Interoperability through Semantic Metastandards

David Janes (davidjanes@davidjanes.com)
February 2016

Abstract

This document proposes a system for enabling IoT Interoperability by semantically describing IoT-related actions and sensing. It also proposes that almost all IoT actions can be describing in terms of manipulating JSON-like data structures, mirroring the RESTful concepts of PUT and GET.

1. Terminology

- Band: a JSON Object of related data
- Composition: individual "attributes" of a Thing are described, and Things are described in terms of those attributes. See also[1].
- IOTDB: a project implementing what is described in this document
- JSON Object / Dictionary: an object that fits into the "JSON" model, that is, being comprised of dictionaries and arrays, a simpler atomic types. The serialization does not necessarily have to be JSON.
- JSON-LD[2]: a way of encoding semantic triples using JSON.
- Linked Data[3]: IDs are expressed using dereferencable URIs rather terms or numbers, the cornerstone concept of the Semantic Web.
- QNames[4]: a way of shorting URIs to make them less unwieldy.

2. Approach

Broadly speaking, our approach is to take what already exists today in the world of Things, find commonalities and patterns, regularize them, and then codify them.

[1] https://en.wikipedia.org/wiki/Composition_over_inheritance
[2] http://json-ld.org/
[3] http://linkeddata.org/
[4] https://en.wikipedia.org/wiki/QName

Prescriptive vs. Descriptive Standards

This proposal takes a radically different approach for how we should create interoperability amongst Things. The traditional idea is that we should have *prescriptive standards*, e.g. “here’s how you do X”. Instead, the “Semantic Metastandard” approach says we should create *descriptive standards*, that is, let’s create terminology and processes that describes how Things are manipulated, and then leave the mapping of that to “real Things” as an implementation detail.

Classes vs. Composition

This article also proposes that we do away the concept of “classes” in terms of defining the IoT, e.g. “here’s the definition of a light”, “here’s the definition of a washing machine”, and so on. Instead, we propose two simple principles:

- Things *are* what they say they are
- Things *do* what they say the do

A Thing is a Light because it says it is a light. There is no common definition of what a Light must do, but by nature this just falls out of how lights actually work:

- it may be turned on and off (`iot-purpose:on`[5])
- it may be dimmed (`iot-purpose:brightness`[6])
- it may have its color changed (`iot-purpose:color`[7])

When a client wants to manipulate the Lights, it finds all the Things that say they are lights, then introspects the *attributes* it exposes do decide how to manipulate them.

Actions, Events, Properties vs. Simple REST

We will argue that almost everything that needs to be done in the IoT is, can, and should be modeled by reading and writing to JSON dictionaries; or in REST terms, GET and PUT. Complicated prescriptive models, such as the W3C’s Actions, Events, Properties[8] are unnecessary, unparsimonious and don’t reflect how most Things today actually work.

[5] https://iotdb.org/pub/iot-purpose.html#on
[6] https://iotdb.org/pub/iot-purpose.html#brightness
[7] https://iotdb.org/pub/iot-purpose.html#color
[8] https://www.w3.org/2015/05/wot-framework.pdf

3. Semantics

The core of the our approach is we *semantically* describe all the terms in our state. Consider the brightness of a light:

- what's the minimum brightness?
- what's the maximum brightness?
- what's the granularity - how many steps are there?
- do we express brightness as an integer, a number or even a string - e.g. "low", "medium" and "high"?

Or even just in general, if we have the term "b", or "br" or "brightness", how do we know that this means "adjust the brightness"?

We believe that it's not practical (or wise) to come up with a "universal" concept of brightness, as for example the Open Connectivity Foundation does[9]. Instead, for all terms associated with a Thing we let the Thing itself describe what it does, defining for example:

- its purpose (in IOTDB: iot-purpose:brightness)
- its minimum value
- its maximum value
- the units it is measured in
- the type of value it takes
- its precision
- and so forth.

Our observation is that in fact there isn't that many things you need to define to create a rich and powerful language that can describe almost all things. We will describe this further below, but for example here's the "purposes" we've defined at the footnote[10].

4. Bands

Our key insight is that it's better to view a Thing as a cloud of related but independent JSON objects, rather than say, as a single object that one manipulates in various ways.

The number of bands is potentially unlimited, but the core ones identified are:

- **istate** - the Input State: the actual state of thing

[9] http://www.oneiota.org/revisions/324
[10] https://iotdb.org/pub/iot-purpose/

- **ostate** - the Output State: the state we want the thing to become
- **meta** - the Thing's metadata
- **model** - the Thing's JSON-LD Model

Furthermore - and this is very important - is that almost every manipulation one needs to do involves simply writing (PUT) to these objects or reading (GET) from them. The exception is “actions” where long running state may be need to be tracked: we deal with this in the footnoted post[11].

Most Things today don’t cleanly separate these concepts, but do have many elements that correspond to them. Besides the obvious conceptual advantages of breaking up a Thing’s data this way, it has massive implementation advantages, as (say) the same code the moves **istate** data around can also be reused for **ostate**, **meta** and **model**.

Interstitial State

The interstitial state is the period of time between when a command is issued to a Thing to when the Thing actually completes the commands.
Consider the WeMo Switch, a popular "starter" IoT device. Turning a light from **off** to **on** looks like this.

1. the switch is off
2. the user sends “turn on” command
3. the WeMo receives the command
4. about 300ms passes
5. the switch actually becomes on

The interstitial is at step 4, where there `on=true`[12] in terms of what the user is asking but the Thing is actually `on=false`. This is noticeable to the user and has to be modelled by our Things in order to create effective interfaces.

The long and short is that we propose that the state be formally by broken into two parallel states: the **ostate** and the **istate**, that share a common set of semantic definitions. Things are manipulated by writing to the **ostate**, and the actual state of a Thing is discovered by reading the **istate**.

---

[11] https://iotdb.org/social/imadeit/post/139237158604/5-actions-should-be-used-lightly-and-should

[12] Note how we’re manipulating the state of the Thing by writing to a JSON object, as explained in the previous section.

Note that this is not a theoretical concept: many devices have something like this today, such as the Philips Hue[13], though in a much more ad-hoc fashion. By explicitly spelling this out, we greatly simplify specifying how Things work, as the **ostate** and the **istate** get to share a common set of semantic definitions (called the **model**, explained below).

Coordinating Updates and Errors

In our reference implementation we use timestamps to coordinate updates. “Deep errors” are detected the same way users do on the web today: if something didn’t work (which can be seen by the timestamps), the command can be resent.

However, we are not making this part of this proposal as the semantics and band concepts are more important.

Band Definitions

istate

The “input state”, the actual state of a Thing. For example, consider a LIFX “White” Light, which has an on-off state and a brightness state.

The **istate** for this looks like

```
{
 "on": true,
 "brightness": 100
}
```

Note that the terms **on** and **brightness** are *not* standardized: they are defined in the **model**. This gives us a massive amount of flexibility of what can be defined in the **ostate** and **istate**, as they can be customized on a model-by-model basis.

ostate

The “output state”, the state we want the Thing to transition to. By convention[14], the values are always null unless it’s actually transitioning, i.e. in the interstitial state.

The ostate looks like this when nothing is going on

[13] http://www.developers.meethue.com/documentation/lights-api
[14] yes, this is defining a standard to some degree but in practice it is quite reasonable

```
{
 "on": null,
 "brightness": null
}
```

and e.g. like this when we are turning it off

```
{
 "on": false,
 "brightness": null
}
```

meta

The **meta** band stores data about the Thing note directly related to the state: e.g. its name, its description, its manufacturer, its reachability, its facets (what it claims it can do), or anything else you may want.

Additional reading about the **meta** band can be found at the footnote[15].

model

The **model** band stores the semantic information associated with the Thing and especially the Thing's **istate** and **ostate**.

In IOTDB we use a domain language called IoTQL[16] for defining the models. We are including this here because it is understandable and terse, but do not consider this part of our proposal. IoTQL compiles into JSON-LD.

Here's the IoTQL for the LIFX White Light. The JSON-LD corresponding to this can be seen at the footnote[17].

```
CREATE MODEL LifxWhite WITH
    schema:name = "LIFX White",
    iot:facet = iot-facet:lighting.light,
ATTRIBUTE on WITH
    schema:name = "on",
    iot:purpose = iot-purpose:on,
    iot:type = iot:type.boolean
ATTRIBUTE brightness WITH
```

[15] https://iotdb.org/social/imadeit/post/138157191389/3-wot-thing-needs-to-have-meta-band
[16] https://github.com/dpjanes/iotdb-iotql
[17] https://github.com/dpjanes/homestar-lifx/blob/master/models/lifx-white.json

```
        schema:name = "brightness",
        iot:purpose = iot-purpose:brightness,
        iot:type = iot:type.integer,
        iot:unit = iot-unit:math.fraction.percent
    ;
```

The important parts of this Model are highlighted, namely how we define the terms **on** and brightness **seen** in the **istate** and **ostate**.

So here's how we use Semantic Metastandards to turn **off** a LIFX Light:

1. let's assume that **istate** (actual state) of the thing is **on=true**, the light is on
2. the client looks for an **attribute** with the purpose **iot-purpose:on**
3. the client see that this is the term **on**, note however that this name could have been anything, e.g. "on-off", "powered" or whatever
4. the client sees that it is a boolean and so writes **on=false** to the **ostate** (the output state, the state we want to go to) of the light
5. the LIFX light turns off
6. the ostate reverts to **on=null** and the istate becomes **on=false**.

5. Implied Interface

Our proposal implies a very natural hierarchical interface for Things:

- things/
    - thing1/
        - istate
        - ostate
        - model
        - meta
    - thing2/
        - istate
        - ostate
        - model
        - meta
    - …

This can be seen below in the live demo.

6. Implementation

This proposal is backed by a well-tested implementation in Node.JS that has integration with over 50 different “real” Things and Platforms, such as WeMos, Hue Lights, LIFX, KNX, SmartThings, and so forth.

Semantic Definitions

Our Semantic Definitions are built as an extension to Schema.org’s system. They can browsed here: https://iotdb.org/pub/ and the GitHub source can be found here https://github.com/dpjanes/iotdb-vocabulary.

The vocabulary is divided into four different sections, the first two being most interesting in terms of this paper: iot, iot-purpose, iot-facet, and iot-unit.

iot

These are the core definitions, including our major types.

https://iotdb.org/pub/iot.html

iot-purpose

This is how Things declare “what they say they do”. Semantic definitions of how things are manipulated / how sensors report data. These are the most important definitions in terms of this paper.

https://iotdb.org/pub/iot-purpose.html

iot-facet

This is how Things can declare “what they say they are”. In terms of this paper, consider this informative, it is not a core part of our proposal.

Consider a WeMo Switch. Independent of its Model, what it is depends on what it's connected to. If it's connected to a light bulb, we can attach **iot-facet:lighting.light** to the metadata. If it's a space heater, perhaps **iot-facet:climate.heating**.

https://iotdb.org/pub/iot-facet.html

iot-unit

This is our "units / weights / measure" definitions. In terms of this paper, consider this informative, it is not a core part of our proposal. Likely a formal spec would prefer something like QUDT[18].

https://iotdb.org/pub/iot-unit.html

Code Base

For installation instructions, see https://homestar.io/about.

The source code is all Apache 2.0 and available at https://github.com/dpjanes in repositories named with “iotdb” or “homestar”.

Live Demo

Note that there is no security turned on in this demo, but it is possible. This paper does not suggest a security model, except that as with the web itself, it has the flexibility to layer strong versions on top.
You can more info about the demo here:
https://github.com/dpjanes/homestar-coap/blob/master/docs/plugfest/entry-points.md

HTML

The user interface presented here is built entirely through introspection of the Things. Note that the Things are all simulators for practical reasons.

http://homestar.io:20000

HTTP

HTTP API presented for Things. Note RESTful and HATEOAS. State can be read via **GET**, and the **meta** and **ostate** can be also manipulated via **PUT**. Note the simplicity and naturalness of the API.

http://homestar.io:20000/api/things

[18] http://qudt.org/

CoAP

Similar to the HTTP API, but via CoAP. It would probably be better if we exposed all the Things via `/.well-known/core`.

coap://184.107.137.234:22001/ts

MQTT

This broadcasts all manipulations of Things. Manipulating Things via MQTT is turned off in this demo (though it is possible).

mqtt://homestar.io:20001/runners/0a6ad141-e2f2-407e-a4a3-e0403821b6e9/api/things

If you have an MQTT client installed, you can follow along with

```
mqtt subscribe --host homestar.io --port 20001 --topic '#' --verbose
```

7. Useful Links

Semantic Definitions

- https://iotdb.org/pub
- https://github.com/dpjanes/iotdb-vocabulary

Source Code

- https://github.com/dpjanes

Presentations / Slideshows

- Semantic Metastandards
  http://www.slideshare.net/dpjanes/semantic-metastandards-will-unlock-iot-interoperability
- Semantics and the Iot
  http://www.slideshare.net/dpjanes/semantic-and-the-internet-of-things

- Standardless IoT / Interoperability
  http://www.slideshare.net/dpjanes/2015-04-global-io-t-day-wien-interoperability-with-standardless-iot
- What a Thing API should look like
  http://www.slideshare.net/dpjanes/what-a-thing-api-should-look-like-global-iot-day