

Semantic Metastandards Will Unlock IoT Interoperability

David P. Janes^(✉)

Founder, IOTDB.Org, Toronto, ON, Canada
davidjanes@davidjanes.com

Abstract. Interoperability within the Internet of Things is mired in entrenched incompatible specifications, backed by industry giants and consortiums seemingly hoping to “win the stack”. There is little hope that a common set of standards will be universally adopted that will allow all our Things to work together. However, by applying some of the core lessons of the Internet - use URIs for IDs, manipulate documents using the REST Model, and that documents should be “JSON-like” dictionary - we can create near seamless interoperability between Things independent of their standards stack.

Keywords: REST · Internet of Things · Interoperability · Semantics · Semantic web · Linked data · API · Semantic Metastandards · Standards

1 Introduction

1.1 The State of Play

The window of opportunity for a common, widely adopted TCP/HTTP/HTML-type stack of Internet of Things standards has passed. We now live in a world with a plethora of manufacturer and consortium based specifications, full of magic numbers, ID strings, and ad-hoc architectures. Like with Internet video in the 1990s, deliberate incompatibility is “baked in”. Common amongst these specifications show little evidence of learning anything from the phenomenally successful Internet architectures of the last 25 years.

1.2 IOTDB/HomeStar

We have an Open Source reference implementation¹ which can be freely used to experiment with the ideas expressed in this paper.

¹ <https://github.com/dpjanes/node-iotdb>.

2 Semantic Metastandards

2.1 Our Proposal

We propose a new approach for creating interoperability amongst Things, by describing what they do (or report) and providing a simple, well-understood method for relaying (or retrieving) these instructions to/from the Things. We call this approach “Semantic Metastandards”.

The beauty of this approach is it’s inherently independent of vendors and consortiums. A TV can be on or off, a lock can be open or closed, a stove burner can be set to a particular temperature: how this is done is out of our control, but the fact that they do it is not.

The trick is to discover this vocabulary and provide a programming model that developers can use to apply it to Things. Of course, something actually has to do the work, but this will be all invisible to end users.

2.2 Not in Our Proposal

Discovery, Security, Authentication, Authorization, Interoperability for the IoT is still a huge space with a lot of work to be done. This paper covers “once you connect, how do you talk?”.

2.3 Architecture

Our Semantic Metastandard is built on top of the architecture that should be familiar to every Internet developer today:

- use dereferenceable URIs² as IDs;
- use the REST Model, that is, we manipulate and understand the state of Things by doing simple Atomic operations against documents found at URIs; and
- use JSON-like dictionary, that is, data hierarchically composed of a few simple types, arrays and dictionaries with strings for keys.

The first point is simply Linked Data³, as outlined by Tim Berners-Lee. The second and third points are the architectural basis of most web APIs today.

Components. The core components of this system are as follows:

- ID: a unique identifier for a Thing
- Bands: JSON-like dictionaries of data associated with a Thing
- Vocabulary: URIs describing what are in bands.

² https://en.wikipedia.org/wiki/Dereferenceable_Uniform_Resource_Identifier.

³ https://en.wikipedia.org/wiki/Linked_data.

The relationship between these points are simple:

- A Thing’s ID will let you access it’s Bands
- We change a Thing’s state by writing to Bands
- We get a Thing’s state by read and observing changes to its Bands
- We understand what the state means via its semantic description, also stored in a Band.

ID. An ID is a persistent unique identifier for a Thing. IDs are not usually assigned so much as systematically created. Ideally, this would be universally unique but unique within a given environment seems to work “good enough”.

Our reference implementation creates IDs using a variety of methods, most notably by using unique identifiers expressed by the Things themselves.

Bands. The core insight here is that a “Thing” is not a single object that we manipulate. Instead we should view a Thing (as identified by some Thing ID) as a jumping off point to a cloud of related but independent bands of data.

These are, at a minimum:

- *istate* - the Input State: the actual state of thing
- *ostate* - the Output State: the state we want the thing to become. When this state is achieved, the corresponding values within the *ostate* are returned to null
- *meta* - the Thing’s metadata (for example, it’s name)
- *model* - the Thing’s Model, which describes what it does

What we’re proposing is that Interoperability can be achieved in the IoT by reading and manipulating the data in these bands. This will be demonstrated further in this paper.

Vocabulary. Note that this entire vocabulary could potentially be swapped out and replaced with another vocabulary though more realistically, the vocabulary can be augmented as needed for certain specific applications. However, we think the current version of this covers certainly most home and small-business related IoT applications.

- *schema*: - the schema.org vocabulary, especially for things like name and description⁴
- *iot*: - core definitions, such as basic types, relationships, IDs etc.
- *iot-purpose*: - descriptions of manipulations, such as “turn this on”, “set the color”, “this is the sensor temperature”, etc.

For completeness, we will mention two other sets of definitions:

- *iot-unit*: - weights and measures
- *iot-facet*: - the “purpose” of Things themselves, for example “Climate Control”, “Lighting”, “Security”, etc.

⁴ <http://schema.org/Thing> - not to be confused with a “Thing” in the IoT sense.

All the `iot-*` types can be browsed at iotdb.org/pub. Forkable versions are also available on [GitHub](https://github.com).

We use JSON-LD⁵ to store the semantic descriptions in the model band.

3 Examples

3.1 Arduino Light

We will work through one of the simplest examples possible to demonstrate this idea.

Consider an Arduino with an LED light connected to Digital Pin 13. Turning the light on looks something like this in code:

```
digitalWrite(13, TRUE)
```

This clearly lends itself to a neater description as the manipulation of a JSON document.

```
{
  "D13": true
}
```

Because we're "reading the docs" we clearly know what this means, but it is hardly a scalable model for the IoT.

The "Model". What we need to do is describe what that JSON dictionary means. Here's one possibility (simplified for readability).

```
{
  "iot:model-id": "arduino-light",
  "iot:attribute:" : {
    "@id": "#D13",
    "iot:purpose": "iot-purpose:on",
    "iot:write": true
  }
}
```

There's a lot going on there, but most importantly, note that there's an attribute that:

- refers to "D13" in the dictionary
- has the purpose of `iot-purpose:on`, which is defined to mean "this turns something on and off"⁶

⁵ <http://json-ld.org/>.

⁶ <https://iotdb.org/pub/iot-purpose.html#on>.

Usage. How do we use this Model to turn on the light?

- we look for an attribute that has `iot:purpose` of `iot-purpose:on`
- when found, we look up the `@id`. This tells us what to manipulate in the JSON dictionary - D13
- we create the appropriate update dictionary `ostate: {D13: true }` and send it to thing's Bridge
- the Bridge does the actual work of turning the light on

Or to summarize, we say `state:on = true` but behind the scenes on the Arduino the code does `digitalWrite(13, TRUE)`. On a (e.g.) Philips Hue, an entirely different code path is executed, but both the initial command and the final result (the light turning on) are the same.

Programming. In our JavaScript reference implementation, this looks like:

```
thing.set(":on", true)
```

In IoTQL, this is done by

```
SET state:on = true;
```

Note. We will briefly note here that in a “real” Model, the pin number would be parameterized - otherwise we'd have to make a new Model for each different pin!

3.2 Interstitial State

The “interstitial state” of a Thing is the time period between when we ask a Thing to do a task to when it actually accomplishes that state. With traditional non-connected Things, when tend not to notice this: we flick a light switch and the light comes on. In a similar fashion, we do not see the concept with traditional APIs because we expect things to work or fail: we write to a database and it updates (or not). In the IoT, this is not tenable: we (e.g.) send a request to a WeMo Socket, and then 400 ms (or so) it changes state. This amount of lag is clearly noticeable to users and must be modeled correctly.

The solution we have come up with is to split the state into two parallel bands: an “istate” (input) for modeling the actual state of a Thing, and “ostate” (output) for modeling requests to change the Thing's state. The ostate transitions always its value to null once a command is executed.

Why do we not have one state, with different terms for input and output? As you start modeling ‘real world’ things is there's a lot of ‘near duplication’ of semantic terms.

For example, modeling Lights you might end up with something like this:

- `on`: turn the light on/off - `{write: true }`
- `is_on`: is this on or off - `{read: true }`

These have identically semantic definitions excepting that one is a command and one is reading. With more complicated Things you end up with large sets of near duplicate terms, which furthermore have to be correlate with each other to show

interstitial state. Splitting the state into `istate` band and `ostate` band provides an elegant simplification.

Bands. Note in this case, the model indicates that we can both read and write to the “on” attribute.

```
model:
{
  "iot:model-id": "we-mo-socket",
  "iot:attribute": {
    "@id": "#powered",
    "iot:purpose": "iot-purpose:on",
    "iot:read": true,
    "iot:write": true
  }
}
```

The `istate` band is the current state of the Thing. In this case, the WeMo Socket is “off”:

```
istate: {"powered": false }
```

(Note that we understand what `powered` means by looking at the model).

The `ostate` band only has data when it is actively doing an action. Thus initially, it has only null data.

```
ostate: {"powered": null }
```

We then send a request to turn the WeMo on by changing the `ostate` band

```
ostate: {"powered": true }
istate: {"powered": false }
```

Notice how we are now in the interstitial state - the `ostate` band has a value, and it differs from the `istate`. Finally, the command is executed, the WeMo turns on.

```
ostate: {"powered": null }
istate: {"powered": true }
```

One alternative is to keep the command value in the `ostate` band rather than transitioning to null; however we believe this makes what is happening more clear.

3.3 Type Units, Ranges, and More

This section is to illustrate that attributes can express more information than the “purpose”; they can also store the type of data expected (e.g. an integer, a Boolean), the units that the value is measured, ranges, enumerations, and so forth. Again, because we are using Semantic Web and Linked Data standards, what can be achieved is open-ended.

3.4 Type and Units Example

A WeMo Insight provides (amongst other things) a measurement of how much power is consumed in a day. This can be modeled as follows:

```
“iot:type”: “iot:type.integer”,
“iot:unit”: “iot-unit:energy.si.joule”
```

3.5 Ranges

Here’s how we would model a hypothetical light that has only ten levels of brightness (and off):

```
“iot:type”: “iot:type.integer”,
“iot:minimum”: 0,
“iot:maximum”: 10,
```

3.6 Enumerations

Here’s an example of a TV band selector

```
“iot:purpose”: “iot-purpose:band”,
“iot:type”: “iot:type.string”,
“iot:enumeration”: [
  “iot-purpose:band.tv”,
  “iot-purpose:band.hdmi#1”,
  “iot-purpose:band.hdmi#2”,
  “iot-purpose:band.component”,
]
```

4 Conclusions

This paper outlines a powerful, easily implemented method for creating interoperability amongst Things. The method is based on well-understood Internet technologies in use today. It has no dependency on the cooperation of Thing vendors and consortiums.

The method uses Semantic Web and Linked Data to describe how Things can be manipulated and what their state is. All data related to Things are stored in “bands”, which are JSON-like dictionaries of data. The “model” is the Semantic Model of a Thing, the “meta” is Metadata associated with a Thing, this “istate” is the current state of a Thing, and the “ostate” describes changes we would like to make to its state. The “model” describes the data in the “ostate” and “istate”, allowing users to understand what the data *means*. The “ostate” and “istate” are *bridged* to the actual Things to transfer actions and data; this bridging is essentially invisible to the end user, hence achieving Interoperability using Semantic Metastandards.