2013 IEEE 10th International Conference on Ubiquitous Intelligence & Computing and 2013 IEEE 10th International Conference on Autonomic & Trusted Computing

Supporting Personizable Virtual Internet of Things

Jia Zhang[1], Zhipeng Li[1], Oscar Sandoval[1], Norman Xin[1], Yuan Ren[1], Rodney A. Martin[2], Bob Iannucci[1], Martin Griss[1], Steven Rosernberg[1], Jordan Cao[3], Anthony Rowe[4]
[1]Carnegie Mellon University – Silicon Valley, USA
[2]NASA Ames Research Center, USA
[3]SAP, USA
[4]Carnegie Mellon University, USA
jia.zhang@sv.cmu.edu, rodney.martin@nasa.gov, bob@sv.cmu.edu, martin.griss@sv.cmu.edu, steven.rosenberg@sv.cmu.edu, jordan.cao@sap.com, anthony.rowe@cmu.edu

***Abstract*—**This paper reports on the design and development of an HTML5-powered Virtual Sensor Editor (VSE) over the Internet of Things cloud. VSE is a scalable tool that allows users to design virtual sensors with user-defined dataflow logic, by visually aggregating existing sensors, either physical sensors or other user-defined virtual sensors. VSE supports a real-time and historical visualization of sensor values and analytical studies, and is a cross-platform and customizable tool equipped with ability to support verifiable sensor data service composability. A discussion on design decisions is presented. Our preliminary work has been applied to NASA Ames' Sustainability Base for smart building monitoring. Preliminary performance and scalability study is also reported.

I. INTRODUCTION

Several government initiatives have focused attention on sustainability, energy efficiency, and the environment. One such initiative is NASA's Renovation by Replacement (RbR), which aims to replace outdated and inefficient buildings at NASA centers with new, energy-efficient buildings. NASA Ames Research Center won an RbR competition and worked with partners to design and build Sustainability Base, a 50,000 sq. ft. LEED Platinum certified high performance office building. In addition to using commercially available technologies, Sustainability Base aims to redeploy innovations and technologies originally developed by NASA for aerospace missions to monitor and control building systems while reducing energy and water consumption.

Technologies developed by NASA Ames' research partners will also be deployed to support these objectives, including those developed by Carnegie Mellon University - Silicon Valley (CMUSV). The ultimate vision of Sustainability Base is to provide a research testbed where different sustainable technologies and concepts can be implemented, tested, and demonstrated. The three primary research objectives involved in this vision are to reduce building energy consumption, to reduce building operating and maintenance costs, and to improve employee comfort levels. In this collaborative project, we focus on exploring how networked sensors can be better leveraged to contribute to these objectives.

The number of active Internet of Things (IoTs) (networked sensors is rapidly approaching 50 billion. The sensors are reporting their surrounding environments and helping people learn about the physical world in detail. For example, NASA Sustainability Base research initiatives are supported by over 2000 sensors of various types, deployed to help maintenance staff understand activities and conditions in the buildings including temperature, relative humidity, barometric pressure, and so on. Such data will help them to operate, monitor, and maintain the buildings, and additional value-added services may be derived based upon the deployed sensors.

However, it is not always a trivial task to find proper sensors and actuators to perform as needed. Even though current technology has allowed for real-time control actions to be taken based upon embedded physical sensors, many semantics-rich commands may not be able to be realized by individual sensors. For example, a more comprehensive query, such as to check the average temperature of the north wing of the second floor of the Sustainability Base, may involve the measurements of all temperature sensors deployed in the corresponding area. Furthermore, such an average temperature may be needed for long-term monitoring and exploration. Thus, its values should be stored and maintained to database, instead of always recalculating from comprising sensor data at runtime.

In a word, there should be a way to make a building "smarter" to be programmable, to allow users to integrate existing sensors with programming logic to query personalizable views of the building and analyze data. Toward this ultimate goal, this paper reports on our design and development of a *Virtual Sensor Editor*, a scalable tool to visually aggregate physical sensors with user-defined dataflow logic into virtual sensors. In our definition as shown in Fig. 1, a *virtual sensor* is an atomic component that provides sensor data service to the outer world. Relying on existing sensors, a virtual sensor will carry programmable workflow logic (using rules or formula) to present curated knowledge of environmental observations over a collection of sensors. Modeling embedded workflows as computable functions, virtual sensors can be composed to form comprehensive views of physical world, leveraging mathematical knowledge on functions as well as computability theory.

978-1-4799-2481-3/13 $31.00 © 2013 IEEE
DOI 10.1109/UIC-ATC.2013.48

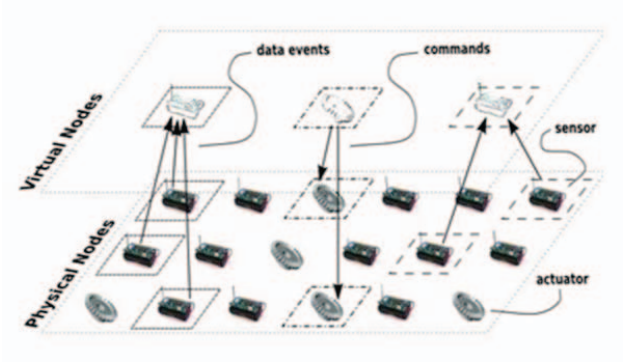


Fig. 1 Virtual sensor concept.

In contrast to various simulation tools over sensor net-networks such as TOSSIM [1] and OPNET [2], our virtual sensor editor focuses on its ability to allow users to create customizable sensors using dataflow logic. Note that virtual sensors are treated on par with physical sensors. First, virtual sensors can be used to compose value-added sensors. Second, virtual sensor readings are stored the same way as those for physical sensors. Third, a virtual sensor can carry the history of how it has become as it is now, which provenance will help analyze corresponding readings (i.e., computed values at the time).

To enable cross-platform development, we have adopted the HTML5 [3] technology to develop a web browser-based integrated development environment. The backend is the Sensor Data and Service Platform (SDSP) developed at CMUSV [4], which provides backbone support of sensor data service registration, discovery, and composition.

Our virtual sensor editor has the following four highlighted attributes: (1) native drag-and-drop: Instead of relying on third-party implementation of drag-drop function like jQuery library, we have utilized the native drag & drop feature provided by HTML5. (2) just-in-time evaluation: Users will obtain real-time feedback during the process of designing virtual sensors. (3) reusability: User can perform create, read, update and delete (CRUD) operation over their virtual sensors. Virtual sensors can be both data sources and data targets, and virtual sensors can be used to compose new virtual sensors. (4) predefined and customizable sensors: Users can specify customized rules to define customizable sensors.

The remainder of the paper is organized as follows. In Section 2, we discuss related work. In Section 3, we explain architectural decisions. In Sections 4 and 5, we present design and development of the virtual sensor development environment, respectively. In Section 6, we discuss virtual sensor composability. In Section 7, we present our performance and scalability study. In Section 8, we summarize conclusions and describe future work.

II. Related Work

Microsoft SenseWeb [5] provides a Web 2.0 platform for users to upload and access sensor data streams from shared sensors across the Internet. SensorBase [6] built a centralized data storage and management platform that allows users to publish and share ("slog") sensor network data using a blog-like approach. Global Sensor Networks (GSN) [7] adopts a scalable P2P model in favor of integrating heterogeneous sensor network technologies. In contrast, we have developed a cross-platform virtual sensor editor tool to allow users to dynamically mashup heterogeneous data sources to provide value-added sensor data services. In addition, virtual sensors are treated as first-class citizens.

Many 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 to help users collect and store sensor data from heterogeneous distributed sensors [8]. 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. In contrast to their work, we have proposed a sensor ecosystem concept, where virtual sensors become atomic service providers to provide customizable and programmable sensing data services. Virtual sensors are contributed back to persistent layers and are treated as composable data sources.

Sensor Observation Service (SOS) is a standard Web service specification, aiming to standardize the way of requesting, filtering, and retrieving sensors and sensor data to enhance sensor interoperability [9]. Some researchers explore how the Semantic Web can be integrated with a Sensor Web, such as SemSOS [10] and Semantic Sensor Web [11]. Some researchers, such as Liu et al. [12], study the scalability of sensor networks. SenseBox [13] introduces an autonomous computing unit encapsulating environment and REST APIs. In contrast to their work, our virtual sensors focus on the programmable dataflow embedded, as well as the data provenance associated with the dataflow.

In our earlier work, we developed an SOA-based Web 2.0 platform that allows users to view and federate heterogeneous sensor data sources through a Sensor Data and Service Platform (SDSP) [4]. In contrast, the work reported in this paper aims to provide a service provisioning layer for SDSP, to expose sensor data to the outer world in a (re)configurable and personalizable manner.

III. Design Decisions

Carnegie Mellon University (CMU) has developed SensorAndrew, the largest nation-wide campus sensor network [14]. Over ten thousand of various types of sensors have been deployed over the Pittsburgh campus as well as the Silicon Valley campus. The CMU – Silicon Valley campus has developed a Sensor Data Service Platform (SDSP) on top of the SensorAndrew infrastructure and its middleware, to provide sensor data and data service

publication, discovery, and composition [4].

Virtual Sensor Editor (VSE) aims to become an integral part of the SDSP, as a design tool to assist the design process of customizable sensors. In more detail, it should allow users to browse available sensors, pick up physical or virtual sensors in which they are interested, add rules under which a new 'virtual sensor' will work, and eventually persist the new virtual sensor if it works as expected.

Extracting conclusions from a user workshop, it is believed that the design tool should possess the following five key features: (1) platform neutrality: The tool should not be restricted to a certain platform. Meanwhile, mobility should be supported. (2) visualization: The tool should support real-time visualization of all sensors, physical and virtual. (3) in-time feedback: The tool should allow users to establish rules and alert policies, to realize real-time monitoring and management of smart spaces. (4) reusability: The tool should support recursive virtual sensor composition with formal validation facility. (5) scalability: The tool should be oriented to the community and support many users in designing, viewing, and managing their virtual sensors simultaneously. Towards fulfilling such user-defined goals, we have made the following architectural decisions.

AD1: Universal Unique Identifier (UUID)

Problem: We need to identify in a unique way every component in the structure of a virtual sensor.

Solution: An identification table is maintained at the server to keep a unique identifier for every sensor registered at SDSP, physical or virtual. At the current stage, version control of the virtual sensors registered is left for individual users to handle.

Alternatives: A1) unique identifier within the scope of a virtual sensor: Although guaranteeing no repetition of the identifiers within the context of a virtual sensor, managing identifiers in interdependent virtual sensors is challenging. A2) disposable identifiers: A new identifier is created every time a component is shared by multiple virtual sensors. In spite of reduced identifier management, this approach increases the complexity to recreate the relationships among components (which are store as relationships among Id's).

AD2: Virtual sensor definition stored in a serialized JSON string

Problem: A persistence mechanism to store created virtual sensors is needed.

Solution: Centralize the definition of a virtual sensor in a unique object that later will be serialized in a JSON string. Such a decision makes it easy to handle at the backend. However, the entire definition of a virtual sensor is serialized every time when a change occurs. This may lead to performance concerns if the number of virtual sensors becomes significant. Currently, the JSON object is saved in the browser's local storage; but it can be sent over the Internet to a backend service.

Alternatives: A1) backend database: All JSON objects are persisted to a backend database. In spite of robustness, it implies additional work to deal with atomic operations (i.e., queries) at development. A2) local storage key-pair: Store editing operations instead of the entire definition.

AD3: Use of separated layers for calculations

Problem: An efficient method is needed to control the overhead due to pulling the data from the sensors.

Solution: We have decided to separate the presentation, business logic and data pulling functions in different layers. The idea is to define a middleware data structure to hold a buffer of the readings of the sensors. Every time when the drawing canvas needs to recalculate the value of a sensor, it does not have to invoke a separate HTTP request to the server. Such a design decision will significantly reduce the overhead required to process the streaming sensor data.

Alternatives: A1) individual request: Every visual component in the canvas will control their individual requests for data to the server. This option however, may experience inefficiency because every component will open a separate HTTP request adding significant overhead. A2) push approach using web sockets: This option may be ideal because the server pushes data to a browser only when changes happen in sensor state. Due to time constraints and the need to re-configure the backend to support web sockets, it will be adopted as future work.

AD4: Use of JavaScript

Problem: A programming language is needed to allow users to define dataflow logic for virtual sensors.

Solution: We evaluated a collection of languages including JavaScript, Matlab, and Python. The main reason why we decided to adopt JavaScript is its ability to conduct real-time processing over data streaming in a web browser. In addition, JavaScript is compatible with other real-time frameworks like node.js. One potential concern though is its fragility to XSS attacks, which makes it important to sanitize code before being persisted to the server. One possible solution is to run code in a JS sandbox (i.e., Caja, a Google security project for "virtual iframes" http://code.google.com/p/google-caja/).

Alternatives: A1) Matlab or python: Many scientific users are more familiar with these languages. However, it is unlikely to achieve the level of performance that JavaScript can reach for processing real-time data. A2) domain-specific programming language: This option can provide a more compact, secure and powerful way to define the logic of virtual sensors. It is in our future work plan.

AD5: Language to define logic of virtual sensors

Problem: Users need a method to define the logic of a virtual sensor.

Solution: A high-level descriptive language is needed for users to precisely define the dataflow logic of a virtual sensor. Visual programming has been proved to be a

powerful way to ensure productivity [15]. In addition, we have embedded domain-specific libraries developed specifically for the Internet of Things.

Alternatives: A1) *ad-hoc formula builder*: This option refers to offering a plug-and-play way to define the logic of a workflow without writing code. How to build a comprehensive formula design tool with a complete list of proper functions remains a challenge. A2) ad-hoc predefined templates (widgets): The option refers to offering a graphical interface for users to define the logic of a virtual sensor without writing code. How to provide a meaningful quantity of widgets remains challenging.

IV. Virtual Sensor Ecosystem

A. Looped Sensor Ecosystem

Fig. 2 depicts our overall blueprint of a sensor ecosystem. Sensors from the physical world are registered into our Sensor Data Service Platform (SDSP) to become persistent and discoverable to the community. Through our browser-based design tool, users browse existing sensors and define their own rules to aggregate the sensors to provide a personalized view as a virtual sensor. The definitions of such user-defined virtual sensors are stored back to the SDSP backend server, so that they can be treated as reusable sensors to be further composed. Meanwhile, users can specify certain rules to control the physical world, e.g., adjust room temperature to prepare a comfortable meeting space.

As a testbed, we have realized a virtual sensor ecosystem at the Carnegie Mellon University Silicon Valley campus (CMUSV). As shown in Fig. 2(a), a number of Firefly [14] sensor devices designed by CMU are heavily deployed inside of a building. We have developed visualization techniques to continuously monitor building conditions, while the sensors send out readings every 5 seconds. As an example, Fig. 2(a) shows the real-time heatmap of the 1st floor of the building.

As shown in Fig. 2(b), all sensor readings are sent to our backend SDSP [4], which resides on the Amazon cloud, and stored in the SAP HANA in-memory database [16]. SDSP provides a collection of web services, which allow users to query sensor data registered. As shown in Fig. 2(a), our visualization tool can show either real-time sensor data, or show historical data based on user queries over a specified time frame.

As shown in Fig. 2(c), the VSE tool provides a web browser-based online editing environment to aggregate physical sensors into virtual sensors, by applying customized business logic. As also shown in Fig. 2(c), a panel on the left-hand side of the window displays all registered sensors organized in various categories such as temperature, motion, light, noise, etc. The right-hand side visual programming canvas allows users to drag and drop available sensors as components. Predefined templates are available for users to define dataflow rules to federate sensor data from comprising components. More details of the editor will be discussed in the next section.

After a virtual sensor is defined, its definition can be saved. Users may choose either a script view that allows further editing as shown in Fig. 2(c), or a time series view that triggers all real-time visualization of all composing sensors as shown in Fig. 2(d).

The virtual sensors are capable of behaving as physical sensors, while additionally achieving some goals that individual physical sensors cannot. Furthermore, users can define virtual sensors in a way to impact the physical world, e.g., adjusting room temperature to 72 degrees through a

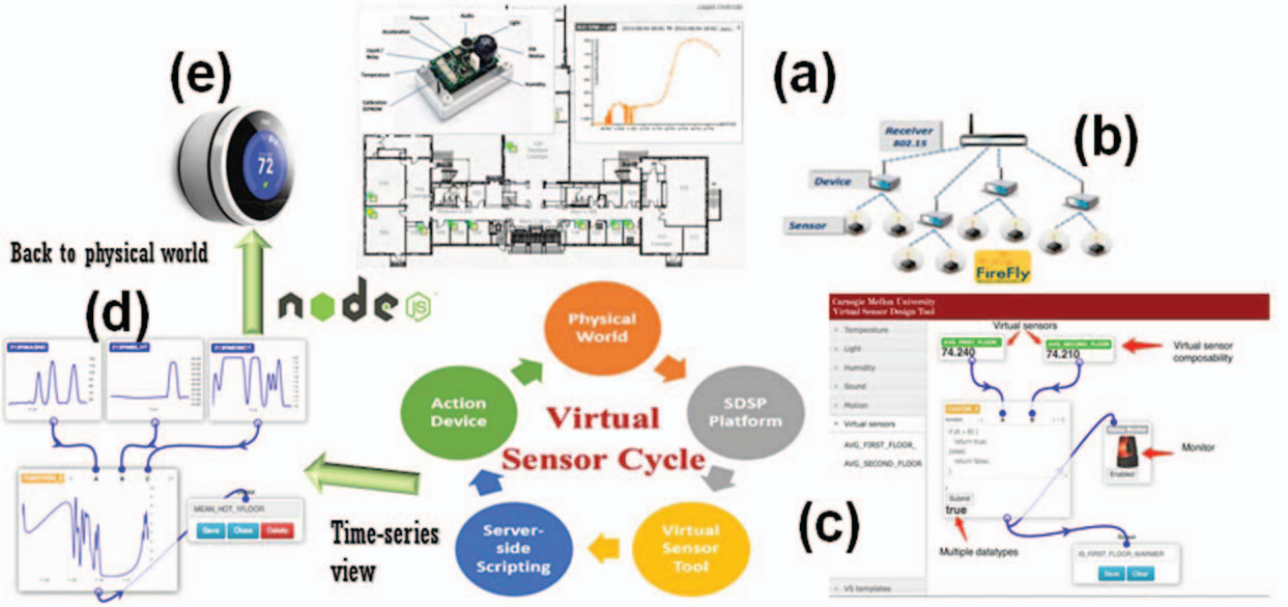


Fig. 2 Virtual sensor cycle.

NEST (http://nest.com, programmable thermostat) air condi-conditioner controller as shown in Fig. 2(e), or triggering an alarm if some predefined threshold is reached as shown in Fig. 2(c). In this sense, our tool extends the SDSP platform to form a "physical-virtual-physical" loop, analogous to the "sense-plan-act" robotic paradigm [17].

B. Virtual Sensor Editor

One core element of the sensor ecosystem is our closed-loop virtual sensor editor, which is an HTML5-powered browser-based sensor logic design tool. As formal definitions and the virtual sensor composability study will be discussed in Section VI, this section will introduce the design of the tool using highly simplified scenarios.

As shown in Fig. 3(a), a virtual sensor design template is a function box comprised of one to many input ports and output ports; the number of the ports can be configured and changed by users. Each input port is assigned an internal identifier (e.g., port "A"), which can be used in dataflow logic design. As long as a data link is established between a virtual sensor and a component sensor, the real-time data value of the latter sensor becomes an input to the former sensor and contributes to its display value. A library of statistical calculations is embedded into the tool and can be selected by users (e.g., function "mean") to speed up the design process. The dataflow logic of a virtual sensor can be defined in JavaScript. The button "Set" will trigger the execution of the defined workflow, so that users can evaluate and adjust the definition as needed. Note that a sensor box dragged to the canvas represents a running sensor data service, whose real-time readings are displayed. Different colors are used to notify the health of corresponding sensors: a red color indicates that the sensor did not send in data normally in the last 3 minutes; yellow indicates abnormal sensor data yielding in the last one minute.

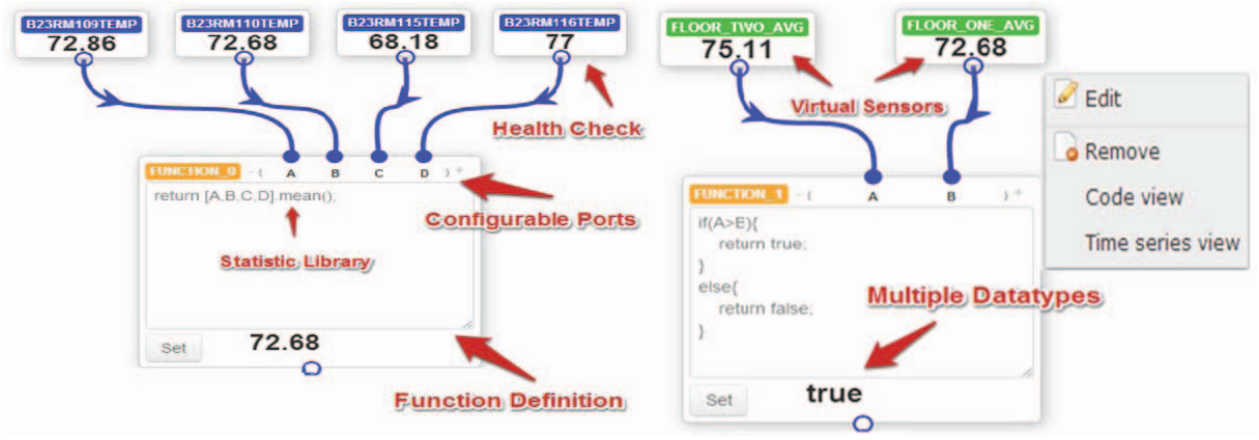
Virtual sensors can be recursively composed. As shown in Fig. 3(b), defined virtual sensors can be used as components to construct other virtual sensors. We use the *blue* color to represent physical sensors, and the *green* color to represent virtual sensors. A virtual sensor can adopt multiple data types as input and output sources. As a starting point, we allow integer, real number, and Boolean data types. As shown in Fig. 3(b), users can right click a sensor definition to edit its carried dataflow logic.

Defined virtual sensors can be registered to the SDSP platform and published as reusable sensor service providers. Their internal logic definitions are open to other users to review and update if needed. Thus, version control is one concern of virtual sensor storage. At this stage, every registered virtual sensor is associated with its contributing user identification as well as a timestamp.

V. System Design

We decided to realize the tool as an HTML5-based web service, aiming to realize the platform neutrality feature described in Section III. HTML5 [3] is the fifth version of the HTML standard, which structures and presents the content for the World Wide Web. HTML5 defines a single markup language that encourages interoperable activities by providing new markups and APIs for complex web applications. All existing main-stream browsers have started to support HTML5 to some extent, and most of the mobile devices like smart phones or tablets, support HTML5 quite well. This makes HTML5 very competitive from a cross-platform perspective. Another important reason that this tool chose HTML5 and JavaScript is that, compared with native applications programmed in Java or C++, or plugin-based web applications like Adobe Flash, the HTML5+Javascript approach provides universal access to all modern computing devices with Internet access, without worrying about issues during the installation phase.

The virtual sensor design tool is formed as a modularized layered application. Fig. 4 illustrates its internal



(a) virtual sensor (b) recursive virtual sensor construction

Fig. 3 Composable virtual sensors.

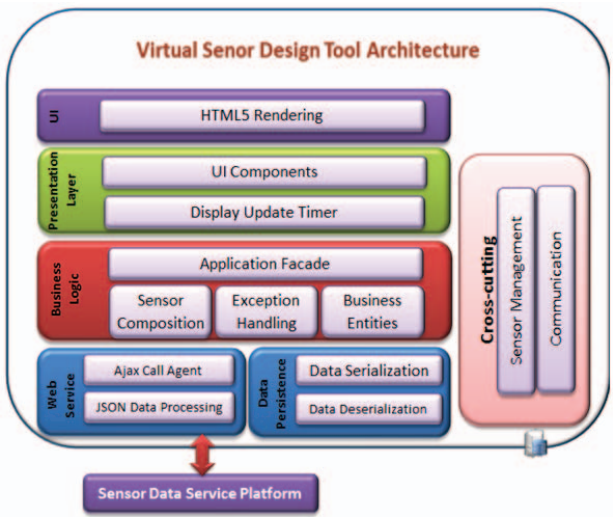


Fig. 4 Internal architectural design.

architectural design. Horizontal layers include UI layer, presentation layer, business logic layer, web service layer, and data presentation layer. A typical Model-Viewer-Controller (MVC) pattern is adopted. Vertical layers include a sensor management layer and a communication layer.

The UI layer is in charge of rendering a user interface in a web browser as an HTML5 webpage. We notice that some tags like <drag> or <audio> only work in an HTML5-compatible browser.

The presentation layer controls the content shown on the webpage. Two core functionalities are sensor value updating and sensor status updating, in a frequency of every three seconds. The presentation layer is a "reflection" of the logic relationship among sensors in a tree-like structure. The actual sensor data is to be retrieved on the fly from the business logic layer, which in turn handles application status such as network connections. Such a separation of concerns is critical to ensure the display performance of the tool, because a virtual sensor may not have to trigger to retrieve data from all sensors unless a detailed view is required.

The business logic layer sits behind the presentation layer to fetch sensor data and monitor sensor status. To provide such a façade for the presentation layer, the business logic layer conducts the following four core functions. First, it retains a live record of the relationships among sensors. Any editing change (for example, a user connects/disconnects two sensors) will be caught and stored. Second, it manages sensor composition. When a virtual sensor is dragged and dropped onto the canvas, the hidden sensors under the virtual sensor are managed by the business logic layer. Third, it is in charge of exception handling. Since the virtual sensor design tool allows customized functionality over input sources, the business logic layer will validate user input before the design is persisted and provide an error message if needed. Fourth, the business logic layer carries extensible embedded statics library (i.e., jsPlumb) to facilitate virtual sensor design. jsPlumb is a JavaScript library that helps draw "dataflow" on the canvas. Only sensors that are displayed on the canvas have corresponding jsPlumb objects.

The web service layer interacts with the backend Sensor Data Service Platform (SDSP) [4] through REST calls. It decouples the direct connection between sensors on the canvas and the backend web services. When dragging and dropping a sensor onto the canvas, the presentation layer will not launch an Ajax call to fetch data. Instead, the presentation layer in turn queries data from the business logic layer, which identifies such data from data structures maintained at the web service layer. Such a decoupling eliminates Ajax calls and increases response performance.

The data persistence layer is responsible for storing the definitions of virtual sensors into browser's local storage in two steps. First, it iterates through the "sensor tree" from the root node to obtain all attributes of the tree nodes. Second, it saves the tree structure as a string in JSON format. The local storage data will be interpreted by the webpage when refreshed; and virtual sensors can be reconstructed by interpreting such JSON strings.

VI. Composability Study

To realize the reusability feature described in Section III, virtual sensors should be reused in the same manner as physical sensors to construct new virtual sensors. In this section, we examine the composability of our virtual sensor concept toward formal reasoning. Composability refers to the capability to select and assemble physical or virtual sensors as components in various combinations into a valid observation to satisfy specific user requirements. A virtual sensor can be viewed as a container that represents an atomic abstract building block of the sensor service network. It is defined as a 4-tuple:

$\boldsymbol{vs} = < \boldsymbol{f}, \vec{\boldsymbol{\imath}}, \vec{\boldsymbol{o}}, \vec{\boldsymbol{m}}, >$, where:

$\boldsymbol{f}$ refers to a workflow; $\vec{\boldsymbol{\imath}}$ refers to a vector of input ports; $\vec{\boldsymbol{o}}$ refers to a vector of output ports; and $\vec{\boldsymbol{m}}$ refers to a vector of internal states.

Definition 1: An embedded workflow is a computable partial function:

$\boldsymbol{f} = \boldsymbol{X} \rightarrow \boldsymbol{Y}$, where:

$\boldsymbol{X} \subseteq \boldsymbol{S} \times \boldsymbol{I}, \boldsymbol{Y} \subseteq \boldsymbol{S} \times \boldsymbol{O}$

$\boldsymbol{S}$ is a non-empty set of states; $\boldsymbol{I}$ is a set of inputs, and $\boldsymbol{O}$ is a set of outputs.

Rationale: The definition of an embedded workflow as a computable function allows the use of the existing body of mathematical knowledge on functions, as well as computability theory [18]. Note that a physical sensor can be wrapped up as a virtual sensor.

The carried workflow $\boldsymbol{f}$ is iteratively executed by the virtual sensor triggered by incoming (streaming) events. At a given time point *n*, the virtual sensor accepts inputs from

its input ports (denoted as $\overrightarrow{\boldsymbol{\iota_n}}$) and the internal state from the previous time point (denoted as $\overrightarrow{\boldsymbol{m_{n-1}}}$), and produces outputs $\overrightarrow{\boldsymbol{m_n}}$ to be maintained for the subsequent time point as well as to its output port (denoted as $\overrightarrow{\boldsymbol{o_n}}$). Note that the input $\vec{\boldsymbol{\iota}}$, internal state $\vec{\boldsymbol{m}}$, and output $\vec{\boldsymbol{o}}$ all represent vectors of data. The execution of the workflow (mashup) carried internal of the virtual sensor at time point n can be specified as follows:

$$(\overrightarrow{\boldsymbol{m_n}}, \overrightarrow{\boldsymbol{o_n}}) = \boldsymbol{f}(\overrightarrow{\boldsymbol{m_{n-1}}}, \overrightarrow{\boldsymbol{\iota_n}}), \boldsymbol{n \geq 0}$$

Since each virtual sensor carries a workflow, composition of virtual sensors thus becomes composition of their aggregate workflows. Computability theory [18] declares that the set of computable functions is closed under composition; therefore, any number of virtual sensors can be composed if the composition of their workflows exists.

Definition 2: Given two workflows $\boldsymbol{f}: \boldsymbol{A} \rightarrow \boldsymbol{B}$ and $\boldsymbol{g}: \boldsymbol{C} \rightarrow \boldsymbol{D}$, their composition $\boldsymbol{h} = \boldsymbol{f} * \boldsymbol{g}$ exists *iff* $\boldsymbol{f}(\boldsymbol{A}) \subseteq \boldsymbol{C}$ and $\boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{f} * \boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{f}(\boldsymbol{g}(\boldsymbol{x}))$.

Here we focus on function composition of the workflows. Our concept of virtual sensor allows for *recursive composability*, or *hierarchical composability*, referring to the ability for a virtual sensor to be composed of other virtual sensors. It is a feature for creating and expanding a virtual sensor. The recursive composability is obtained by allowing us to link the output port of a virtual sensor to an input port of another virtual sensor.

Our composability study is reflected in the design of our tooling environment. A verification component is associated with each virtual sensor construct. When a (virtual) sensor is connected to another, the verification component is triggered to validate whether the composability requirements are satisfied or not.

VII. Scalability and Experimental Study

In Section III, we have explained the five key implementation features of our virtual sensor editor. The previous sections have covered how our design has fulfilled the requirements of platform neutrality, visualization, in-time feedback, and reusability. In this section, we discuss how we enhance scalability.

Since our tool is a community-oriented web tool, we have to take into consideration when the number of simultaneous users increases significantly. We have adopted three strategies. The first is to reduce the number of Ajax calls from the browser to the backend persistent layer to prevent network congestion. There are two options for multi-window visualization. One is to keep all Ajax windows independent of each other, in the sense that they all connect to the backend system individually. The other option is to establish a controller for a virtual sensor editor instance, which queries all backend sensors at the same time. By adopting the latter option, the enhancement largely reduces network traffic to the level of 1/N.

The second strategy of enhancing the scalability is a lazy evaluation strategy, meaning that we defer raw data processing from the reading stage to the callback method when it is needed. In more detail, the value of a virtual sensor is calculated at runtime upon a query. Unlike our storing all time series data for physical sensors after they are registered into our system, the values of a virtual sensor are stored every time they are queried. If a user requests the values of a virtual sensor during a past time period, calculations will be conducted and the values will be stored. The rationale is that it is possible that a virtual sensor is constructed but never used, so there is no need to store its values unless requested.

The third strategy is to adopt an in-memory database to speed up data retrieval and data analytics at runtime. We use SAP HANA [16] as a scalable solution to the volumes of time series data. Unlike other relational databases, SAP HANA provides column-based storage which supports real-time big data analytics. Note that although our current environment is built on HANA, the techniques developed are generic enough to be applied to other environments.

We have designed and conducted a series of experiments to evaluate the performance enhancement by adopting the first two strategies. Fig. 5 illustrates the performance changes before and after the enhancements. As shown in Fig. 5, the enhanced method remains stable when the number of concurrent users increases, all the way to 80 concurrent users.

We also ran simulations to evaluate the effectiveness of adopting strategy three, by comparing the performance of our system on top of SAP HANA and NoSQL DynamoDB. As shown in Fig. 6, we simulated different scales of sensor networks, which comprise from 1, 100, 1,000, 10,000, to 30,000 physical sensors. On each simulated sensor network, we simulated different numbers of concurrent virtual sensor editor users: from 1 to 50 users. Our simulation shows that, when the scale of the sensor network is moderate (comprising less than 1,000 physical sensors), our system scales well even when the number of concurrent users go up to 50. When the scale of a sensor network increases to 10,000, the average system response time goes up when

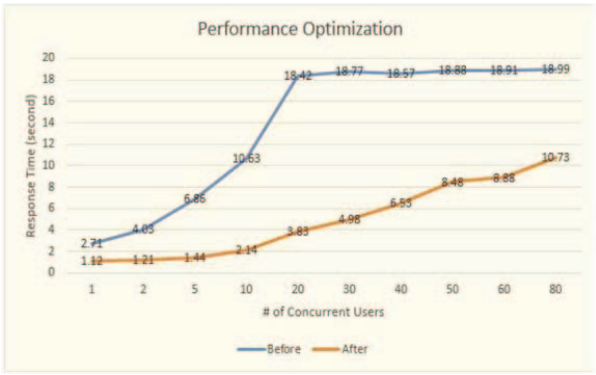


Fig. 5 Performance study.

serving more than 10 concurrent users. When the scale of a sensor network increases to 30,000, the scalability of the system becomes questionable. To better illustrate the experimental results, we keep only significant data in Fig. 6.

As shown in Fig. 6, the scalability of the system relying on HANA significantly surpasses that on DynamoDB. This symptom results from the specific pattern of frequent queries over certain types of sensors, e.g., a user is working on a virtual sensor involving multiple temperature sensors. Thus, the specific column-based query used by HANA yields better results.

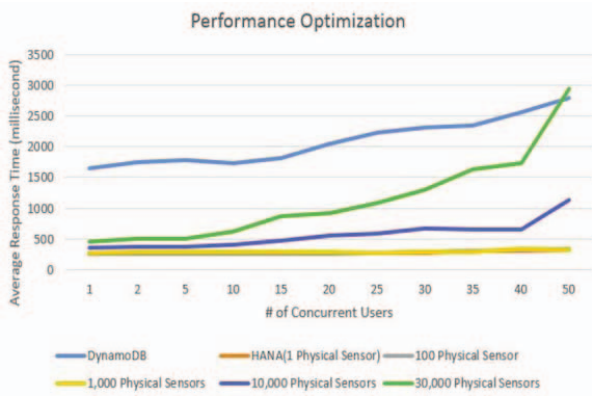


Fig. 6 Scalability study.

VIII. CONCLUSIONS

In this paper, we have presented the design and development of an HTML5-based virtual sensor editor tool, on top of our sensor data service platform. It supports real-time and historical display of sensor values for both physical and virtual sensors. It is cross-platform and customizable, and provides verifiable sensor data service composability.

In our future work, we plan to explore the web socket technique. Currently, our tool adopts polling to query the sensor database. A possible more efficient and scalable alternative might be to utilize the HTML5 web socket technique to fetch data. Clients will be called if and only if there are data updates on the server side. Such a strategy will further reduce the number of Ajax calls. In addition, we plan to enrich the notification widget collection. Currently, our tool has a single “monitor” tool to handle sound and visual effect. More widgets will be developed and added to the toolkit to simulate more physical objects. For example, an alarm clock, an LED screen or a message sender will be developed.

IX. ACKNOWLEDGEMENT

This project is partially sponsored by research gift provided by SAP to Carnegie Mellon University; as well as NASA grant NASA NNX12AQ95G and NNX13AD49A.

X. REFERENCES

[1]. P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications", in Proceedings of *the 1st ACM Conference on Embedded Networked Sensor Systems (SenSys)*, Nov. 5-7, 2003, Los Angeles, CA, USA, pp. 126-137

[2]. Riverbed, "Riverbed OPNET nCompass - Real-Time Network Visualization and Monitoring", accessed on: 11/7/2013, Available from: http://media-cms.riverbed.com/documents/Riverbed_OPNET_nCompass.pdf.

[3]. B. Frain, *Responsive Web Design with HTML5 and CSS3*. 2012: Packt Publishing.

[4]. J. Zhang, B. Iannucci, M. Hennessy, K. Gopal, S. Xiao, S. Kumar, D. Pfeffer, B. Aljedia, Y. Ren, M. Griss, S. Rosenberg, and A. Rowe, "Sensor Data as a Service - A Federated Platform for Mobile Data-Centric Service Development and Sharing", in Proceedings of *IEEE International Conference on Services Computing (SCC)*, Jun. 26-Jul. 2, 2013, Santa Clara, CA, USA, pp. 446-453.

[5]. 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.

[6]. 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.

[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*, May 7-11, 2007, Mannheim, Germany, pp. 198-205.

[8]. 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.

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

[10]. 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)*, May 18-22, 2009, Baltimore, MD, USA, pp. 44-53.

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

[12]. 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)*, Apr. 10-15, 2011, pp. 873-881.

[13]. 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.

[14]. A. Rowe, M. Berges, G. Bhatia, E. Goldman, R. Rajkumar, J.H. Garrett, J.M.F.M. Jr., and L. Soibelman, "Sensor Andrew: Large-Scale Campus-Wide Sensing and Actuation", *IBM Journal of Research and Development*, Jan.-Mar., 2011, 55(1): pp. 1-14.

[15]. "LabView", accessed on: Aug. 21, 2013, Available from: http://www.ni.com/labview/.

[16]. SAP, "SAP HANA", 2013, accessed on: Aug. 12, 2013, Available from: www.sap.com/HANA.

[17]. R.C. Arkin, *Behavior-Based Robotics*. 1998: MIT Press.

[18]. S.B. Cooper, *Computability Theory*. 2003: Chapman and Hall/CRC; 1st edition.