
Quality Attribute	**Interoperability**
Attribute Concerns	The application must be easy to integrate with 3rd party data sources, software, and data processing.
Scenarios	• System shall accept all kinds of sensors send in data in all forms. • New data visualizers can be integrated with the solution to provide more views for querying the data. • 3rd-party tools and software can query the data in our database and receive results in JSON format for consumption.
Quality Attribute	**Security**
Attribute Concerns	Platform must transfer data over a secured channel.
Scenarios	• All data connections must be over Transport Layer Security (TLS).
Attribute Concerns	Users/sensors must be securely authenticated to access the platform.
Scenarios	• Access control shall be supported by sufficient credentials. • Contributing sensors must provide sufficient credentials must be provided.
Quality Attribute	**Integrity**
Attribute Concerns	Data shall maintain accuracy and consistency during any operation, such as transfer, storage or retrieval.
Scenarios	• During data transportation, connection must be secured by Transport Layer Security (TLS) from corruption.
Quality Attribute	**Data Freshness**
Attribute Concerns	Data freshness shall be ensured.
Scenarios	• When sensors send data, proper timestamp shall be attached.
Quality Attribute	**Privacy**
Attribute Concerns	Data stored must be private by default, but can be shared with authorized users.
Scenarios	• Users shall be able to share their data with other users. • Users shall be able to change configurations.
Attribute Concerns	Certain sensitive data must be manipulated to protect source of information.
Scenarios	• Stored data shall be obfuscated in a way to protect privacy of individual while keeping dataset statistically accurate over dataset. • Uniquely identifying data shall be drooped from authorized data sources to avoid reading from being traced back to sources.
Quality Attribute	**Performance**
Attribute Concerns	Querying a large dataset may be time and computationally intensive.
Scenarios	• Database shall be indexed on certain elements to expedite common queries. • Data shall be compressed or preprocessed to minimize data transfer over the network and storage utilization.
Quality Attribute	**Reusability**
Attribute Concerns	Different parts, or the entire application could be reused as services.
Scenarios	• Query service and visualizer shall be reused for other data query services rather than sensor data. • The entire solution shall be reused or redeployed as a subsystem in related systems.
Quality Attribute	**Extensibility**
Attribute Concerns	System supports extensions to future growth.
Scenarios	• System shall handle addition of new features such as different methods of querying data. • System shall support future changes in hardware and software infrastructure.

Distributed Processing. Performance is one critical requirement. How to ensure real-time queries over big data has to be resolved. We have decided to explore the programmability of sensors, and introduce a MapReduce-like approach to enhance performance by allocating processing tasks to distributed sensors. Our belief is that the storage and processing capabilities of sensor systems needs to scale with the addition of sensors, and that sensor devices, gateways and other networking means will, increasingly, incorporate processing. This favors the notion of pushing computation to the lowest levels of the sensor directed acyclic graph (DAG) and introduces interesting questions about how this programmability can be abstracted. In addition, it is important to recognize that distributed computing can be used in either a "pull" fashion (query-demand-driven) or a "push" model (data streaming). In both cases, the computation that is distributed within the DAG should be viewed as performing both data-processing functions as well as power-management functions. A jabbering sensor that repeats an unchanging value, burning precious local energy and bandwidth in the process, is much less desirable than a sensor that reports truly interesting information as it happens. We discuss, below, the concept of "actionable information." One other aspect of distribution is the notion of *memoization and invalidation*. Storage of raw readings with on-the-fly or scheduled translation to application meaningful readings permits delayed binding on the translation function (e.g., updating a TEDS record for a set of sensors). But, for

performance reasons, the sensor system may elect to memoize such translated values to avoid the reprocessing costs. When any such value is derived from translation rules (again, TEDS being an example), the dependence of these values on the translation rules must be maintained so that proper invalidation takes place when, for instance, the rules are updated.

Load balancing. To ensure availability of the platform, load balancing has to be supported. Scheduling algorithms will be adopted to direct requests to some of the stack of backend servers.

Service Health Monitor. To ensure scalability, performance, and availability, a dedicated service monitoring and management module has to be established on the platform.

IV. Data Model

One central step in designing a heterogeneity-oriented data platform is an appropriate data model [12]. Recall that our main goal is to favor interoperability and reusability. We do not intend to add yet another sensor data and service repository in the already crowded space. Instead, our platform is designed as a data and service layer, to leverage SOA to allow other repositories to be plugged into our platform serving in different roles, such as sensor data storage.

We thus differentiate the cyber world from the physical world and allow dynamic composition, by introducing two concepts of *container* (device) and *virtual unit* (sensor/device). A container represents an aggregation relationship of one to many sensors. As shown in Fig. 3, various types of sensors serve different purposes, for example, sensor measuring pressure, sensor catching audio, sensor recording light, humidity, temperature, and so on. In other words, each sensor is able to provide some services (functionality) to the outer world. In reality, a device is typically used to gather a collection of sensors (as well as actuators) and install them into a physical container. As shown in Fig. 3, a physical device (e.g., Firefly at Carnegie Mellon University, http://www.ece.cmu.edu/firefly/) is constructed to host a set of sensors serving various functionalities, and then deployed and installed into an actual physical location. Such a device is therefore able to provide a vector of measurements. If a comprised sensor is broken, it can be replaced by another one and the device will remain functioning properly.

We also adopt a concept of virtual unit (sensor/device) to

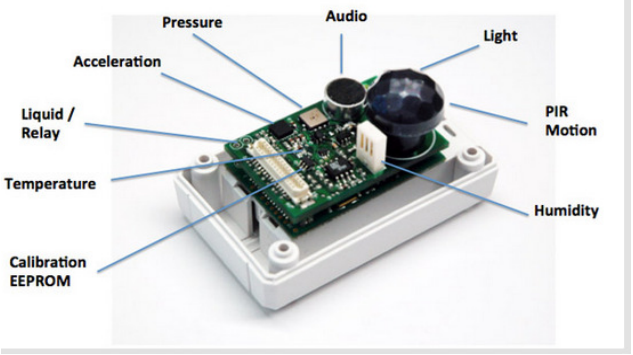

Fig. 3 Concept of device/sensor.

represent certain services (functionalities) without referring to actual physical implementations. For example, a virtual temperature sensor may be defined to represent the average temperature inside a room, which can be computed over a collection of sensors distributed inside the room. Similarly, a virtual device represents a set of services to be provided (*e.g.*, temperature+humidity+noise), which can be realized by a set of concrete sensors that are actually located in different physical devices. As a result, these two concepts enable the seamless integration of logical units (functional units) and physical units (sensors) allowing dynamic composition.

Fig. 4 illustrates a simplified version of our data model underlying our sensor data and service platform. It centers around the concept of an abstract Discoverable, Federatable Sensor (DFableSensor). A DFableSensor is registered to our platform with a public service interface, representing its functions that can be leveraged by the outer world. An abstract DFableSensor is realized as either a virtual sensor or a physical sensor. As aforementioned, a virtual sensor may be an aggregation of multiple physical sensors. A DFableSensor is modeled as an autonomous unit, carrying meta-sensor section that helps to interpret how to understand its sensor reading.

Sensor reading is identified as an independent entity associated with each sensor unit, which may refer to the data format of either discrete reading or streaming reading. Such a separation enables separate storage of raw sensor reading data from the metadata of corresponding sensor.

An abstract Discoverable Device (DisDevice) is introduced as a generalization of device and virtual device, as shown in Fig. 4. It shows an aggregation relationship to a collection of DFableSensors. A DisDevice is published with a service interface. Both sensors and devices can carry attributes including locations (e.g., gps-location), context (the best way to use the unit), and security and privacy requirements. Their detailed discussions and implementations are out of the domain of this paper.

As shown in Fig. 4, all discoverable and reusable units are organized by corresponding public registries hosted by our sensor data and service platform: sensor registry, device registry, and device agent registry, all generalized as registries. At the registry, license and access control permissions can be defined and associated by users. Note that sensors from other third-party repositories can be registered into our registries so as to expand our data pool and enable distributed data storage and handling. The same as third-party devices. The mappings between registered sensors and devices are stored in the registry, serving as "yellow-pages" to enable sensor/device sharing, discovery, and federation. The device registry stores information about each physical device, which can be used for queries and to look up devices when in the event of a failure in a sensor network.

To leverage the programmability of sensors, each device is associated with a dedicated device agent, which serves as the manager (gateway) of the corresponding local sensor

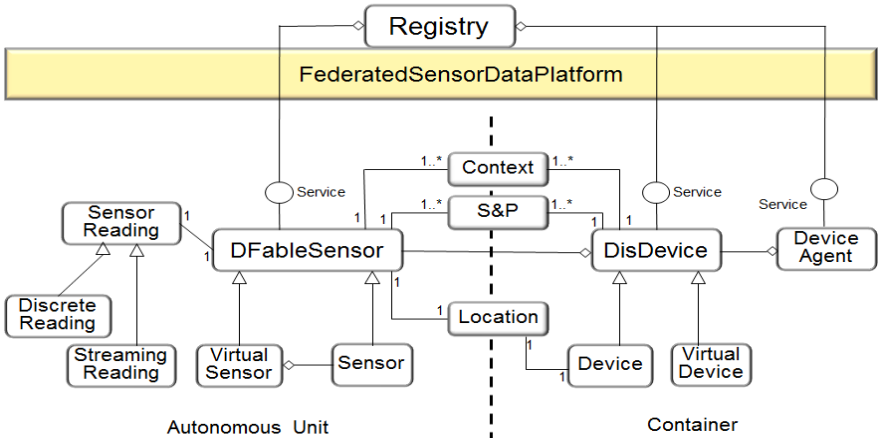

Fig. 4 Data model.

network. A device gateway may also be responsible for converting the comprising device readings to a consumable format (*e.g.,* JSON), and transmitting it to our platform for storage. All entities are centered around the entity location, which represents the contextual situation of a unit. The entity of location also implies the portability of a unit.

Focusing on that subset of the IoT that is made up of sensors, experience has shown that the simple-minded approach of transmitting sensor readings periodically to a central database suffers from the real cost of communication and the fact that, amidst the sea of data, there is comparatively little actionable information. Our improvement over this approach, as mentioned above, is to design and program sensors to process locally and recognize the actionable information - which is then transmitted up the chain for further action. The downfall of this approach is that one's view of "actionable information" may change over time - and it may not be recognizable purely from single-sensor readings. This lends support to the notion that the preferred IoT architecture should presume programmability and generality top-to-bottom including the ability to aggregate, store, process and react at every level – in both pull and push models. As experience with IoT networks grows, we imagine that domain-specific specialization will naturally follow. But for now, general networks, nodes and gateways provide the richest proving ground.

V. System Architecture

Based on our established data model, we have designed the high-level system architecture of our sensor data platform. As shown in Fig. 5, a Model-View-Controller pattern is adopted. Sensor reading data and metadata serve as the backend persistent layer. A personalizable dashboard serves as a front-end interface based on user requirements and profiles. In the middle tier of the federated sensor data platform, six major service components are defined.

The data modeler provides templates for users to describe their sensor data. The data federator offers facilities for users to dynamically aggregate data to create new knowledge. The search engine helps users define their needs and find data that may interest the users. The Data service modeler allows users to define and contribute specific data analysis procedures that can be researchable and reusable to other community users. The data service mashup offers a workflow engine to help users compose existing data services into value-added services. The contributor manager handles registered community users and their profiles.

The deployment view of the entire platform is illustrated in Fig. 6. Sensors are grouped and encapsulated into devices. Covering a certain range, a receiver is established dedicated for a collection of devices. A sensor gateway is established to interface between sensors and our platform. Raw sensor readings and registry management data are decoupled, to enhance scalability and adaptability of the platform. Raw sensor readings may be gathered and used in various ways oriented to the open community. Their trend toward "big data" demands scalable data storage. On the other hand, registry management data should be organized in a structured manner to ensure metadata discovery and analysis.

VI. Prototype Implementation

The prototyping platform was implemented as a proof of concept of our data model at Carnegie Mellon University - Silicon Valley campus (CMUSV), as shown in Fig. 6. Amazon DynamoDB (http://aws.amazon.com/dynamodb) was adopted as a NoSQL database to host raw sensor data. One major reason why we selected it is its scalability. Facing traffic, storage can automatically grow, by spreading data and processing over a sufficient number of servers. A sim-

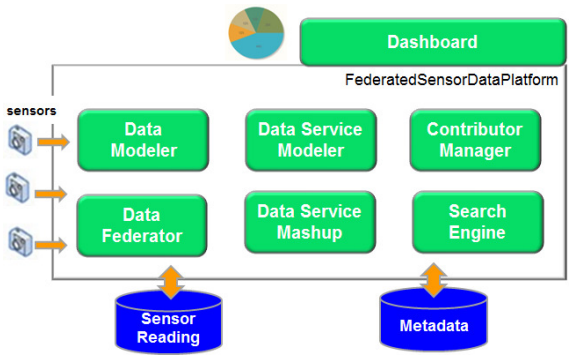

Fig. 5 System architecture.

plified database schema includes sensor name, status, hash key, range key, read and write throughput.

PostgreSQL (http://www.postgresql) was used to deploy the relational databases of the metadata and data dependencies at the platform. It is a powerful, open source object-relational database system. A collection of tables were created: *device_agent, device, device registry, device_type, sensor, sensor registry, sensor_type, and user.*

In order to integrate the NoSQL and the relational databases and be able to retrieve data corresponding to each sensor, a sensor's GUID, the primary key identifying a sensor,is used as a foreign key in the NoSQL database.

The platform was implemented as a web application deployed on the open Heroku using Ruby on Rails (http://rubyonrails.org/), providing RESTful web services. Ruby on Rails is a popular technology known for helping develop web applications and services. Data visualization was implemented using jqPlot (http://www.jqplot.com/). It is a configurable and rich "plotting and charting" plugin for the jQuery Javascript framework.

Sensor readings are either pushed or pulled from heterogeneous types of sensors (encapsulated in the device shown in Fig. 3), which are deployed over the buildings at CMUSV for smart space projects. Sensor types include FireFly, Sweetfeedback (CMUSV-created sensor), and JeeNode (http://jeelabs.net/projects/hardware/wiki/JeeNode).

As part of our sensor deployment, about 60 FireFly devices are deployed over the Building 23 at CMUSV. Each device pushes reading into our platform every 5 seconds, each reading carrying about 100 bytes of raw data. On the daily basis we are accumulating about 103.7MB of sensor data. The dataset is going toward to be a big dataset – how to handle its storage and query efficiently will be our future research.

Programs have been developed at the Sensor Gateway. Upon registration and configuration, the Sensor Gateway will automatically push sensor readings in the JSON format to the platform, or publish sensor readings and wait for the platform to pull the data at prescheduled time intervals. Below is a simplified sensor reading containing temperature data (e.g., pressure, humidity, timestamp):

[{"acc_z":1005,"bat":1005,"gpio_state":1,"temp":510,"light":889,"__class__":"Env_
Data","humidity":23,"motion":1005,"pressure":101740,"__module__":"pyfly","digital_
temp":241,"audio_p2p":1,"timestamp":1353441771000,"id":"1","acc_y":344,"acc_x":336}]

Data visualization tools are developed in our platform. The dashboard allows users to select sensor readings and visualize the data in a preferred manner. Fig. 7 shows a screen shot in a chart form supporting [Line, Pie] styles. Users can specify the start and end date for the readings.

A collection of data services have been developed in our platform. Here we briefly explain a simplified example service of user-defined sensor data reading. Given a sensor, the input parameters include *start_time* and *end_time*. They allow users to delimit the time frame (inclusive) of the sensor readings in which users are interested. The return values will be tuples. Instead of returning the entire set of sensor readings as raw data, the service will return the specified amount of aggregate readings or tuples. If there are fewer readings than the given number of tuples, then no results will be returned. Attributes unique to tuples are: average_timestamp, first, last, min, max, average.

Given Example 1 with a request on readings on sensor 1:
http://sdscmusv.herokuapp.com/sensor_readings/1

The response will be:

[{"id":"1","temp":71.0312,"timestamp":"1353441831000.0"},
{"id":"1","temp":71.7818,"timestamp":"1353441891000.0"}]

For Example 2 with a request on sensor 1 over a specified time frame:

http://sdscmusv.herokuapp.com/sensor_readings/1?start_time=
1353441891000&end_time=1453442011000&tuples=4

The response will be in the following JSON format:

[{"id":"1","average_timestamp":"1354431255526.1044176706
82730923694779116","first":71.7818,"last":70.20554,"min":53.01
68,"max":88.295,"average":70.3395828915661},{"id":"1","averag
e_timestamp":"1354580794706.82730923694779116465863453 8"
,"first":70.20554,"last":70.28059999999999,"min":69.37988,"max
":70.58084,"average":70.2916530120474}]

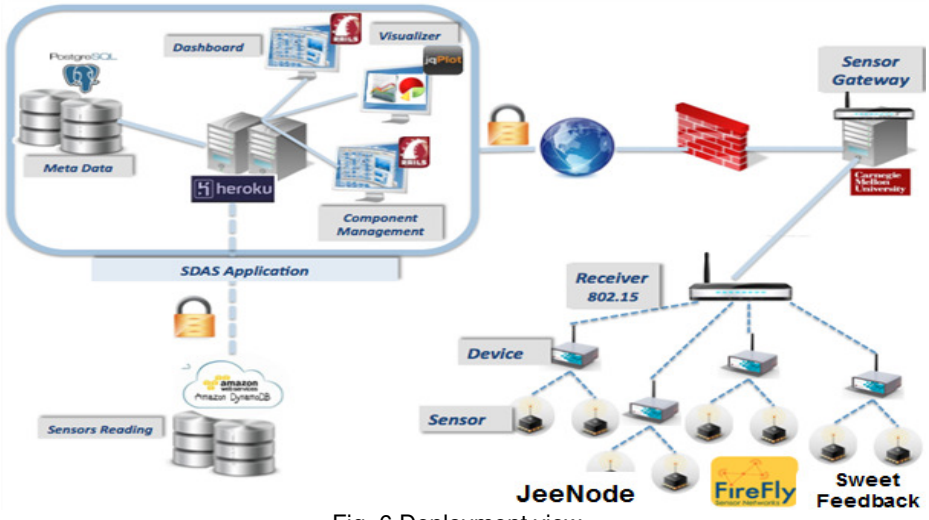

Fig. 6 Deployment view.

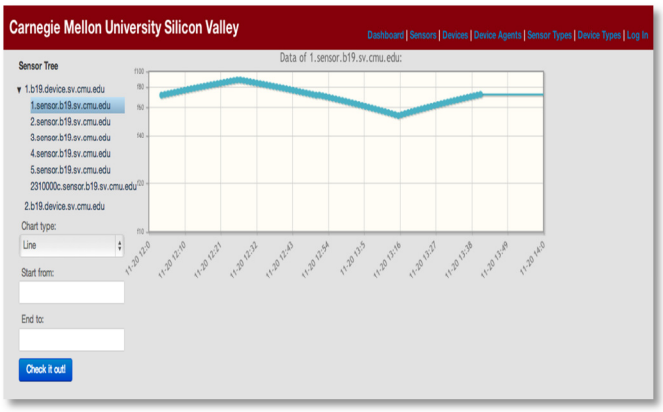

Fig. 7 Data visualization.

This data service is provided as a REST service, so that other applications can "plug-in" our platform to build new applications. For example, we are constructing an iPad application illustrating all sensors located in the buildings in our campus as well as the dynamic statistical temperature information at various rooms.

In order to study big data analytics that is important to our sensor data service platform, we also explored the option of leveraging in-memory database (SAP HANA: http://www.saphana.com) as an alternative to NoSQL database to store, monitor, and analyze streaming sensor data. Our preliminary experience shows noteworthy performance enhancement.

VII. Summary and Future Work

In this paper, we have presented the core design and prototyping of a Web 2.0 platform aiming to provide sensor data as a service, to allow users to discover reusable data and data analysis tools, and to integrate them into value-added workflows. Our work will contribute to an open community to support small research groups and individuals to contribute and share data sources and data services.

In our future work, we plan to develop version-controlled data analysis tools. In addition, we plan to study a business model, aiming to monetize sensor data by charging users to access and query the federated set of information from a community of contributed. Furthermore, we plan to evaluate and enhance our platform through a collection of ongoing smart space applications and disaster management applications under construction at the CyLab Mobility Research Center (CMRC) and Disaster Management Initiative (DMI) of Carnegie Mellon University Silicon Valley.

VIII. Acknowledgement

This project is partially sponsored by research gifts provided by SAP to Carnegie Mellon University.

VIV. References

[1] Y. Liu, Y. He, M. Li, J. Wang, K. Liu, L. Mo, W. Dong, Z. Yang, M. Xi, J. Zhao, and X.-Y. Li, "Does Wireless Sensor Network Scale? A Measurement Study on GreenOrbs", in Proceedings of *IEEE International Conference on Computer Communications (INFOCOM)*, 2011, Apr. 10-15, pp. 873-881.

[2] R. Nagpal, H. Shrobe, and J. Bachrach, "Organizing a Global Coordinate System from Local Information on an ad hoc Sensor Network", in Proceedings of *2nd International Conference on Information Processing in Sensor Networks (IPSN)*, 2003, Palo Alto, CA, USA, pp. 333-348.

[3] P.P. Rezaeiye and M. Gheisari, "Performance Analysis of Two Sensor Data Storages", in Proceedings of *2nd International Conference on Circuits, Systems, Communications & Computers (CSCC)*, 2011, pp. 133-136.

[4] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A Survey on Sensor Networks", *IEEE Communications Magazine*, 2002, 40(8): pp. 102-114.

[5] J. Yick, B. Mukherjee, and D. Ghosal, "Wireless Sensor Network Survey", *Computer Networks*, Aug. 22, 2008, 52(12): pp. 2292-2330.

[6] W.I. Grosky, A. Kansal, S. Nath, J. Liu, and F. Zhao, "SenseWeb: An Infrastructure for Shared Sensing", *IEEE MultiMedia*, Oct.-Dec., 2007, 14(4): pp. 8-13.

[7] K. Aberer, M. Hauswirth, and A. Salehi, "Infrastructure for Data Processing in Large-Scale Interconnected Sensor Networks", in Proceedings of *International Conference on Mobile Data Management*, 2007, Mannheim, Germany, May 7-11, pp. 198-205.

[8] K. Chang, N. Yau, M. Hansen, and D. Estrin, "SensorBase.org-A Centralized Repository to Slog Sensor Network Data", in Proceedings of *International Conference on Distributed Computing in Sensor Network (DCOSS)/Euro-American Workshop on Middleware for Sensor Networks (EAWMS)*, 2006, San Francisco, CA, USA, pp.

[9] P.B. Gibbons, B. Karp, Y. Ke, S. Nath, and S. Seshan, "IrisNet: An Architecture for a World-Wide Sensor Web", *IEEE Pervasive Computing*, Oct.-Dec., 2003, 2(4): pp. 22-33.

[10] A. Sheth, C. Henson, and S. Sahoo, "Semantic Sensor Web", *IEEE Internet Computing*, Jul./Aug., 2008: pp. 78-83.

[11] N. Mohamed and J. Al-Jaroodi, "A Survey on Service-Oriented Middleware for Wireless Sensor Networks", *Service Oriented Computing and Applications*, 2011, 5(2): pp. 71-85.

[12] S. Santini and D. Rauch, "Minos: A Generic Tool for Sensor Data Acquisition and Storage", in Proceedings of *19th IEEE International Conference on Scientific and Statistical Database Management*, 2008, pp.

[13] L. Mottola and G.P. Picco, "Programming Wireless Sensor Networks: Fundamental Concepts and State of the Art", *ACM Computing Surveys (CSUR)*, Apr., 2011, 43(3).

[14] *IEEE Std 1451.5™-2007, IEEE Standard for a Smart Transducer Interface for Sensors and Actuators— Wireless Communication Protocols and Transducer Electronic Data Sheet (TEDS) Formats*, 2007, IEEE Instrumentation and Measurements Society, TC-9, The Institute of Electrical and Electronics Engineers, Inc., New York, NY, USA.

[15] M. Botts and A. Robin, "OpenGIS ® Sensor Model Language (SensorML) Implementation Specification", Open Geospatial Consortium, Inc., 2007, Accessed on Feb. 15, 2013, Available from: http://www.opengeospatial.org/standards/sensorml.

[16] S. Madden, M. Franklin, J. Hellerstein, and W. Hong, "TinyDB: An Acquisitional Query Processing System for Sensor Networks", *ACM Transactions on Database Systems*, 2005, 30(1): pp. 122-173.

[17] S. Chen, P.B. Gibbons, and S. Nath, "Database-Centric Programming for Wide-area Sensor Systems", in Proceedings of *1st International Conference on Distributed Computing in Sensor Systems (DCOSS)*, 2005, Marina del Rey, CA, USA, Jun. 30-Jul. 1, pp. 89-108.

[18] OGC, "Sensor Observation Service (SOS)", Open Geospatial Consortium, Accessed on 12/30/2012, Available from: http://www.opengeospatial.org/standards/sos.

[19] C.A. Henson, J.K. Pschorr, A.P. Sheth, and K. Thirunarayan, "SemSOS: Semantic Sensor Observation Service", in Proceedings of *2009 International Symposium on Collaborative Technologies and Systems (CTS)*, 2009, Baltimore, MD, USA, May 18-22, pp. 44-53.

[20] A. Bröring, A. Remke, and D. Lasnia, "SenseBox-A Generic Sensor Platform for the Web of Things", *Mobile and Ubiquitous Systems: Computing, Networking, and Services*, 2012, 104: pp. 186-196.

[21] "TempoDB", Accessed on 12/31/2012, Available from: http://tempodb.com/features/.

[22] "Cosm", Accessed on 12/31/2012, Available from: https://cosm.com/.

[23] MicroStrain, Accessed on 12/31/2012, Available from: www.sensorcloud.com.