EE Journal

Interop Gets Complicated

Just When You Thought You Understood – and Needed – It

by Bryon Moyer
July 13, 2015

I was wrong. There: I said it. I can sleep unburdened tonight.

And what earth-threatening mistake had I made? I mistook syntax for semantics. But I get ahead of myself. Let's dig in to see, first, why anyone would even care about this.

It has to do with – surprise! – the Internet of Things (IoT). Of course. And it has to do with the desire of some to leave behind, or avoid altogether, walled-garden models of interconnected devices that lock users into a particular vendor or technology or… something, in favor of interoperability.

The bulk of what follows assumes that the ability for devices to interoperate on a broader level is a good thing. There are actually contrary voices, but we'll defer that for a few minutes. The big challenge for IoT developers is to implement an environment where any device can talk to any other device. Which is a woefully inadequate way of simplifying the notion of interop.

I was dragged into this rabbit hole by a useful discussion with a gentleman from the UK, one Geoff Revill, who spun a private conversation off of a rather lengthy **LinkedIn discussion** on data-centricity. And it forced me to rethink the "layers" that I had been **positing**, with basic communication overlaid by "semantics" or "business objects."

Geoff articulated a definition of interoperability that goes something like this: interoperability means the successful interaction between:

- different systems
- built at different times
- with different hardware
- using different software
- doing different things.

Sounds pretty straightforward, at least in concept. What that discussion immediately did was to make me realize that what I had been calling "semantics" wasn't. Let me explain.

Just the Messenger

Let's take a messaging protocol like **MQTT**. It has a basic format that includes headery sorts of things and a place for payload data. That payload can be any data in any format; MQTT doesn't care. In order for two ends of the communication to do anything meaningful with that data, they

have to know what it means. Which means you can't just plop down some super-serialized micro-marshalled chunk of bittage and hope the other guy will figure out what it means.

In order for communication to happen, both sides need to understand the structure of that payload. That's above and beyond the structure that MQTT (or whatever protocol you're using) mandates. I had been referring to this next-level structure as "semantics" – when, in fact, it's more like the syntax of the communication between the two ends.

The actual semantics is what is attempted by things like the Zigbee profiles. It goes so far as to say what, for example, an IoT-connected ball-point pen is capable of doing on command. What, you've never heard of a connected ball-point pen? Come on, how many times have you been in a taxi on the way to the airport, and you wondered, "Did I close the pen before leaving?" Or, on your way back home, ready to sign checks, you've never wanted to open the pen remotely before you got there? Or maybe it's the "pen-as-a-service" thing you've been wishing for, where you don't pay for the pen; you pay by the word written.

But I digress.

A profile would suggest which commands are available to a pen. Open; close; explode; show ink level (pre-explosion); etc. The syntax would specify how the commands are structured. MQTT, as our example, specifies neither such syntax nor semantics; it's left to other standardization bodies to do that. Other protocols try to address these higher layers.

You could argue that the syntax and semantics should be independent of the messaging protocol. Just like a given message protocol may be adapted to run over WiFi or Ethernet or Bluetooth or cellular, so Thing profiles could be abstracted to run over Zigbee or Thread – oh wait, that already **happened** – or MQTT or AMQP or Weave or whatever.

You could likewise argue that syntax and semantics themselves should be separated into different abstraction layers. That would allow the semantics for a type of device to be implemented over differing syntax rules. Kind of the way human languages work (except for those all-too-human semantic concepts that defy translation).

From Structure to Discovery

This link to human languages isn't necessarily frivolous. Any of you who have dabbled in linguistics probably know that there's yet another layer above semantics: it's called "pragmatics." It deals with fuzzier notions like context and level and such. I wondered idly whether such notions might apply to the IoT as well.

Some of my discussion with Mr. Revill touched on what could realistically be considered pragmatics instead of semantics. Discussions of syntax and semantics tend to involve specific rules and structure. And yet, increasingly, we're living in a world where the structured, the pure crystalline, is giving way to the amorphous – with lots of junk inclusions.

Let's momentarily leave the IoT for another example. In the past, if you needed information, you had structured ways of finding it. Perhaps the encyclopedia; perhaps the library, with its Dewey Decimal precision.

Today? The information is simply lumped out there somewhere unspecified, unstructured, along with a bunch of false or irrelevant information. We do a Google search and find a list of things that look promising and then decide which to click on.

So, yesterday, if I'd wanted information on the element cobalt, I might have consulted my latest CRC manual. Today? I Google, and I might get that same information from some odd PowerPoint presentation posted on some obscure site by some assistant professor somewhere. As long as I trust the source, then clicking the presentation is just as useful – perhaps even more so if well presented – than trundling about with the massive CRC.

Let's bring that back to the IoT. And let's assume that I'm a node on some network, and I'm looking for temperature information from some part of the world. Perhaps my sensor-soulmate. As defined right now, I can talk only to temp sensor nodes that speak my syntax and understand my semantics. But what if there's a higher layer? One that admits discovery: "I'm looking for a connected thermometer that has the following characteristics:

- Operates in a specified part of the world;
- Has a specified level of accuracy;
- Has a specified history and guarantee of calibration and maintenance;
- Can issue updates at a specified interval (or faster);
- Has a history of providing measurements for a specified period of time;
- Has no history of problems or reports of faulty measurements;
- Has rock-hard abs." (OK, this one is negotiable. And you can blame me, not Geoff.)

Whether or not you call this "pragmatics" isn't really of concern here; the point is that there can be much more to interoperation than simply an API for reading temperatures. There's a larger context associated with the problem that furthers interoperability.

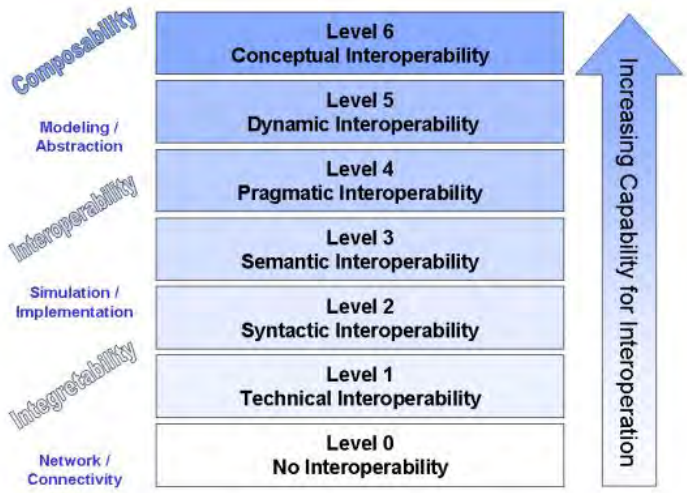
If I find my desired node but can't speak with it due to incompatible syntax and semantics, I've still got a problem. But it doesn't require that the entire world settle on one "language" at every layer as long as the number is manageable and brokers can act as simultaneous translators. There are plenty of systems today, and have been for years, that negotiate communication details as part of the building of a connection.

Of course, there are protocols that permit discovery already, so it's not like this is a wild new idea. But for those striving for global interop – and there have been efforts underway for some time, like OASIS's XDI project – the realization came eventually that it wasn't about protocols and APIs and such: it was about data. And we're back to that whole **data-centric** thing. You're not discovering nodes via a protocol; you're discovering data.

Above Pragmatics

While noodling about on this concept of pragmatics, I did some bouncing about on the Internet. And a bit of Googling made my jaw drop.

There's an interop model that includes not only syntax and semantics, but pragmatics above that – and then two further layers of abstraction. This is a highly idealized model with abstracted ideas about what's required for complete interoperability. At least, according to a couple of guys named Petty and Weisel and further refined by the Virginia Modeling, Analysis and Simulation Center (VMASC).



(Image credit: "LCIM". Licensed under CC BY 2.5 via Wikipedia - https://en.wikipedia.org/wiki/File:LCIM.png#/media/File:LCIM.png)

I won't go over **all the layers**, but here's a summary of those top two:

- Level 5: This level deals with the fact that data models and their underlying assumptions are not static; they may change over time. If nodes can comprehend and deal with that long-term changeability, then they've achieved Dynamic Interoperability.

- Level 6: Quoting from Wikipedia, "Finally, if the conceptual model – i.e. the assumptions and constraints of the meaningful abstraction of reality – are aligned, the highest level of interoperability is reached: Conceptual Interoperability. This requires that conceptual models are documented based on engineering methods enabling their interpretation and evaluation by other engineers. In essence, this requires a "fully specified, but implementation independent model" …; this is not simply text describing the conceptual idea."

Granted, per the diagram, those upper layers transcend interop into composability, an area that I won't venture into because you'll immediately discover me to be a fraud. But conceptually, they add richness to what it means for machines to "understand reality."

But Do We Need This?

So that's all nice in that it sets this impossibly high bar for interop that probably has half of you polishing up your resumes to work in the general store of some remote town where "interop" means speaking clearly enough so that the guy on the stoop understands you when you say, "Howdy."

And all those big, mean, grasping companies that really want to lock everyone into Their Walled ~~Prison~~Garden can lick their chops and weep crocodile tears that interop is so hard that it's really senseless to try so why bother. $20 bucks please for this month – oh, wait, sorry, no… seems it's gone up to $30 this month. Bummer. No, you can't use someone else's device.

That's the picture we tend to paint of companies with too much control and no ability for customers to find alternatives. It might not be fair; after all, not every company is the cable company.

In recent discussions about the Industrial IoT, it's been noted that, while this M2M space has made much more progress than the consumer space, it's largely of the walled-garden variety. Companies like Honeywell and Rockwell are named, and your entire factory works under one such system. No running about swapping just any vibration sensor in willy-nilly; it won't work.

And some folks don't see that changing. I suppose it depends on how the dominant companies handle it. If they continue to provide value, then the costs and uncertainties of trying to open things up may not be worth it. If there's a risk that your line has to go down due to some interop problem, you're going to stick with what's tried and true. The walls in the garden provide safety.

If the companies subsequently try to leverage the fact that they now have an entire factory built out with their technology, making the costs of switching prohibitive, then a cry for interop may develop more steam. Not clear we're at that point yet – especially since the build-out is still underway.

I also had a discussion with Olea Sensors – they implement local (i.e., not cloud-based) analytics, and it's very much of a custom business. It's not a plug-and-play thing by any stretch.

For a lot of the companies they work with, full, complete interop is simply not important. Companies are solving vertical problems, and as long as everything works within that vertical domain, it just doesn't matter whether, in theory, the connected farm tractor can talk to a baby monitor or a drilling head miles into the earth.

They don't outright say that interop isn't important or that it might never happen, but they do put an interesting perspective on it. They note that interop isn't needed to get things done: interop

helps to get things done more efficiently. Right now, the focus isn't on efficiency; it's on getting things done.

Yeah, it would be nice if we could all wait at the starting line, engines revving, while the nice folks up in the booth decide all the aspects of interop needed at all levels, but that's going to take a long time and the folks at the starting line have itchy clutches. And a bunch of them have started without waiting, because you can make more money by going earlier.

Which means that imposing interop later on may be a bit messier. But, on the other hand, there's no way the folks in the booth are going to be able to think of everything with just paper in front of them. They're going to need examples and first movers and – let's be clear – people who will make mistakes that we can all learn from in order to sort through what we really want. So you need that over-eager driver to spin out in the oil patch to realize, "Ah, oil patch not good. Didn't see that coming."

So first we'll get a bunch of implementations that don't talk to each other. And we'll be happy we can do new things – until we get annoyed by the fact that these things don't talk outside of their designated cliques. And then we'll start worrying about that whole interop thing.

So what's the upshot (now that you've hopefully moved beyond, or better yet, forgotten my opening comment)? Well, I summarize it as two (plus) things:

- Full interop is harder than we think
- While interop is nice, it's probably not going to happen for a while.
- As a corollary, you'd be happy enough for now to get all the Things on one network to talk to each other reliably at the lowest level. First things first.

But, of course, you may disagree… and are invited to do so below.