Loci of Interoperability for the Internet of Things.

Ted Hardie
Google
Submission to the IAB IoT Semantic Interoperability Workshop

Introduction

Common Internet of Things deployments currently connect a set of nodes to a hub; the hub may then be connected to a cloud-based service. Some of the IoT nodes are sensors; some are more complex edge devices. One class of edge devices has both sensors and manageable state[1], but may be modeled as if the two were independent. A more complex set of edge devices also has the ability to contain and implement rules specific to the device[2]. When a hub is connected to a cloud based service, the service may act as a point of management, as an external audit point, or much more.[3]

As we think about semantic interoperability, we first need to unpack the node-hub-cloud model, as the details affect which devices must be able to interpret specific semantics. Each locus of interpretation presents different issues for the interoperability of data models, and the differences in structure may also imply different information models.

Where is the data model understood?

In the case of nodes which are simple sensors, there is no need for the node itself to understand any semantics; it does not even need to know what it is saying, much less what any other nodes are saying. For IoT actuators, the node must be able to expose the available state transitions but, again, has no need to understand why a specific state transition is occurring.

If, in the node-hub-cloud model, the hub is interpreting the data from the nodes and taking action in the local network, then it clearly must have an understanding of the data models for the information arriving from individual nodes. It must also understand how to activate specific controls in response to a set of rules which are locally available. When a motion sensor sends data indicating that motion has been detected, the hub might activate nearby lights, cameras, or both, depending on the local rules. In this "strong hub" model, the cloud based service may be used to log data or install the set of rules in a hub, but the service may not need to understand the actual sensor schemas or the data models of the IoT actuators.

---

[1] Hereafter the implementations of manageable state are called *IoT actuators*. Note that this usage is broader than the commonly understood mechanical actuators.

[2] The interpretation of these rules is not treated in this paper due to space considerations.

[3] This node-hub-cloud architecture has been called out by Phil Levis and Dan Boneh, among others.

If, in contrast, the response logic is based in the cloud, then the hub can act as a clearinghouse for packets destined for the remote system without needing to know anything about the packets' payloads. It may be re-packaging in order to adapt to link-layer transitions or security contexts, but it fundamentally does not need to understand any of the data models or controls. In this "weak hub" model, the locus of interpretation is solely in the cloud.

It seems unlikely that any system will be purely hub-centric or purely cloud-based. Having lock and alarm logic based remotely has failure modes that would make such a system insecure; having potentially long response times for motion sensing or similar activities would also result in poor user experiences. Having management systems be purely local would have the opposite issue: the ability to change state remotely would be lost or much riskier, as it would require packet traversal into the home network from arbitrary remote destinations.

In a vertically integrated system, such as that by a single vendor, the locus of application for specific rules could be tightly specified. That, in turn, would indicate where each data model would be need to be understood. All irrigation-related rules could be set as cloud-based, for example, so the local hub need do no more than pass through the moisture sensor data and then the commands destined for the IoT actuators at individual sprinklers. That same hub might, in contrast, be the locus of application for all motion-sensing rules, and it would need detailed understanding of both that sensor data and the relevant IoT actuator commands.

Mixing and matching

While creating a common systems engineering model without vendor lock-in is theoretically possible via standardization, systems engineering standards of this type tend to have very slow release cycles. The ability to mix and match sensors, hubs, and clouds without a common systems engineering model would significantly speed the adoption of the Internet of Things. The consequence is that the different components of a mix-and-match system should not be built with a presumption about where specific data will be interpreted and rules applied, nor with the presumption that specific data will be handled at only one other point in the node-hub-cloud system. To put this in the terms of RFC 3444, the builders of a specific node cannot know into what information model its data model will be incorporated, as the information model at each locus of interpretation may be different.

The workshop organizers alluded to one approach which may be used in this deployment mode in their question "What is the role of metadata, which is attached to data to make it self-describing?" Metadata associated with data (or IoT actuator commands) may make it significantly easier for each part of the system to act as a locus of interpretation; it may also make it easier for that locus of interpretation to move within the system as a particular deployment evolves. For example, an initial deployment of managed HVAC at a business might use a local hub and a very simple time-of-day and day-of-week calendar. A later iteration might be cloud-based and integrated with the individual calendars of those who use the office, so that early mornings, late nights, and weekend work are accommodated. Identifying metadata for the

related temperature sensor data and IoT actuator commands would make it significantly easier for the cloud-based service to interact with the deployed system.

Minimal Metadata

While the workshop organizers used the phrase "self-describing", practical deployments will contextualize data using a layer of indirection which references an existing schema or data model. To achieve the best semantic interoperability in a context where the locus of interpretation is unknown, it is critical to reference the most primitive schema and element that yields the right interpretation for the data. This, in turn, implies that it is critical to construct schemas such that this is possible. This focus on the primitive increases the likelihood that a specific state will be understood by allowing at least some states and commands to be accessed when newer devices are added to an existing system. It also reduces the overlap of common features being referenced in multiple ways. Lastly, it decreases the amount of information leakage related to the system, because only messages about advanced features reveal their existence and permit them to be used for fingerprinting.

A useful example here is a lighting model from UPNP. In this model, On and Off for lights are simple states referencing *urn:upnp-org:serviceId:SwitchPower1*. Despite the overall schema for a binary light containing manufacturer, model, and serial number[4], the actual data reference for a status check or actuation does not even reveal that the node is a light, and an application would not need to understand all of *urn:schemas-upnp-org:device:BinaryLight:0.9* to interact with it at this level. Similarly, if a dimming feature is added to the light, this does not change the mechanism by which On and Off are referenced; they remain the original primitives. Instead, a second independent feature, *urn:upnp-org:serviceId:Dimming1,* gives a load level status or target[5]. This approach allows incremental deployment of features and reduces the information leakage of nodes.

One consequence of this approach is that we must modify our statement above that simple sensors need not understand what they are saying. While that remains true, they must be capable of contextualizing what they are saying in line with the approach above. That does not imply that they could take or implement rules related to the data, as those remain external, but does set a minimum bar.

An additional consequence is that since no single piece of information arriving from a node necessarily indicates the full data model for that node, any part of the overall system building an information model must allow it to change as new data appears. This design constraint is not necessarily a large additional burden, though, as it would need to make similar changes were entirely new nodes with those capabilities added to the system.

---

[4] http://upnp.org/specs/ha/UPnP-ha-BinaryLight-v1-Device.pdf
[5] http://upnp.org/specs/ha/UPnP-ha-DimmableLight-v1-Device.pdf

Conclusion

Current IoT deployments use a multi-tiered architecture that combines nodes, hubs, and cloud-based services. In the absence of a common systems engineering approach to specify where different types of rule are applied, nodes cannot know where the data they supply will be interpreted, or even that it will be interpreted only once. Contextualizing the data they send will increase the likelihood that it can be interpreted correctly. That contextualization should reference the most primitive possible schema or data model that results in a correct understanding, in order to increase further the chance of correct interpretation and to avoid leakage of unnecessary data about the system to observers.