IoT Mashups with the WoTKit

Michael Blackstock, Rodger Lea
Media and Graphics Interdisciplinary Centre
University of British Columbia
Vancouver, Canada
mblackst@magic.ubc.ca, rlea@ece.ubc.ca

***Abstract*—Toward reducing barriers for developing applications for the Internet of Things, researchers have connected things to the web enabling the development of IoT *mashups*. While establishing a Web of Things for mashup development has been an important step forward, we believe that web-centric IoT toolkits have the potential to increase the use of Internet-enabled things further by increasing the pool of developers and applications that can take advantage of the connected physical world. In this paper we derive several key requirements for IoT mashup toolkits based on existing systems, past research and our experience with an IoT mashup toolkit called the Web of Things Toolkit (WoTKit). Unlike other systems, the WoTKit aims to address key requirements for IoT mashup developers in one system. From this experience we derive key lessons learned for the community toward improving toolkits for developing IoT mashups.**

I. INTRODUCTION

The Internet of Things (IoT) promises to enable novel applications in areas such as home automation, the environment, social networks, transportation, and health. To ease the development of large scale IoT applications, various high-end IoT or M2M platforms and toolkits have been developed. While their power and flexibility often affords the broadest range of possible solutions, this can come at the cost of complexity and a steep learning curve for web developers who aim to build IoT *mashup* applications: web applications using data and services available on the web. Mashups are often personalized, situational, short-lived, non-business critical applications developed using familiar web development tools and technologies [8]. We believe that innovative and novel new IoT services will be realized when tools are available that reduce barriers to entry for the development of this important class of applications.

Toward this goal, researchers have built on the ubiquity of web protocols and the Representational State Transfer (REST) architectural style of the web [1] to connect "islands of functionality" [2] calling this approach the Web of Things [3–5]. Things are identified by URIs and use a common protocol (HTTP) for stateless interaction between clients and servers. Using web protocols makes the creation of mashups possible allowing developers to combine data from both physical data sources and virtual sources on the web [6–8]. While leveraging the ubiquity of the web is an important step forward, we believe a further step, of developing web-centric IoT mashup toolkits has the potential to increase the use of Internet-enabled things further by increasing the pool of developers and applications that can take advantage of the connected physical world.

In this paper we derive the requirements of IoT mashup toolkits from several representative existing systems and our experience with a platform we've developed called the Web of Things Toolkit (WoTKit). Based on three years of experience with this system and other systems [9–11], we derive key lessons learned to share with the community. These lessons aim to shape the future evolution of WotKit but also, we hope, help shape and improve other toolkits and contribute to the vision of the IoT as a foundation for novel and innovative new applications and services.

II. BACKGROUND

Several high-end IoT or M2M systems such as ThingWorx [12] , AirVantage [13] and Axeda [14] provide many of the capabilities required for large scale IoT application development. Unlike more web-centric systems, these systems include support for non-HTTP protocols, device management, data management, and security. While these systems are flexible and powerful, they often require an investment by qualified developers to fully utilize their capabilities. Additionally, these systems generally do not have a focus on web development, and do not aim to enable a *connected* Internet of Things where things are shared on the Internet to create quick mashup applications. Rather these systems concentrate on customized, and often closed, business solutions for specific applications and organizations.

In contrast, there have been a number of web centric platforms for the IoT that aim to encourage rapid IoT application development. Pachube [15] aggregates collections of data streams called *feeds* to store information about sensors and the data they emit over time. The service also provides a directory of applications that provide processing, integration and data visualization capabilities. Developers can send data to the system, or set up device gateways to be polled by hosting a web server. The system supports the notion of *triggers*, where data from a feed can be sent immediately to a specified URL when a condition is met. A key feature of Pachube is the ability to share sensors and data, allowing others to take advantage of the integration work of others.

The Open Sen.se toolkit [16] aims to provide a set of applications for users to track data from themselves and their things. Users create dashboards called *Sense Boards* containing the user interface from a wide collection of plug ins installed by the user to enter, visualize and process data.

Figure 1. Laptop connected to Bluetooth Pulse Oximeter with WoTKit dashboard

Developers can easily integrate devices such as a suitably equipped Arduino by sending or receiving data from named input and output feeds that containing time stamped values using its RESTful API. Output feeds allow Sen.se to not only collect sensor data, but also control things. Unlike Pachube, Open Sen.se feeds can contain integers, float, boolean and string values. Currently, Sen.se devices cannot be shared with others on the system.

Paraimpu [17], [18] is a relatively new system now entering widespread testing[1] that aims to connect physical and virtual things to Web including arduino devices, social networks and other IoT platforms such as Pachube. Paraimpu provides a palette of configurable sensors, actuators and connections that provide processing capability such as filtering and mapping between sensor inputs and actuator outputs. Sensors and actuators can be public, allowing them to be shared between Paraimpu users who are also your friends[2].

ThingSpeak [19] supports a simple data model of channels that contains up to eight typed fields. Each field is visualized on the channel page of the site. ThingSpeak includes several applications that support web service integration, triggers, and integration with twitter. "Plug ins" that can display data from ThingSpeak in mashups can be created on the platform.

Tools and frameworks have emerged to ease the development of mashup applications [8], [20–23]. Yahoo Pipes is used to collect and process data using a dataflow-programming paradigm. Google Fusion Tables[3] can be used to collect and merge data for use in visualizations. Mixup [21] and QedWiki [23] from IBM are mashup tools that integrate information from the web at the presentation layer. Unlike web-centric IoT toolkits, these general-purpose mashup all require integration work to aggregate data from objects in the real world.

[1] Alpha testing is ongoing as of April 22, 2012.

[2] Paraimpu friends are Twitter followers.

[3] Google Fusion Tables http://www.google.com/fusiontables/public/tour/index.html# Accessed May 7, 2012

Each of these representative systems bring to light one or more requirements for a comprehensive IoT mashup toolkit. We believe that no one system to date has addressed them all. Pachube for example, supports only numeric sensor data, and its built in processing capability is limited to the ability to send triggers when a certain criteria is met. Pachube and ThingSpeak focus on supporting storage for sensor feeds, but do not support visualization dashboards as part of the toolkit. Sen.se supports built in visualization and processing components, but does not supporting sharing devices with others on the platform itself. Paraimpu supports easy integration, but no visualization, focusing on connecting input sensors to output actuators that include physical devices and social networks. Sen.se uses individual applications for processing and visualization, making it difficult for a developer to combine components, and write new processing components with the platform itself. Both ThingSpeak and Pachube have easy to use and RESTful APIs, but do not include flexible visualization dashboards or a processing engine out of the box.

In the next section we describe our experience with developing mashup applications using WoTKit with the goal of better understanding the complete set of requirements for an effective IoT mashup toolkit.

III. MASHUP EXPERIENCE

Like others, we envision IoT applications in a variety of areas such as home automation, the environment, social networks, transportation, and health monitoring. To date we have found the early implementation of WoTKit flexible and robust enough to begin using it for development. Initially we focused on collecting a wide variety of data to make it available on the system for web developers.

Researchers in the health domain have found WoTKit to be a useful prototyping tool [24]. The system was used to monitor the output from Bluetooth based pulse oximeters. These sensors were connected to a Bluetooth PAN host, which relayed the data to the WoTKit for visualization to facilitate patient monitoring during movement and transportation. The sensors and dashboard on a laptop for monitoring are shown in [24].

We have also used the WoTKit to prototype a mobile air quality monitoring application. To gather the needed data we wrote a simple script to query for updates to air quality information supplied on a public web site. This was then pushed into the WoTKit in a format that made it easy to process by a mobile application .

For transportation-related scenarios we have integrated several sources of location data. We have written simple applications for Android phones that relay the GPS coordinates of the user to the system periodically. We have also created a Google Latitude gateway to relay the location of users in the system for monitoring transportation patterns. To monitor several vehicles in a prototype dispatch application, the WoTKit's Processing Engine was used to aggregate sensors and display them on the Google Maps widget in the dashboard.

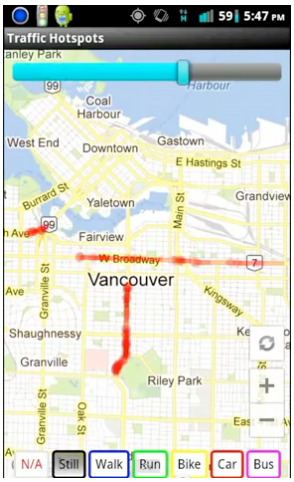

Figure 2. Traffic hotspots application aggregates transportation mode and location.

A recent application used multiple sensors on Android and iOS smart phones to send users' locations and their transportation mode inferred from both GPS and accelerometer readings as in [25]. An aggregation script then accessed the WoTKit API to view traffic congestion as shown in Figure 2. Unlike our previous transportation applications, this aggregation script performs periodic processing on a single feed that collects sensor data from all users, rather than processing the data as it arrived using the WoTKit Processing Engine.

To gauge the system's usefulness for home automation applications we have written simple gateways for Phidget sensors and actuators such as servo motors. More recently we have used the system to monitor activity using Zigbee based temperature, light and power sensors which deliver, via a gateway, data to visualizations supplied by the WoTKit on the web. We have integrated several hardware sensors including light sensors and custom power sensors connected to power bars and outlets.

In support of these mashups and other applications we provide a simple mechanism for posting sensor data and support a variety of sensor feeds ranging from physical infrastructure such as CPU, Network and power usage, through web data (scraped via tools such as Beautiful Soup[4]) such as airport arrival/departures, ferry and bus status upto softer sensor sources such as 'Tweets' and other social network feeds.

IV. TOOLKIT REQUIREMENTS

Based on our understanding of existing toolkits and our experience with the deployment of WoTKit mashups, we have identified seven abstract requirements that a web-centric IoT toolkit needs to address:

A. Meta-data and Data Storage

While it is clear that IoT platforms need to store thing data, e.g. sensor readings etc, it is perhaps less clear that they also need to store information about the things they are managing on behalf of the community of developers they serve, as well as some subset of the sensor data generated. Meta-data includes location, tags, and descriptions, enabling users to find things for their mashups. Sensor data includes numerical physical sensor readings such as temperature, light, power, or less structured data such twitter updates.

Tight integration of meta-data with sensor data and a simple means to query and use this meta level information is invaluable for rapid mashup development. While it is possible for meta-data to reside elsewhere, perhaps even in devices connected to the IoT, an ability to quickly search and connect to data sources provides an easy and intuitive way for application developers to locate and use thing data.

B. Integration

It can be difficult to integrate things with the web. Several toolkits, especially Sen.se and Paraimpu make it easy to integrate data from variety of devices such as Arduino-based projects, as well as web-based data sources such as social network feeds.. New gateways need to be developed that provide a web server interface to the thing. This means that the integrator needs to decide on the appropriate representations for things, decide on a security models for access and sharing. In some cases, it can be difficult to make the web presence of things available to the outside world because of firewalls.

To simplify this integration task, most toolkits serve as a hub for interacting with things. When the state of things changes, or periodically, a script or gateway can send information to the platform where it is saved and/or relayed to applications. There should be no need for each developer to set up a web server and decide on a suitable representation for the things they would like to integrate – the toolkit can provide this service.

C. Visualization

To make it easier to create useful and aesthetically pleasing visualizations, Se.nse and several general-purpose mashup tools provide a variety of visualizations of data from things 'out of the box'. Google and others have contributed visualization frameworks to make it straightforward for web developers to draw graphs, charts and maps. Unfortunately these frameworks depend on different representations for the data. The Google Chart Tools API[5] has its own JSON data representation, the jQuery Flot plug-in[6] uses another; Google Maps uses KML and the Google Maps API. To make it straightforward to generate visualizations, a toolkit should bridge the gap between the data representations of data from things to that needed for visualization frameworks.

While these frameworks have made it easier for developers to programmatically add custom visuals to their custom

[4] Beautiful Soup. http://www.crummy.com/software/BeautifulSoup/. Accessed March 15, 2012.

[5] Google Charts Tools. https://developers.google.com/chart/interactive/docs/index. Accessed May 10, 2012.

[6] Flot jQuery Plug-in. http://code.google.com/p/flot/. Accessed May 10, 2012.

applications, in many cases developers just need a quick way of visualizing different types of data. For example, a gauge can be used to visualize speed, a map to visualize location, and a bar chart to see power use over time. Toolkits should support a ready-to-use dashboard for quickly visualizing data from the platform, or to control various actuators in a flexible manner.

D. Control

Sen,se and Paraimpu not only collect data but can output data, to control or actuate things. This may include sending a message to a twitter feed, turning on or off an LED, or moving a servo. In addition to collecting data from sensors, it should be possible for a toolkit to control things and send data to the web. It may be desirable for a mashup to turn on the heat just before I arrive home in the evening, water plants when the soil becomes too dry, or tweet the current temperature in a greenhouse to open or close windows. To do this, the toolkit should support a means for transmitting to web-connected devices and data feeds.

E. Sharing

Pachube and Sen.se provide a capability for users to share their integrated thing data streams and, in some cases, other toolkit components. A key enabler for the web of things is to permit others to access and use the things that have been published publicly on the web. It should be possible for users to make use of things that others have shared and to make use of things in their own applications, perhaps in ways unanticipated by the owner of the thing. This requirement means we need a sophisticated set of mechanisms to publish and share things - and ways to find and access those things.

F. Processing and Alerts

An easy to use information processing capability to support simple data processing is included with Sen.se as well as toolkits like Yahoo Pipes. In many scenarios, data from multiple sensors needs to be combined and processed. In some cases, alerts need to be sent when an interesting event occurs; Pachube includes such an alerting capability. One challenge in providing data processing capability in a toolkit is the need to find a balance between ease of use, expressivity and generality in a programming language. Ideally a toolkit should allow developers to create their own processing facilities, either by combining existing processing modules, or creating new ones. Toward addressing this issue for developers, visual programming languages have been used to ease the development of mashups. Yahoo Pipes allows developers to easily combine Internet data from various sources to provide new sources of data and simple visuals. Some systems such as Sen.se and Paraimpu provide configurable processing components that can be dropped into the system as is, but these components are relatively stand alone, making it difficult to combine them to process data in unanticipated ways.

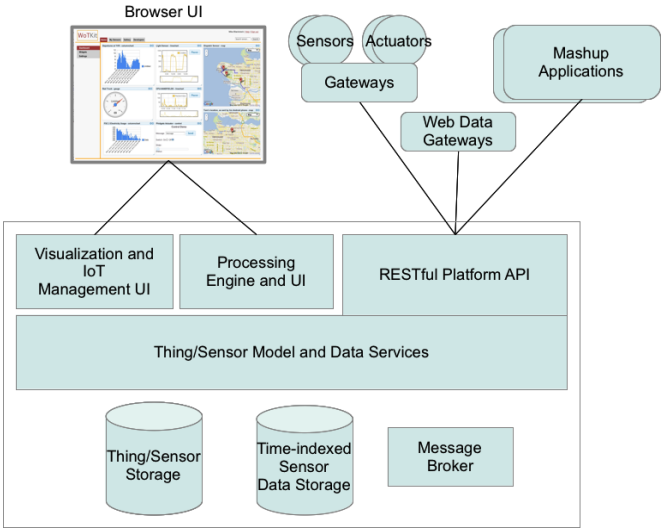

Figure 3. WoTKit architecture

G. Application Programming Interface

Existing IoT Mashup systems support developers by providing an API to the things they integrate, making it possible to scale the application from a quick prototype to more advanced application services. Since a toolkit cannot provide all of the functionality needed by an application, it is important to support the integration of external applications and components. Once an application is prototyped, a suitable API allows developers to create their own custom applications, visualizations and processing components, or integrate new sources of data into the system. Ideally, the programming interface should be RESTful to allow web developers to take advantage of the extensive tools and techniques available.

These abstract requirements have evolved out of an analysis of existing systems and our own work, which has spanned wide area UbiComp platforms [9], [26], IoT platforms [11] and most recently the WoTKit. In the next sections we describe the WoTKit and its design and implementation.

V. WoTKit Overview

WoTKit is a Java web application that leverages the Spring Framework[7] a popular development framework for enterprise applications. The data model consists of *sensors* with fields describing either a sensor or actuator connected to the system. Sensors are associated with time stamped sensor data containing multiple typed fields. To deliver sensor data between components, a standard Java Messaging Service (JMS) broker called Active MQ[8] is used for moving new data between components for fast processing and control applications. The high level architecture of WoTKit is shown in Figure 3.

Included in the main web application is a data model for managing user's dashboards and visualizations. Visualizations

[7] Spring Framework http://www.springsource.org/. Accessed May 10, 2012

[8] Active MQ http://activemq.apache.org/. Accessed May 10, 2012

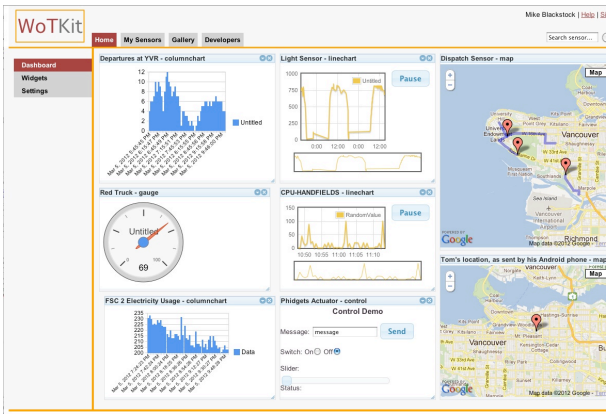

Figure 4. Example WoTKit dashboard.

are associated with sensors to create "widgets" that are added to the user's dashboard for rapid visualization of sensor data. When a dashboard is displayed in the browser, it requests the user's dashboard configuration and dynamically draws the various visuals containing sensor data.

A key difference between WoTKit and other web centric-toolkits is in how visualization and processing facilities are delivered. Rather than providing these components as applications or plug-ins, they are provided as core system facilities. Like others, the WoTKit serves as a sensor data aggregator, visualization, remote control and processing tool. It aggregates data from a variety of sensors, and allows simple control messages to be passed to actuators. WoTKit allows users to quickly find and subscribe to sensor data of interest, process data and visualize the data using widgets on a dashboard. By providing these built in facilities, we do not prevent developers from creating their own applications; but we supply easy to use baseline functionality for users to get up and running quickly.

A. Sensor Gallery

The WoTKit sensor gallery provides a way for users to search for sensors of interest that they have contributed themselves, or that others have made available on the system. Developers can elect to subscribe to a sensor, adding it to their sensor list making it available for generating visualizations and processing pipes.

B. Gateways and Thing Integration

Integration gateways for the WoTKit are generally simple scripts that (optionally) register discovered sensors, gather data from the sensors they serve, and push data into the system either periodically or when the data changes. Because these gateways are web clients, not servers themselves, they can be located behind firewalls. They typically consist of only a few lines of code to register themselves, update their state, and get control messages.

To illustrate, the following shell script posts the current CPU use of a PC to a sensor to the default data fields called value. This script assumes the CPU sensor has already been registered on the system.

```
#!/bin/sh
while (true) do
      cpu=$(uptime | sed 's/.*load averages: \([0-
9]\.[0-9]*\).*/\1/')
      echo "average cpu use: "$cpu
      curl -user {user}:{password} -data "value="$cpu
http://{host}/api/sensors/{user}.cpu/data
      sleep 60
done
```

More advanced gateway scripts can send data to the platform by posting additional named fields containing numeric or string data specified by the user.

Actuators can also be connected to the system. A simple dashboard visualization containing a radio button switch, message field and a slider is used to signal connected actuators. To receive signals behind a firewall, actuators subscribe to control messages sent to the sensor. Using HTTP long polling, thing gateways listen for control messages. When an application or dashboard controller widget sends a signal, the device gateway receives a JSON encoded message, and does the appropriate thing such as turning on or off a switch, or moving a servo.

C. Dashboard

For quickly visualizing sensor data, the WoTKit provides a JavaScript based dashboard for quickly and easily displaying a variety of sensor data visualizations and control components as illustrated in Figure 4. The dashboard supports the generation and placement of *widgets*: a combination of a thing with a chosen visualization. To support widgets, the system supports representations needed by the visualization code hosted on the browser platform. The system currently leverages Google Maps, the Google Chart Tools and Flot; the WoTKit client side dashboard framework can incorporate other JavaScript visualizations as needed.

D. Processing Services

An event-based data processing subsystem called the Processor is provided with the WoTKit. Sensor data is processed as it is pushed into the system from gateway components. The main purpose of the system is to allow users to generate new, and in some cases, higher-level sensor information from lower level sensor data in a straightforward manner. The primary interface is a visual programming environment that leverages the WireIt toolkit[9] for JavaScript-based visual languages presenting an interface similar to Yahoo Pipes. The programming paradigm is a data flow where processing *pipes* made up of connected *modules* are built by end users to generate new sensor data from other systems in the system.

A management page provides a list of pipes that the user currently has running. Using this page, users can start, stop and edit the pipes. Administrators can manage all pipes for all users on the system. To develop a new pipe, or edit an existing pipe, the visual programming interface allows users to drag and drop modules to the main pane and then connect them with wires as illustrated in Figure 5.

[9] WireIt: A JavaScript Wiring Library. http://neyric.github.com/wireit/. Accessed May 15, 2012

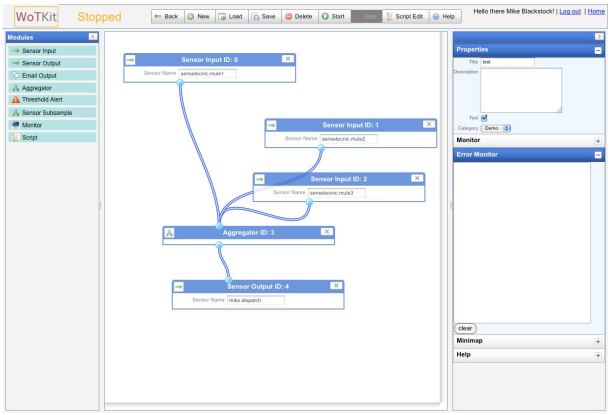

Figure 5. WoTKit processor pipe editor based on WireIt.

Once the user saves and executes the pipe, it is first checked for errors and "compiled" by instantiating pipe modules in the server. The system subscribes to data sent into any sensor input modules, and based on the configuration of these modules in the pipes, executes the pipe on behalf of the user.

Our implementation uses a multithreaded execution scheduler to process sensor data as it is added to the system. When a user creates a pipe, modules are instantiated as shown in Figure 6. Typically a pipe will include one or more sensor input modules to subscribe to data from sensors and add it to the execution queue. The multithreaded scheduler waits on this queue, retrieves the next message containing data, looks up the next module instance in the pipe and calls the module's `process()` function to process the message. The executing module may then add additional messages to the execution queue before it exits.

To date we have implemented several modules in the following categories:

Input/output. These modules are the primary integration point with the rest of the system.

Processing. The system currently supports two modules for processing: an *aggregator* module takes data from two sensors, adds a new field to the data to indicate the originating sensor, and sends this aggregated data to its output connection. A *threshold* module sends a single message to an output connector when the value of the input data meets a condition. To reset the output, another input called the trigger is used. We anticipate adding more built in processing components for averaging, filtering and other useful primitives.

Testing and Debugging. To see data as it flows through a pipe, a Monitor module can be added. Data that is sent to these modules appears in a pane on the visual editor when the pipe is executing. This can be used for testing and debugging pipes under development.

Alerting. To allow users to send alerts, the system includes an email module that will send emails to a configured email address containing message data. This can be used in tandem with the threshold module to send an email when a certain condition is reached. We anticipate the system supporting other integrations such as RSS feeds, social network feeds, SMS messaging and others.

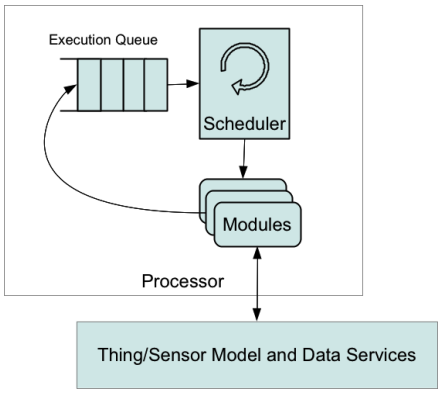

Figure 6. Processing engine architecture.

User scripting. Finally, we include the ability for end users to write their own modules using a scripting language, currently Python. By convention, the script takes input from an input dictionary, executes some code, and then puts any output into an output dictionary for downstream processing. Once a script is found to be useful, the user may save copies of these scripts for use in other pipes. Having this capability allows the user to extend the built in primitives with new modules as needed.

E. RESTful Service Interface

The WoTKit has a RESTful API for things allowing applications to control things, get the historical data from things and register new things and their meta-data with the system. Applications register sensors with the system by POSTING a JSON representation of the sensor to the following URL.

```
http://{host}/api/sensors/{sensor-name}
```

The sensor representation consists of the sensor name, a long name used for the user interface, the location of the sensor, whether it is a public or private sensor, and the information about the fields of data used by the sensor.

The primary APIs are for sending and receiving data into the system using the sensor. Gateways POST fields to the data URL

```
http://{host}/api/sensors/{sensor-name}/data
```

While applications GET data from the same URL, specifying query parameters for the range of data required by the application and the representation. The system currently supports CSV, KML (specific for location sensors), HTML and a JSON format for direct use by Google visualizations.

F. WoTKit Summary

By providing these services, the WoTKit addresses the requirements outlined in Section 4 as follows:

- Data storage for things meta-data such as a description, location, and the data things produce is included. WoTKit can store meta-data including the name, description, and location of sensors and actuators as well as multiple non-

numeric (string) sensor data values in a single sensor reading.

- To share things, WoTKit users can specify whether their things are public or private, allowing users to take advantage of the integration work of others using the Sensor Gallery.
- Things are integrated by writing simple HTTP client scripts that either push (POST) new data in to the system, or poll for actuator control commands.
- A built in visualization dashboard is included making it easy for developers to view a variety of visualizations and add them to their applications.
- The processing engine allows developers to create new processing pipes combining built in modules, and the creation of new modules using a scripting language.
- An easy to use API to register new sensors, publish data, and retrieve data in several formats including CSV, JSON and HTML is included.

Like Sen.se, the WoTKit provides a flexible dashboard; processing components are included and sensor feeds can contain numeric and string types, however, the WoTKit focuses less on the integration of applications into the platform user interface, rather providing basic built in visuals and processing components. Like Pachube and ThingSpeak, WoTKit serves as an aggregator of sensor data, allowing developers to push data into the system for others to use. Unlike Pachube, WoTKit and others support non-numeric feed values and unlike Pachube, the WoTKit includes a more comprehensive event-based processing engine. Like Paraimpu the WoTKit can create pipes that act as connectors, connecting sensor data to actuators such as email and other output modules, however, the power of the pipes, however the processing engine's flexibility allows it to be used for sensor data processing as well as connectors.

VI. LESSONS LEARNED

Our experience with the WoTKit, and examining similar systems has highlighted the need for certain key features and apporaches that provide the right balance between ease of use and flexibility for web developers.

Data Schema and Representations. From our experience, it is important to have a very flexible representation for things and the data they generate. Like Pachube we decided to give every sensor a location, however, in some cases, we found this did not make sense: a mobile phone's location changes constantly; a social network feed doesn't necessarily have a location. Initially our system focused on supporting numeric sensor data values only, but we quickly realized we needed more flexibility. Based on this experience, we have chosen to take a very generic approach to data. Our need to support both "hard" physical sensors such as temperature, speed and light readings, as well as soft sensors such as information from the web and social networks implies a sensor data model that allows both very simple schema initially that can be extended to more complex schema in the platform.

Sharing. Several WoT toolkits support sharing things and their data to allow users of the system to take advantage of the integration work of others. This is an important facility; some developers will not be interested or able to integrate all of the things needed for their applications. A facility for sharing things with friends using social networks [10] or the public is critical and may cause a network effect – the more things on a given toolkit platform, the more valuable that platform is to the users of that platform.

Component Model. In addition to sharing things, it is useful to add new visuals and processing components to a toolkit over time. In some systems visuals and data processing are both exposed as dashboard components, other systems expose processing components as "connectors", while others integrate both visual and processing components as "apps". In the WoTKit, we have found it necessary to integrate visualization components (widgets) differently from processing pipe components, and unlike Sen.se, for example, believe these components are sufficiently different to warrant separate toolkit integration points.

Push or Pull Sensor Data. Today most web-centric toolkits can poll data from things periodically, or wait for things to push data into the system. One disadvantage to polling is that the infrastructure needs to poll for data even if there is no change, just to ensure up to date historical data is available. Another is that gateways cannot be behind firewalls. Gateways that push data to the system, may send data that no application or user is interested in. To reduce the frequency of polling, it may be worthwhile having thing gateways maintain some history, and respond to short-term historical requests on demand. Similarly, a gateway can be configured to send data only when there is an interested subscriber. Currently the WoTKit relies on gateways to send data to the system regularly, however, we intend to add support for subscriptions and polling for data in future versions.

Processing Model. Just as data can be pulled or pushed into a toolkit, it is possible to process data as it is "pushed" or added to the system, or when it is queried by applications. Push processing allows alerting and filtering data before it is saved but does not support aggregation and processing of historical data. Pull-based processing allows historical data to be process when it is queried and potentially cached for periods of time. We believe a toolkit should support both mechanisms. Future versions of our processing engine will support both models, providing a similar visual programming interface for both event and query-based data processing.

Batteries Included. In this paper we outlined some of the basic requirements for a WoT toolkit. From our experience with the WoTKit, we found that when end users can quickly capture data and visualize it, they are willing to invest time in exploring further the capabilities of the toolkit such as data processing, alerting capabilities and the API. Essentially, a toolkit with "batteries included", ie with sufficient features and functionality to get the user up and running quickly, is necessary. While we believe this to be true, it is not clear what the minimum set of visualization processing and integration components are required. We intend to continue building

applications and leverage techniques from non-IoT mashup toolkits to answer this question (e.g. [20–22]).

VII. Conclusions

The WoTKit and other IoT mashup toolkits offer the promise of easing application development for web developers. By lowering barriers to developing mashups these toolkits will encourage the uptake of the IoT. Based on our experience with the WoTKit we outline several lessons learned to better serve mashup application developers, improve our toolkit and inform the design and implementation of other toolkits to support the vision of the IoT as a foundation for novel and innovative new applications and services.

VIII. Acknowledgements

We would like to acknowledge the contributions of Vincent Tsao, Nima Kaviani, Ian Henry, Tom Hazelton, Walter Karlen, and Ashkan Deylami for the testing and development of the WoTKit and Adrian Friday for his comments and suggestions. Funding has been provided by NSERC and Nokia Corp.

IX. References

[1] R. T. Fielding and R. N. Taylor, "Principled Design of the Modern Web Architecture," *ACM Transactions on Internet Technology (TOIT)*, vol. 2, no. 2, pp. 115–150, May 2002.

[2] W. Drytkiewicz, I. Radusch, S. Arbanowski, and R. Popescu-Zeletin, "pREST: a REST-based protocol for pervasive systems," in *Mobile Ad-hoc and Sensor Systems, 2004 IEEE International Conference on*, 2004, pp. 340– 348.

[3] D. Guinard, V. Trifa, and E. Wilde, "A resource oriented architecture for the Web of Things," in *Internet of Things (IOT), 2010*, 2010, pp. 1–8.

[4] D. Guinard, V. Trifa, F. Mattern, and E. Wilde, "From the Internet of Things to the Web of Things: Resource Oriented Architecture and Best Practices," in *Architecting the Internet of Things*, D. Uckelmann, M. Harrison, and F. Michahelles, Eds. New York Dordrecht Heidelberg London: Springer, 2011, pp. 97–129.

[5] S. Duquennoy, G. Grimaud, and J.-J. Vandewalle, "The Web of Things: Interconnecting Devices with High Usability and Performance," in *Embedded Software and Systems, 2009. ICESS '09. International Conference on*, 2009, pp. 323–330.

[6] T. Mikkonen and A. Salminen, "Towards Pervasive Mashups in Embedded Devices," in *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2010 IEEE 16th International Conference on*, 2010, pp. 35–42.

[7] D. Guinard, V. Trifa, T. Pham, and O. Liechti, "Towards physical mashups in the web of things," in *Proceedings of the 6th international conference on Networked sensing systems*, Pittsburgh, Pennsylvania, USA, 2009, pp. 196–199.

[8] Jin Yu, B. Benatallah, F. Casati, and F. Daniel, "Understanding Mashup Development," *Internet Computing, IEEE*, vol. 12, no. 5, pp. 44–52, 2008.

[9] M. Blackstock, R. Lea, and C. Krasic, "Evaluation and Analysis of a Common Model for Ubiquitous Systems Interoperability," in *Proceedings of the 6th International Conference on Pervasive Computing*, Berlin, Heidelberg, 2008, pp. 180–196.

[10] M. Blackstock, R. Lea, and A. Friday, "Uniting online social networks with places and things," in *Proceedings of the Second International Workshop on Web of Things*, New York, NY, USA, 2011, pp. 5:1–5:6.

[11] M. Blackstock, N. Kaviani, R. Lea, and A. Friday, "MAGIC Broker 2: An open and extensible platform for the Internet of Things," in *Internet of Things (IOT), 2010*, 2010, pp. 1–8.

[12] "ThingWorx – The 1st Application Platform for the Connected World," *ThingWorx – The 1st Application Platform for the Connected World*. [Online]. Available: http://www.thingworx.com/. [Accessed: 13-Mar-2012].

[13] "AirVantage M2M Cloud Platform." [Online]. Available: http://www.sierrawireless.com/productsandservices/AirVantage.aspx. [Accessed: 12-Mar-2012].

[14] "Axeda Application and Data Integration Platform," *Axeda Application and Data Integration Platform*. [Online]. Available: http://www.axeda.com/. [Accessed: 13-Mar-2012].

[15] "Pachube - The Internet of Things Real-Time Web Service and Applications," *The Internet of Things Real-Time Web Service and Applications*. [Online]. Available: http://www.pachube.com/. [Accessed: 12-Mar-2012].

[16] "Open Sen.se Feel, Act, Make sense," *Feel, Act, Make sense*. [Online]. Available: http://open.sen.se/. [Accessed: 12-Mar-2012].

[17] "Paraimpu - The Web of Things is more than Things in the Web:," *Paraimpu.crs4.it*. [Online]. Available: http://paraimpu.crs4.it/. [Accessed: 12-Mar-2012].

[18] A. Pintus, D. Carboni, and A. Piras, "The anatomy of a large scale social web for internet enabled objects," in *Proceedings of the Second International Workshop on Web of Things*, New York, NY, USA, 2011, pp. 6:1–6:6.

[19] "The Internet of Things - ThingSpeak," *The Internet of Things - ThingSpeak*. [Online]. Available: https://thingspeak.com/. [Accessed: 12-Mar-2012].

[20] J. Yu, B. Benatallah, R. Saint-Paul, F. Casati, F. Daniel, and M. Matera, "A framework for rapid integration of presentation components," in *Proceedings of the 16th international conference on World Wide Web*, Banff, Alberta, Canada, 2007, pp. 923–932.

[21] J. Yu, B. Benatallah, F. Casati, F. Daniel, M. Matera, and R. Saint-Paul, "Mixup: a development and runtime environment for integration at the presentation layer," in *Proceedings of the 7th international conference on Web engineering*, Como, Italy, 2007, pp. 479–484.

[22] K. Ito and Y. Tanaka, "A visual environment for dynamic web application composition," in *Proceedings of the fourteenth ACM conference on Hypertext and hypermedia*, Nottingham, UK, 2003, pp. 184–193.

[23] "IBM - Service Oriented Architecture (SOA): QED Wiki." [Online]. Available: http://www-01.ibm.com/software/solutions/soa/newsletter/jan07/article_QEDwiki.html. [Accessed: 07-May-2012].

[24] W. Karlen, M. Blackstock, and J. M. Ansermino, "Location independence in patient monitoring," in *Anesthesia & Analgesia*, Las Vegas, USA, 2011, vol. 113, p. 37.

[25] S. Reddy, M. Mun, J. Burke, D. Estrin, M. Hansen, and M. Srivastava, "Using mobile phones to determine transportation modes," *ACM Trans. Sen. Netw.*, vol. 6, no. 2, pp. 13:1–13:27, Mar. 2010.

[26] M. Blackstock, R. Lea, and C. Krasic, "Toward wide area interaction with ubiquitous computing environments," in *Proceedings of the First European conference on Smart Sensing and Context*, Berlin, Heidelberg, 2006, pp. 113–127.