WoTKit: A Lightweight Toolkit for the Web of Things

Michael Blackstock, Rodger Lea
Media and Graphics Interdisciplinary Centre
University of British Columbia
Vanouver, Canada

{mblackst, rlea}@magic.ubc.ca

ABSTRACT

In this position paper, we discuss our experiences with a lightweight Web of Things (WoT) toolkit and use those experiences to explore what an effective WoT toolkit looks like. We argue that while the WoT community has experimented, like us, with a variety of toolkits, it hasn't yet found one that appeals sufficiently to a broad range of developers. This failure, we believe, is hindering the adoption of the WoT and the growth of the community. We conclude the paper with a set of open questions, which, although not exhaustive, are aimed at opening up a community discussion on the needs of developers and how best the community can meet those needs and so further the adoption of the WoT. In essence, we believe that the time may be right to begin to agree on some basic functionality and approaches to WoT toolkits.

Keywords
Internet of Things, Web of Things, Mashup Toolkits, REST

1. INTRODUCTION

Building upon the vision of a connected Internet of Things, the Web of Things (WoT) [7, 15] aims to leverage web protocols and technologies to facilitate rapid construction of web applications exploiting real world objects. By using the representational state transfer (REST) architectural style of the world wide web [6], things are identified by URIs and a use a common protocol (HTTP) for stateless interaction between clients and servers. Given the simplicity of this approach and the prevalence of web expertise, it is perhaps surprising that the WoT and associated applications have not developed as rapidly as imagined.

While we feel there are a number of technical reasons for this, we believe that a significant barrier is the lack of simple and lightweight toolkits (or even a single toolkit) that strikes the right balance between functionality and simplicity. We believe that the community has been seduced by the open and prevalent nature of web technologies into focusing less on tools and support for developers, believing that by simply exploiting Web technologies, the WoT will naturally evolve.

Obviously, we are not claiming that the WoT community has not explored platforms and toolkits; there have been many excellent research projects and industrial efforts. Rather, our position is that we, as a community, need to do more to understand what will enable a significant uptake in the WoT and how we, as a community, can enable that uptake.

To help frame the discussion we present our work on the *WoTKit*, a lightweight toolkit and platform (run as a service) that provides a simple way for end users to find, control, visualize and share data from a variety of things. Based on our own experiences with previous systems [3–5] and using the WoTKit we present a set of high-level requirements that have driven the evolution of our toolkit with a core goal of making it suitable for the rapid development of WoT applications. While our work and that of others in the community has explored many of the issues around how to enable the WoT, we certainly don't have all the answers; in fact, our experiences have raised many questions about our approach. In the spirit of a position paper, and with the goal of generating discussion, we end by outlining these questions and some initial thoughts. We look forward to further discussion at the workshop.

2. BACKGROUND

High end industrial IoT and M2M systems such as ThingWorx [17], Axeda [2], and AirVantage [1] address many of the requirements for building IoT applications, however, their focus is on providing a comprehensive set of tools for building build end-to-end solutions, often using private data sources. They do not focus on web enabling things and capabilities that can be shared for web developers to easily build simple solutions.

Researchers have proposed the use of the REST architectural style supported by the web to connect things to the Internet [7, 15]. Ideally this would begin to address the lack of interoperable application layer standards. Others have demonstrated the use of Web tools and techniques such as the use of browsers, search engines, caching and scripting languages alike JavaScript to create mashups that integrate with the real world [8, 9]

Realizing that the use of the web alone doesn't fully address the needs of web developers, solutions such as Pachube [12], Open Sen.se [11], ThingSpeak [16], Paraimpu [13, 14] and others including our own work presented here have emerged. Pachube aggregates data feeds for public use and provides a directory of applications that provide processing, integration and visualization capabilities. Open Sen.se also aims to provide a set of tools to collect and display data about themselves and others from sensors. Users create *sense boards* containing the output from various applications installed by the user to visualize and process data. Paraimpu is used to connect physical and virtual things to Web and to social networks. Paraimpu *connections* provide processing capability such as filtering and mapping between sensors and actuators. ThingSpeak supports some basic visualization of

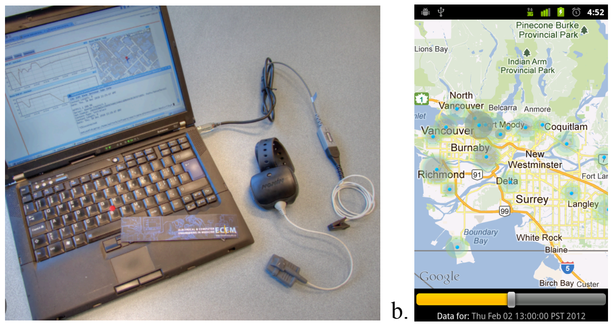


Figure 1a. Laptop connected to Bluetooth Pulse Oximeter with WoTKit dashboard [10]. b. Android air quality application shows pollution data for the region overlaid on Google Maps.

historical data and data processing when data is pulled from the system.

These solutions aim to close the gap between the needs of developers and the emerging Web of Things to increase the number of users able to take advantage of the connected physical world. Such toolkits can afford incremental innovation, allowing advanced end users and developers to build on others' work, and find uses for sensors and data that may not be obvious or suitable for only a small number of end users.

Based on our experience developing a sophisticated web-based ubiquitous computing integration platforms [3], a light weight platform for web of things applications [4] and integration with social networks [5] we believe that a key enabler of the WoT is the development of low-end or lightweight toolkits toward increasing the use and popularity of web enabled things.

Like other systems, the WoTKit system presented here meets several basic requirements. The difference between WoTKit and others is really on how these facilities are delivered. Rather than providing them as external applications or plug-ins, they are provided as core system facilities out of the box. The intent is not to prevent developers from creating their own applications for use with WoTKit, but rather to provide easy to use baseline functionality to get up and running quickly. In the following section we discuss some of the basic requirements for a WoT toolkit.

3. WOT TOOLKIT REQUIREMENTS

Toward gaining an understanding of WoT toolkit requirements, we aimed to support applications in a variety of domains such as home automation, environmental monitoring, social networks, transportation, and health. To date we have found the early implementation of WoTKit flexible and robust enough to begin using it for development. Initially we have focused on collecting a wide variety of data to make it available on the system and to exercise the platform.

Researchers in the health domain have found WoTKit to be a useful prototyping tool [10]. 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 Figure 1a. This project took advantage of the WoTKit's ease of integration and visualization capabilities.

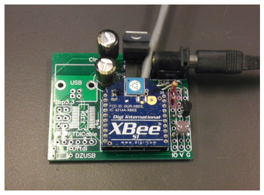


Figure 2. Zigbee based current, light and temperature sensor.

We have also used the WoT kit to prototype a mobile air quality monitoring application. To gather the needed data we wrote a simple script to query for updates from city-owned air quality sensors 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 as shown in Figure 1b. This project exercised the RESTful APIs that were needed to retrieve the air quality data for the native Android application. Using the sharing facilities of WoTKit, this public data is also available for other applications.

For transportation-related scenarios we have integrated several sources of location data. We have written several applications for Android phones that relay the GPS coordinates and inferred transport modes 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 Processing Engine supplied with the WoTKit was used to aggregate sensors and display them on the Google Maps widget in the dashboard.

To assess the system for home automation applications we have written simple integration gateways for Phidget sensors and actuators such as servomotors. Control gateway scripts leverage the API to listen for control messages sent from the platform to the listening actuator. More recently we have integrated Zigbee based temperature, light and power sensors to monitor activity as shown in Figure 2.

A variety of small projects have used the WoTKit to monitor computing resources and other infrastructure. This includes simple scripts to send the CPU and network use of one of our sensing computers running several gateway scripts, making it easy to monitor the health of machines remotely and send alerts when critical levels are reached. Web scraping tools such as Beautiful soup[1] have made it straightforward to write simple 'virtual sensor' gateways that push information from the web into the system for visualization and reuse by applications. We have used these techniques to track the number of airport arrivals and departures by the hour, the electricity use in several buildings on our campus, the overall electricity demand of the UK and other interesting web-based data making them available to our dashboards and any toolkit application that may make use of them.

Lastly, we have explored social networks as sensor feeds writing a variety of simple scripts that feed tweets, Facebook updates and other social networking activity into our system for use in mash-up applications. As we have built these applications and evolved our platform towards WoTKit, we have tried to identify essential

[1] Beautiful Soup: We called him Tortoise because he taught us. http://www.crummy.com/software/BeautifulSoup/. Accessed March 15, 2012.

features and functionality that we know that WoTKit needs to support well. These have included:

- Simple integration between a variety of things, both physical and virtual, and the toolkit.
- Easy to use visualizations of data from a thing, and user interface to control things remotely, using the web.
- An easy to use information processing capability for simple data processing and alert generation.
- The capability for users to share their integrated things and other toolkit components with others.
- The ability to scale up simple prototypes to more advanced applications by providing a comprehensive and easy to use API.

In the following section we describe each of these requirements in more detail.

3.1 Integration

It can be difficult to integrate things with the web. New gateways need to be implemented to provide a web server interface to the thing. This means that the integrator needs to decide on the appropriate web representations for the things, security models, and other issues. In some cases, it can be difficult to make the thing available to the outside world because of firewalls.

To simplify this integration task, we suggest the toolkit itself should serve as the hub for thing interaction. When the state of things changes, a script can send information to the toolkit where it is saved for applications. There should be no need for every developer to set up a web server and decide on a suitable representation – the toolkit can provide this service.

3.2 Visualization

Creating useful and aesthetically pleasing visualizations on the web can difficult not only for end users but also web developers. To address this, companies like Google and open source efforts have contributed visualization frameworks to make it straightforward for developers to draw graphs, charts and maps. Unfortunately these frameworks depend on different data representations. To make it straightforward to generate visuals, a toolkit should bridge the gap between the data representations of data from things to that needed for visualization frameworks.

3.3 Processing

In many scenarios, data from multiple sensors needs to be combined and processed. In others, alerts need to be sent when an important event occurs. One challenge in providing processing capability is the need to balance ease of use, expressivity and generality in a programming language. Toward addressing this issue for Web developers, Yahoo Pipes[2] introduced an easy to use visual programming language for the development of mashups. This approach has been shown to be useful when integrated with Pachube[3]. Another approach is to provide simple configurable processing components that can be dropped into the system as is. Toolkits should have suitable general purpose processing capabilities for simple application development.

3.4 Sharing

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.

3.5 Advanced Application Support

Since a toolkit cannot provide all of the functionality needed by an application, it is important to support the integration of external components. Once an application is prototyped, a suitable API allows developers to create their own visualizations, 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 to web developers today.

4. WOTKIT DETAILS

The WoTKit aims to address these basic requirements. It serves as a sensor data aggregator, visualization, remote control and mashup tool. In this section we discuss WoTKit gateways, the dashboard, the Processor service, sharing features and RESTful API that the system exposes to applications and things.

4.1 Gateways

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

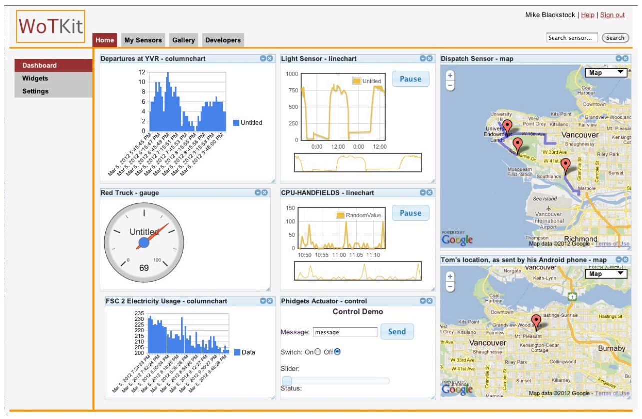
To illustrate, the following simple example shell script posts the current CPU use of a PC to a sensor to the default data fields called *value*. This script assumes the *cpu* sensor has already been registered on the system.

```
#!/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 receive data from sensors, and send additional data to the platform in different named fields. Note that gateways themselves need not concern themselves about the representation of things on the web, leaving this to the system.

[2] Pipes: Rewire the Web http://pipes.yahoo.com/pipes/. Accessed March 12, 2012.

[3] Fetch Pachube Feed on Yahoo Pipes http://pipes.yahoo.com/pipes/pipe.info?_id=RG78xpPB3RGfyBwQ6ycw5g. Accessed April 16, 2012



WoTKit
Home
My Sensors
Gallery
Developers
Dashboard
Widgets
Settings
Departures at YVR - columnchart
Light Sensor - linechart
Dispatch Sensor - map
Red Truck - gauge
CPU-HANDFIELDS - linechart
FSC 2 Electricity Usage - columnchart
Phidgets Actuator - control
Tom's location, as sent by his Android phone - map
Control Demo
Send
Pause

Figure 3. Example Dashboard with a variety of sensor data.

The simplest way for an application to retrieve data is to issue a GET request to the appropriate sensor specifying the representation desired and query parameters. For HTML:

```
http://{host}/api/sensors/{sensor}/data?tqx=out:ht
ml
```

Actuators can also be connected to the system. A simple remote control widget containing a text field, radio button and a slider can be used to signal connected actuators. To receive signals behind a firewall, actuators subscribe to control messages sent to the sensor and using HTTP long polling, listen for data on the subscription. When an application or dashboard controller widget sends a signal, the device gateway receives a JSON encoded message, and performs the appropriate action such as turning on or off an LED, or moving a servo.

4.2 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 shown in Figure 3. The dashboard supports the generation and placement of *widgets*: the combination of a chosen thing and visualization. To support widgets, the system supports representations needed by the visualization code hosted on the browser platform. The system currently leverages the Google Chart Tools[4] and Flot[5] a jQuery plug in for generating visualizations; the WoTKit client side dashboard framework can incorporate other visualizations as needed.

4.3 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

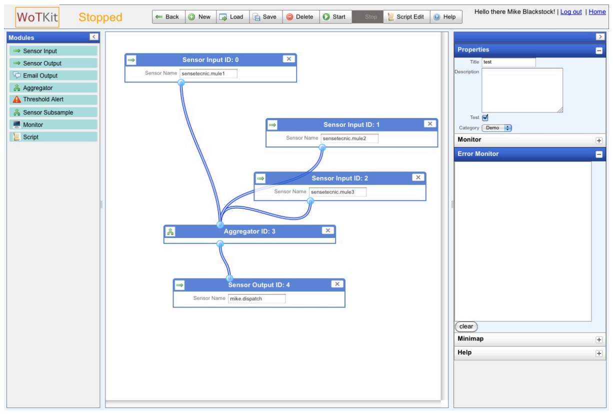


Figure 4. Processor component pipe editor based on WireIt

toolkit[6] 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 sensor inputs in the system.

A management page provides a list of pipes that the user is currently executing or working on. Using this page, 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 4.

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. To date we have implemented several modules:

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

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.

Processing. The system currently supports two modules for processing: an *aggregator* takes data from two sensors, adds a new field to the data to indicate the origin 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 additional components for averaging, filtering and other useful primitives.

Integration. To send alerts, the system includes an email module that will send emails to a configured email address containing sensor data formatted using a template. This can be used in tandem with the threshold module to send an email when a certain condition is reached for example. We are working on other integrations such as RSS feeds, social network feeds, SMS

[4] Google Chart Tools http://code.google.com/apis/chart/interactive/docs/index.html Accessed March 6, 2012.

[5] Flot Javascript Plotting Library http://code.google.com/p/flot/ Accessed March 6, 2012.

[6] WireIt – The JavaScript Wiring Library http://neyric.github.com/wireit/ Accessed March 12, 2012.

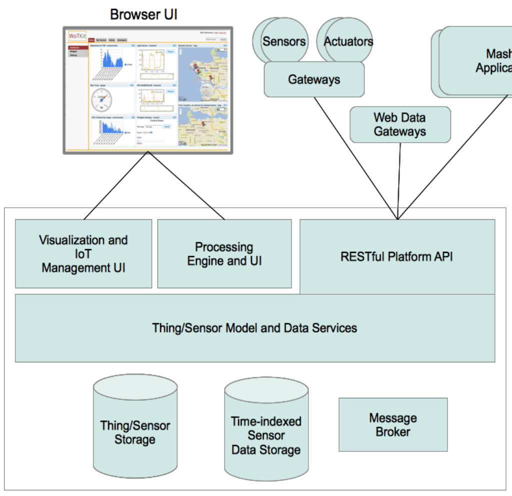


Figure 5. WoTKit architecture

messaging and others.

User scripting. Finally, we include the ability for end users to write their own modules using 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. The user may save copies of scripted modules for use in other pipes. This capability allows users to extend the built in processing capabilities with new functionality as needed.

4.4 Sharing

To enable sharing things, the WoTKit provides a searchable gallery. Users can search things by name, or the contents of their description. All things have a global location to allow users to easily find things of interest nearby. Users then subscribe to the things of interest in the gallery to build visualization widgets and processing pipes.

4.5 RESTful Service Interface

The WoTKit has a RESTful API for things allowing applications to control things, get historical data from things, and register new things 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 representation consists of a sensor name, a long display name for the user interface, the location of the sensor, whether it is a public or private sensor, and information about the fields of data supported by the sensor including type, display name and units.

The primary APIs are used to send and receive sensor data. Typically 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 time range and the representation. The system currently supports CSV, KML (specific for location sensors), HTML and a JSON format for direct use by Google visualizations.

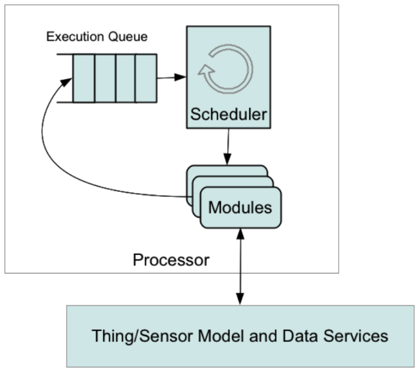


Figure 6. Processing Engine architecture

5. DESIGN AND IMPLEMENTATION

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*, and time stamped sensor data. The Apache Active MQ message broker[8] is used to deliver sensor data between components to support low latency processing and control applications. A high-level architecture diagram is shown in Figure 5.

Included in the main web application is a data model for managing user's dashboards and visualizations. Visualizations are linked 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 configuration of containers of widgets and dynamically draws the various visuals: bar charts, line charts, maps, gauges, and controls.

The Processing Engine processes sensor data as it is sent to the system by sensor gateways. Our initial implementation of this system uses a multithreaded execution scheduler. The modules of all executing pipes, implemented as Java objects, are instantiated in the Processing Engine. Typically a given pipe will contain at least one sensor input module to receive data from a sensor by listening for messages on a corresponding broker topic. Data received from a sensor is put into the execution queue as shown in Figure 6. The multithreaded scheduler waits on this queue for messages, retrieves the next message, and then looks up the next pipe module instance to process the message. The executing module may then add additional messages to the execution queue before it exits. While this simple scheme does not ensure fairness between pipes and pipe owners, we have found it useful to establish the basic execution architecture; we aim to enhance this engine over the coming months using a prioritized queue that more evenly spreads processing engine CPU resources between users and their pipes.

6. OPEN QUESTIONS

Based on our experience with WoTKit, and by examining similar systems, several questions arise toward the continued evolution of lightweight toolkits for the Web of Things.

[7] The Spring Framework http://www.springsource.org/. Accessed March 12, 2012.

[8] The Apache ActiveMQ Message Broker, http://activemq.apache.org/. Accessed March 12, 2012.

Naming Things. The WoTKit uses two namespaces: a globally unique numeric thing identifier, and a human-readable account-relative name. Our previous work used a hierarchical namespace allowing things to be organized and addressed in groups for ease of organizing and finding things in containment and administrative relationships [16]. Some systems allow things to be tagged to find related things. In a system used to manage many related and unrelated things what is the most appropriate way to name things?

Thing Representations and Schema. To date, each toolkit provider has their own abstractions and representations for things and their data. Pachube calls streams of data "environments" or "feeds" containing DataStreams and DataPoints. Open sen.se and ThingSpeak use the concept of "channels" for storing data from people or things. Our toolkit calls things "sensors" containing sensor data. What are the best abstractions for things and their data (current and historical)? Do all things have a location? Does all sensor data have a scalar numeric value? Given the variety of things, what are the appropriate field names, types, and order for their data? How do we make these systems flexible enough to support any type of data while allowing fast and flexible queries and filtering?

Integration Points. It is useful to add new visuals and processing components to a toolkit over time. What are the best integration points for such capabilities? 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, visuals are dashboard "widgets", while processing components are "pipe modules". With different possible integration approaches, what is the best practice for both usability and easy of integration by developers?

Push or Pull? It is possible for a WoT platform to poll data from things periodically, or wait for things to push data into the system. One disadvantage is that the infrastructure needs to poll for data even if there is no change to ensure historical data is available. When things push data in, they can be located behind firewalls, but may send data to the system that no one is interested in. The best approach may be a hybrid: things respond to data requests, only sending data when there is an interested subscriber.

Processing Model. Event based processing allows immediate alerting real time updates and filtering data, but does not permit aggregation and processing of historical data. Should a toolkit support both processing approaches, and if so, how should they be presented to the toolkit user?

Sharing. Most WoT toolkits support sharing of things and their data to allow users of the system to take advantage of the integration work of others. What are the appropriate sharing mechanisms for the WoT? Should they leverage social networks, or are the groups and relationships between things different from the social relationships of their owners?

Toolkit Integration and Standards Given the increasing number of WoT toolkits available from both the industrial and research communities, the time seems right to begin work toward agreement on basic functionality and approaches. Initial work toward providing gateways between systems is a good step forward. It allows users to leverage the things available as well as the strengths and different approaches of each toolkit to create more compelling applications and components. Additional steps could be taken to standardize the abstract model and service interfaces of WoT toolkits that are essential for a well-connected Web of Things.

Batteries Included. In this paper we outlined some of the basic requirements for a WoT toolkit. However, it is not clear what specific visualization, processing, and integration components are required. For us, this raises the question as to what does it mean for a WoT toolkit to "include batteries". What are the key elements for a basic toolkit comprehensive enough for basic application development?

7. CONCLUSIONS

The WoTKiT and others provide the basic requirements for lightweight toolkits: easy integration, visualization and processing components and a RESTful API. Based on recent experience with the WoTKit system and past experience with IoT and ubicomp platforms, we have raised several questions for the WoT community around the abstractions, technical approaches, and future directions of toolkits toward greater uptake of the WoT.

8. REFERENCES

[1] AirVantage M2M Cloud Platform: http://www.sierrawireless.com/productsandservices/AirVantage.aspx. Accessed: 2012-03-12.

[2] Axeda Application and Data Integration Platform: http://www.axeda.com/. Accessed: 2012-03-13.

[3] Blackstock, M. et al. 2008. Evaluation and Analysis of a Common Model for Ubiquitous Systems Interoperability. Pervasive 2008 (Sidney, Australia, May 2008), 180–196.

[4] Blackstock, M. et al. 2010. MAGIC Broker 2: An open and extensible platform for the Internet of Things. Internet of Things (IoT) 2010 (Tokyo, Japan, Dec. 2010)

[5] Blackstock, M. et al. 2011. Uniting online social networks with places and things. Web of Things Workshop at Pervasive 2011 (San Francisco, USA, June 2011), 5:1–5:6.

[6] Fielding, Roy T. and Taylor, Richard N. 2002. Principled Design of the Modern Web Architecture. ACM Transactions on Internet Technology (TOIT). 2, 2 (May. 2002), 115–150.

[7] Guinard, D. et al. 2010. A resource oriented architecture for the Web of Things. Internet of Things (IoT) 2010 (Tokyo, Japan, Dec. 2010)

[8] Guinard, D. 2010. Towards opportunistic applications in a Web of Things. WoT Workshop at PerCom, 2010 (Mar. 2010), 863–864.

[9] Guinard, D. et al. 2009. Towards physical mashups in the Web of Things. INSS '09, (Jun. 2009), Pittsburgh, USA

[10] Karlen, W. et al. 2011. Location independence in patient monitoring. Anesthesia & Analgesia (Las Vegas, USA, Aug. 2011), 37.

[11] Open Sen.se Feel, Act, Make sense: http://open.sen.se/. Accessed: 2012-03-12.

[12] Pachube - The Internet of Things: http://www.pachube.com/. Accessed: 2012-03-12.

[13] Paraimpu - The Web of Things is more than Things in the Web: http://paraimpu.crs4.it/. Accessed: 2012-03-12.

[14] Pintus, A. et al. 2011. The anatomy of a large scale social web for internet enabled objects. WoT Workshop at Pervasive 2011 (June, 2011). San Francisco, USA

[15] Stirbu, V. 2008. Towards a RESTful Plug and Play Experience in the Web of Things. 2008 IEEE International Conference on Semantic Computing (Aug. 2008), 512–517.

[16] The Internet of Things - ThingSpeak: https://thingspeak.com/. Accessed: 2012-03-12.

[17] ThingWorx – The 1st Application Platform for the Connected World: http://www.thingworx.com/. Accessed: 2012-03-13.