The Web of Things and Semantic Interoperability

Michael McCool, Principal Engineer, Intel
IETF100, November 14, 2017
IoT/Fog Evolution Towards an Ambient Ecosystem

Cloud Supported IoT Devices
IoT devices connect to mostly independent cloud services

Locally Networked IoT Devices
IoT devices connect to others in isolated ecosystems

Fog Supported IoT Devices
IoT devices can discover and connect to local “fog services” that facilitate interop.

Ambient Service Ecosystem Computing
Services composed from dynamically discovered devices and services
Standards Stack

Integrate complementary standards that promote interoperability

- Ambient Services Standard: e.g. OpenFog
  - Defines platform for federated edge and cloud services
  - Common services for discovery, security, monetization, updates, etc.

- Metadata Standard: e.g. W3C Web of Things
  - Enables cross-ecosystem service discovery and composition
  - Supports semantic interoperability which enhances usability

- IoT Device Standards: e.g. OCF, MQTT, LwM2M, AWS IoT, etc.
  - Define standard local interfaces for devices and services
The Web of Things and IoT Interoperability

Web of Things supports IoT interoperability
…to enable an open “IoT service ecosystem” to develop
…by making use of well-developed web ecosystem
…without adding to the confusion of “too many standards”

Agenda:
What is interoperability?
Interoperability: What is it?

Two entities can be interfaced “directly”

• Web browser can access a web server and render content from it
  • Despite my never having visited that site before
• I can plug my laptop into the wall to charge
• I can replace lightbulbs in my apartment
• I can buy bolts and nuts and expect them to work together

For the IoT:

• I want to use devices (and data from those devices) from different manufacturers in the same system
Levels of Interoperability

**Semantic:** *Be able to decode the meaning of data.*
- Convert to a representation that has tagged elements with identifiers, properties, and relationships using elements rooted in shared contexts, vocabularies, and ontologies.

**Structural:** *Be able to decode the organization of data.*
- Convert data into a set of common primitive types (numbers, characters, etc) structured with a set of organizational patterns (lists, sets, arrays, links, objects, tables, etc).

**Syntactic:** *Be able to decode the encoding of data.*
- Convert data in a consistent way between a serialized (transmissible, storable) representation (eg a sequence of bytes) and an internal data structure (eg a parse tree).
Other Definitions...

- **Level 6**: Conceptual Interoperability
- **Level 5**: Dynamic Interoperability
- **Level 4**: Pragmatic Interoperability
- **Level 3**: Semantic Interoperability
- **Level 2**: Syntactic Interoperability
- **Level 1**: Technical Interoperability
- **Level 0**: No Interoperability

**Conceptual Interoperability**
IoT Structural and Syntactic Interoperability

For the IoT, “data” can also include actions and events mediated by communications exchanges (eg the “application protocol”).

There is potentially structure between such exchanges as well as within the exchange payloads…

- However, the REST and HATEOAS patterns try to make each exchange meaningful in itself.

- We also need to consider security: onboarding, negotiation of identity, origins, certificates, trust, etc. also constrain interactions

Within the payload of a given communication:

- Syntactic and structural interoperability can be accomplished today by using prescriptive standards designed for this purpose, such as JSON or XML.

- In these technologies, the structure is given explicitly by the syntax, although we can also constrain it with metadata (DTDs, etc).
Why do we want interoperability?
Business Value of Interoperability

“Interoperability between IoT systems is critically important to capturing maximum value; on average, interoperability is required for 40% of potential value across IoT applications and by nearly 60% in some settings.” [+$/4.1B TAM by 2025]

Business Value of Interoperability

“Interoperability between IoT systems is critically important to capturing maximum value; on average, interoperability is required for 40% of potential value across IoT applications and by nearly 60% in some settings.” [+-$4.1B TAM by 2025]

Interoperability Enables IoT Scaling

- *Interoperability* is the key to scaling the IoT ecosystem
  - Maximize number of devices and services that can interoperate
    → Quadratic growth…
  - Provide ecosystem that can compose devices and services flexibly
    → Combinatoric growth!
How?

How to achieve interoperability?
WoT provides a way to build systems that use devices from multiple IoT standards.
In other words, WoT’s goal is to enable interoperability.
W3C Mission for Web of Things

But... how does the W3C WoT differ from other IoT standards?

*Most standards also have the goal of interoperability.*
The difference is in how interoperability is achieved.
Most standards *prescribe* how devices should operate.
W3C Mission for Web of Things

Most standards **prescribe** how devices should operate. The WoT **describes** how devices operate.
Web of Things: Standardised Metadata

**Metadata simplifies application development**
- Decouples underlying protocols
- Enables automated tooling

**Metadata enables interoperability**
- Describe the interfaces exposed to applications
- Describe the communication and security requirements for accessing things
- Describe the data models, semantics, and domain constraints

![Diagram showing interoperability between platforms](image)
Web of Things: Thing Abstraction

Applications act on “Things”
- Things are *software objects*
- Digital avatars representing physical or abstract entities (such as services)
- Have properties, support actions and events
- Can be local or remote

**Metadata descriptions for every “Thing”**
- Each Thing has a URI for its name
- URI provides access to its description
- Ontologies describe Things and their relationships
- Using W3C’s Linked Data semantic framework
1. WoT Thing Description (TD) with simple interaction model

2. WoT Binding Templates to connect to different platforms and ecosystems

3. WoT Scripting API for a browser-like runtime environment

Things can be in client and/or server role: “Servient”
JSON-LD (Linked Data)

```json
{
  "@context": ["http://w3c.github.io/wot/w3c-wot-tdcontext.jsonld",
               "http://w3c.github.io/wot/w3c-wot-common-context.jsonld"],
  "@type": ["Thing", "Light", "iot:LightControl", // Capabilities
             "iot:Actuator", "iot:BinarySwitch"],
  "name": "Intel-OCF-Smart Home LED (2relay)",
  "interaction": [
    "name": "Switch Status",
    "@type": ["Property", "OnOffState", "iot:SwitchStatus"], // Interactions
    "link": [
      {"href": "/a/led2relay?di=79683ab5-8df1-4b7a-b110-c1b8fe251e7d",
       "mediatype": "application/json",
       "http:methodName": "http:post", // Methods (specific to protocol binding)
       "rel": "setProperty"},
      {"href": "/a/led2relay?di=79683ab5-8df1-4b7a-b110-c1b8fe251e7d",
       "mediatype": "application/json",
       "http:methodName": "http:get",
       "rel": "getProperty"}
    ],
    // Payload structure and semantics
    // Other interactions: Properties, Events, and Actions
  ]
}
```
Thing Description Example: A Property

// Payload structure and semantics for a Property interaction

"inputData": {
    "type": "object",
    "fields": [
        {
            "name": "value",
            "value": {
                "@type": "iot:Toggle",
                "type": "boolean"
            }
        }
    ]
},

"outputData": {
    "type": "object",
    "fields": [
        {
            "name": "value",
            "value": {
                "@type": "iot:Toggle",
                "type": "boolean"
            }
        }
    ]
}
WoT Semantic Interoperability

*Meaning is associated with data entities by annotating various metadata entities with vocabulary elements*

Vocabulary elements can also have relationships with each other.

- A reasoner can be used to *infer* additional properties or relationships
- The vocabulary may be explicit or implicit

Semantic Web technologies like RDF and JSON-LD make the context and vocabulary *explicit*

- These technologies can also define relationships between vocabulary elements
- *Linked Data* gives each vocabulary element a unique URI and interprets connections between them as hyperlinks.
Linked Data and Semantic Interoperability

Well-defined semantics ensure that platforms share the same meaning for the data they exchange

- Discovery based upon properties and relationships
  - Search engines that can index the Web of Things
- Verify consistency and interoperability
  - Design service compositions based upon knowledge of which services are compatible
- Reuse existing domain knowledge
  - Using iot.schema.org, SSNO, SAREF, etc.

W3C has a rich suite of related standards
Principals: Michael Koster (Samsung/Smart Things), David Janes (IOTDB), Darko Anicic (Siemens), Dan Brickley (Google)

See: iotschema.org site, discussion doc, and meeting minutes
IoT Service Semantics as Capability Bundles

“Things” considered as

- Bundles of Capabilities
- Capabilities support specific sets of properties, actions, and events
- Services support sets of Capabilities

→ This is a more flexible model than just having definitions of particular Services or Things

→ Ideally we want a way to *model* novel devices, not just *select* them from a list

→ Capabilities allow us to model a novel device by selecting (and perhaps parameterizing) a set of capabilities

**Bundle theory**, originated by the 18th century Scottish philosopher David Hume, is the ontological theory about objecthood in which an object consists only of a collection (bundle) of properties, relations or tropes.

As opposed to **substance theory**…
Now!

Proof of Concept Work…
Proof of Concept Development Goals

- Understand the relationships of different technologies and standards
- Identify technology and standards gaps
- Overcome specific technical obstacles and gaps
- Demonstrate the business value of new technologies or standards
- Test integration patterns for multiple technologies and standards
- Advance engagements with key ecosystem players
Stages

1. **Metadata Bridging** (value: increase adoption and applicability)
   - Support *all* existing IoT devices from *multiple* ecosystems
   - Rather than trying to bridge data peer-to-peer, bridge *metadata*
   - Supports end-to-end security since data translation can be pushed to secure endpoints

2. **Semantic Voice Control** (value: demonstrate utility, engage key ecosystem player)
   - Support *any* IoT device with adaptive semantic voice controls

3. **Fog Integration** (value: develop support for ubiquitous localized services)
   - Deploy using local compute resources for proxies, translators, and acceleration

4. **Service-Oriented Development System** (value: support ecosystem development)
   - Support development and deployment of code for services, not (just) devices
1. Metadata Bridging

Goal: Maximize number of devices accessible

• Translate all metadata into common format (WoT TD)
• Infer and add extra information not available in native metadata
• Make IoT devices available to any system that can process WoT TDs
• Does not need specific changes to target IoT devices.

→ Prototype built using OCF Smart Home Demo as test target.
1. Metadata Bridging

Goal: Translate OCF metadata and make it available to via WoT Thing Description

Client
- Browser
- Postman
- WoT Test Servient

Real Devices - Leaf Processor (Edison)
- OCF proximity
- OCF temp
- OCF light
- OCF button

Physical Hardware

Gateway (UP Squared)

Auxiliary Metadata
- OCF metadata bridge

Simulated Devices - Docker Container
- OCF proximity
- OCF temp
- OCF light
- OCF button

Simulated Hardware

HTTP bridge
1.5 Metadata Bridging

Goal: Improve and expand system architecture

- Support NAT traversal
- Support Thing Directories
- Translate native metadata more completely
- [Support devices from multiple standards]
- Support secure transport and authentication
1.5 Metadata Bridging
Burlingame WoT Plugfest, July 2017

Add: Ingest official OCF data models, traverse NAT, use Thing Directory, …
Metadata Bridge

Metadata generator updated to include:

• Semantic Annotation
  • WoT tdcontext, common context; iotschema.org,
  • HTTP methodName, rel
  • Capabilities (on Things), Interactions, Data
• Protocol Bindings
  • inputData, outputData

```json
{
  "@context": [
    "http://w3c.github.io/wot/w3c-wot-tdcontext.jsonld",
    "http://w3c.github.io/wot/w3c-wot-common-context.jsonld",
    "http://http://iotschema.org/"],
  "@type": [
    "Thing",
    "Light",
    "iot:LightControl",
    "iot:Actuator",
    "iot:BinarySwitch"
  ],
  "name": "Intel-OCF-Smart Home LED (2relay)",
  "interaction": [
    {
      "name": "Switch Status",
      "@type": [
        "Property",
        "OnOffState",
        "iot:SwitchStatus"
      ],
      "link": [
        {
          "href": "/a/led2relay?di=79683ab5-8df1-4b7a-b110-c1b8fe251e7d",
          "mediatype": "application/json",
          "http:methodName": "http:post",
          "rel": "setProperty"
        },
        {
          "href": "/a/led2relay?di=79683ab5-8df1-4b7a-b110-c1b8fe251e7d",
          "mediatype": "application/json",
          "http:methodName": "http:get",
          "rel": "getProperty"
        }
      ],
      // Methods (specific to protocol binding)
      // Payload structure and semantics
      // Other interactions: Properties, Events, and Actions
    }
  ]
}
```
Metadata Bridge

Metadata generator updated to include:

- Semantic Annotation
- WoT tdcontext, common context; iotschema.org,
- HTTP methodName, rel
- Capabilities (on Things), Interactions, Data
- Protocol Bindings
- inputData, outputData
Semantic Annotation: Issues

- Missing Terminology:
  - TemperatureSensor? IlluminanceSensor?

- Inconsistent Terminology:
  - SwitchStatus/Toggle; Light Colour/Current Color; Temperature/TemperatureData

→ Most of these are due to maturity in IoT ontology development

→ Not a blocking issue…
  → Semantic tooling can bridge multiple ontologies
  → New and more precise ontologies can be used as they become available
  → … and as we learn how to write good ones and what the requirements are
Technical Details – System and Network

• SSH Tunnel Reverse Proxy
  • SSH Tunneling is used for NAT traversal, to give local Things a globally accessible access point and URL.
  • Base address in Thing Description needs to be modified depending on whether it is entered into a global or a local Thing Directory

• OCF Metadata Bridge and Thing Directory Cache
  • Global Thing Directory exists in cloud but it is inefficient for local devices to go through cloud to talk to other, local devices on the same subnet
  • In addition to translating metadata and registering TDs, the gateway also runs a local Thing Directory that stores versions of TDs with local links (via a local base address).
To Do: Security Features

• Use Cases from Plugfest
  • Additional lower-level “patterns” or “system configurations”
  • Can also present current set of system configurations in WoT-Sec doc (sec 5)
• Authenticated, Encrypted, Authorized universally available endpoints
  • Authorization: HTTPS + Oauth 2 (global), Web Tokens (local)
  • Payment: Interledger
• Document POCs
  • Intel: HTTPS; SSH tunnel for NAT traversal; OAuth; COAPS locally
  • Issues: local certs for HTTPS? Lets Encrypt/certbot does not work; cert renewal (need certbot…). Look into “HTTPS Local” (W3C CG).
2. Semantic Voice Control

Goal: Enable *automatic* voice control of *any* WoT-enabled device

- Demonstrate use of semantic markup of Thing Description
  - Using iot.schema.org, SSNO ontologies, and semantic inferencing tools
- Generate adaptive AVS Alexa skill, bridging with Alexa Home Skill
- Layer with WoT metadata bridges to control devices from multiple ecosystems (including OCF)
2. Semantic Voice Control

Goal: Provide generic voice interface using WoT Thing Descriptions

- Adapter queries Directory to find Things during Discovery
- AVS Client invokes skill via Server and Adapter
- Intents are mapped to appropriate semantic Actions
Alexa Smart Home Skill API - Summary

• Assumes devices controlled by “device cloud”: a web service with a public URL.
• Supports set of standard, built-in “intents” with predefined semantics
• When understands one of these intents, creates a “directive” which is sent to an “adapter”, which is typically an AWS lambda function. Lambda is called for discovery first, then later with directives.
• Directive contains: customer authentication, device identifier, and an action-specific payload.
• Adapter should check the authentication (OAuth2.0), then send the payload to the specified device via the device cloud. The “device identifier” includes uuid and cookies provided during discovery.
• Respond with an event that indicates success or failure. Can be synchronous (from the lambda) or asynchronous (from the device cloud).
• Devices can also have shadowed status information and this can be proactively updated.

→ Note that all interactions MUST complete within 8 seconds, and ideally 5 seconds.

For details, see https://developer.amazon.com/docs/smarthome/understand-the-smart-home-skill-api.html
Alexa Smart Home Skill API - Summary
Device functionality is modeled by specifying a set of standard capabilities.

- Each capability has a specific interaction model and payload
  …really more like an “interface” than just a semantic category
- Devices can support more than one capability
- “Device clouds” can have more than one device…

There are also a set of standard capabilities to get/set properties

Many capabilities support both absolute and relative mechanisms to adjust properties

Many “actions” also report the current state
## AVS Home Skill Capabilities: Targets

<table>
<thead>
<tr>
<th>Capability</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PowerController</td>
<td>TurnOn or TurnOff a device (has associated powerState)</td>
</tr>
<tr>
<td>PowerLevelController</td>
<td>SetPowerLevel (absolute) or AdjustPowerLevel (relative) on a device. Has an associated powerLevel property that can be set to a value between 0 and 100</td>
</tr>
<tr>
<td>PercentageController</td>
<td>Generic interface similar to PowerLevelController or BrightnessLevelController, but used when “power” or “brightness” is not a valid description of the controlled property. Assumes a value between 0 and 100 in an “percentage” property. Can also be used to read sensors.</td>
</tr>
<tr>
<td>BrightnessController</td>
<td>SetBrightness (absolute) or AdjustBrightness (relative) on a light. Has an associated brightness property that can be set to a value between 0 and 100</td>
</tr>
<tr>
<td>ColorTemperatureController</td>
<td>SetColorTemperature (absolute) or DecreaseColorTemperature, IncreaseColorTemperature (relative, no value) on a device. Has an associated colorTemperature property that can be set to a temperature in Kelvin (voice interface understands color names, “warm”, “daylight”, etc).</td>
</tr>
<tr>
<td>ColorController</td>
<td>SetColor of a light using “color” whose type is a hue, saturation, brightness (HSB) triple. Note that mapping from this to RGB involves a matrix transform and a nonlinearity. Voice interface uses color names.</td>
</tr>
</tbody>
</table>
## AVS Home Skill Capability: Additional Examples

<table>
<thead>
<tr>
<th>Capability</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LockController</td>
<td>Lock or Unlock a lock device</td>
</tr>
<tr>
<td>ThermostatController</td>
<td>Supports thermostats with one, two, or three setpoints. Can also be queried to find the current temperature using a property and ReportState/StateReport. There is also a “mode” property: ECO, AUTO, COOL, HEAT, etc.</td>
</tr>
<tr>
<td>InputController</td>
<td>Select AV input to a TV using set of standard names (HDMI, etc)</td>
</tr>
<tr>
<td>ChannelController</td>
<td>Select Channel on a TV (both absolute and relative)</td>
</tr>
<tr>
<td>PlaybackController</td>
<td>Start and stop audio source</td>
</tr>
<tr>
<td>SceneController</td>
<td>Select a scene by name. Some limits on discovery based on device type.</td>
</tr>
<tr>
<td>Speaker</td>
<td>SetVolume, AdjustVolume, SetMute</td>
</tr>
<tr>
<td>StepSpeaker</td>
<td>AdjustVolume, SetMute (a subset of “Speaker” with no absolute volume, for when range of volume is not known).</td>
</tr>
<tr>
<td>CameraStreamController</td>
<td>Start and stop streaming video</td>
</tr>
</tbody>
</table>
AVS Home Skill Semantics: System Messages

Directives can also get or create messages with various types corresponding to “system capabilities”.

<table>
<thead>
<tr>
<th>Capability</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discovery</td>
<td>Every device: reports initial identification and capabilities</td>
</tr>
<tr>
<td>Authorization</td>
<td>AcceptGrant, provides bearer token, currently only OAuth2.0 supported</td>
</tr>
<tr>
<td>Response</td>
<td>Respond to directive - success</td>
</tr>
<tr>
<td>ErrorResponse</td>
<td>Respond to directive – error/failure, gives type of error, message for logging. Errors can include system issues (eg connectivity) but also payload issues (value out of range).</td>
</tr>
<tr>
<td>ReportState</td>
<td>Request state report</td>
</tr>
<tr>
<td>StateReport</td>
<td>Reply to state report</td>
</tr>
<tr>
<td>DeferredResponse</td>
<td>Used to indicate that will respond asynchronously to request (typ. &gt;8s needed)</td>
</tr>
<tr>
<td>EndPointHealth</td>
<td>Check if physical endpoint can be reached from cloud shadow. Has associated Boolean property, “connectivity”</td>
</tr>
</tbody>
</table>
AVS Issues and Notes

- “Device Cloud” (IoT service endpoint) must support HTTPS and be visible to AWS cloud
- OAuth2.0 authentication must be supported.
- Responses cannot take longer than 8s.
  - Live querying of the device status, if it takes longer than that, needs to use a shadow. The shadow state then needs to be proactively or asynchronously updated.
  - Actions that take longer than 8s to complete should use an asynchronous response to indicate completion.
- Property status reporting
  - Not just the value, but also the timestamp, time uncertainty, and units
- Optional features:
  - Status query, proactive status updates, scenes and scene discovery, streaming video and audio
3. Fog Integration

Goal: Make services discoverable and locally available using WoT combined with fog capabilities.

- Run WoT services on fog (OpenFog) software stack
  - Thing directory (WoT discovery)
  - Metadata bridges (eg to OCF)
  - Voice skill services (eg AVS)
  - Application servients (IoT services)
- Connect to computational services
  - Recognition service (eg Movidius NCS)
3.5 Fog Integration

Goal: Provide accelerated compute services in the fog.

- Hardware and software stack
  - Fog Reference Design (FRD)
  - Kaby Lake CPU and Arria 10 FPGA
  - DLIA and OpenCL software libraries
- Host accelerated compute service
  - WoT Servient (virtual Thing)
  - Discovery via Thing Directory
  - Other management services needed
3.x Fog Integration

Add: Compute services running on local and remote fog nodes

- Fog services may be on either side of the NAT
- Other services may also run in fog nodes instead of the cloud or gateway
- (Not shown) Fog compute services register with Thing Directory and run as WoT Servients
To Do: Industrial POCs

• Voice control is primarily a home use case
• What do we want for industrial use cases?
• What is different about industrial requirements?
  • Set of standards and their requirements are different
  • Real-time, pub-sub architectures, TSN, functional safety, central management, asset management, complex access controls, energy management…
  • Less emphasis on privacy, more on safety
• Are municipal and building use cases more like home or industrial use cases?
• What about automotive? Transportation? Retail? Medical?
  • They all have their own ecosystems…
Scenarios

Need to work through some specific scenarios…

- Voice Control for Consumer End Users
- Service Composition for System Integrators
- Compute Services (eg person recognition) for other Service Providers

Include not only home, but also industrial and municipal

- Long tail – translation to older protocols
- Adapt to different requirements, eg real time and functional safety

→ Need to talk to industrial users about requirements
Smart Security Scenario: Person Detection

This specific scenario can be demonstrated using OCF Smart Home capabilities, combined with one or more IP cameras. This application is of use in multiple verticals (home (eg security system), industrial (eg functional safety), municipal (eg subway track ingress).

Problem: Simple proximity detection systems, such as motion sensors, can be triggered by events other than human presence, leading to false alarms. However, constantly running AI-based human recognition is expensive.

Solution: Combine simple distributed motion sensors with AI-based person recognition. When a motion sensor is triggered, an image is captured in that location and sent to a person recognition service. Only if a person is detected is an alarm triggered. In addition, the AI service can be centralized so that only one recognition service is needed even if there are multiple cameras and motion sensors. As an extension, cameras could be mounted on mobile platforms (aka robots…).
Summary

• Interoperability has significant business value in IoT
• The Web of Things approach supports interoperability
  • By describing system interfaces (as opposed to prescribing them)
  • Main deliverable: a universal metadata format (“Thing Description”) for IoT services (“Things”)
• *We need to work though some concrete scenarios (POCs)*
• Ideally, include fog computing in system architecture
WoT: Related W3C Standards

- **RDF**: Resource Description Framework
  - General mechanisms for defining data semantics and vocabularies
  - Useful for working with metadata
- **SSNO**: Semantic Sensor Network Ontology
  - Vocabularies (ontologies) and semantics for sensor data
- **JSON-LD**: JSON (JavaScript Object Notation) Linked Data
  - Mechanism for encoding RDF in JSON, used for Thing Description serialization
- See also:
  - iot.schema.org: Vocabularies and semantics for IoT, defined using RDF
Web of Things: Resources and Links

W3C: World Wide Web Consortium: https://www.w3.org

Web of Things Interest Group: https://www.w3.org/WoT/IG/
  • Charter: Leverage web standards and technology to enable IoT interoperation
  • Web architecture: https://www.w3.org/standards/webarch/

Web of Things Working Group in the W3C to develop standard recommendations:
  • https://www.w3.org/2016/09/wot-wg-charter.html
  • Co-chairs: Matthias Kovatsch (Siemens), Kazuo Kajimoto (Panasonic), Michael McCool (Intel)
  • White paper on WoT architecture: http://w3c.github.io/wot/charters/wot-white-paper-2016.html

WoT current practices: http://w3c.github.io/wot/current-practices/wot-practices.html