One Data Model
Semantic Definitions for Connected Things

T2TRG – WoT
Workshop on IoT Semantics @IETF 106

November 14, 2019

What is One Data Model?

- A loose organization of SDOs, Device vendors, IoT Platform operators, and IoT experts
- Goal is to harmonize IoT semantic models across SDOs and vendors
- Heavy participation from connected home sector
- Initially – a common "language" for IoT semantic models, usable by application domain experts
- Eventually - convergence of semantic definitions for common IoT device types, broad adoption of the language

History

- Emerged from Zigbee "Hive" meeting, fall 2018
- Cross-industry consensus on lack of common IoT data models as a key inhibitor to IoT growth
- Broad industry group of SDOs and vendors
- No legal organization – working under a liaison
- Weekly teleconferences, 4 face to face meetings, in 2019
- Working in a github repository
- Language, tools, and models

Process

- Create a common representation language for existing IoT data and interaction models
  - Enable contribution of the best existing models across all participating organizations
- Collect a set of representative models for a "pressure test" of the language
  - Convert to the new language and note any gaps
- Organizations contribute models for evaluation
  - Process for selecting a single model per function, e.g. lighting, door lock, thermostat
- Publication of selected models

Status

- Weekly technical meetings since December 2018
- Four face to face meetings
- Diverse models are being used to test the language
- At the October 2019 Face to Face meeting we approved a version of the modeling language to proceed with contributions
  - SDF - Simple Definition Format
- Set up an area for contributions to be uploaded and evaluated

Outcomes

- All participants have agreed to publish the models under the BSD 3-Clause Open Source license
- 2-way translation between OMA LWM2M XML models and the SDF language
- SDF models for Dotdot thermostat and lighting clusters are in progress, energy clusters next
- SDF models for OCF appliances are in progress
- OCF may use the SDF language as the "entry point" for developers to create and maintain data models
  - Automatic mapping to OCF styled Swagger definitions

What is a semantic model – Practical IoT Semantics

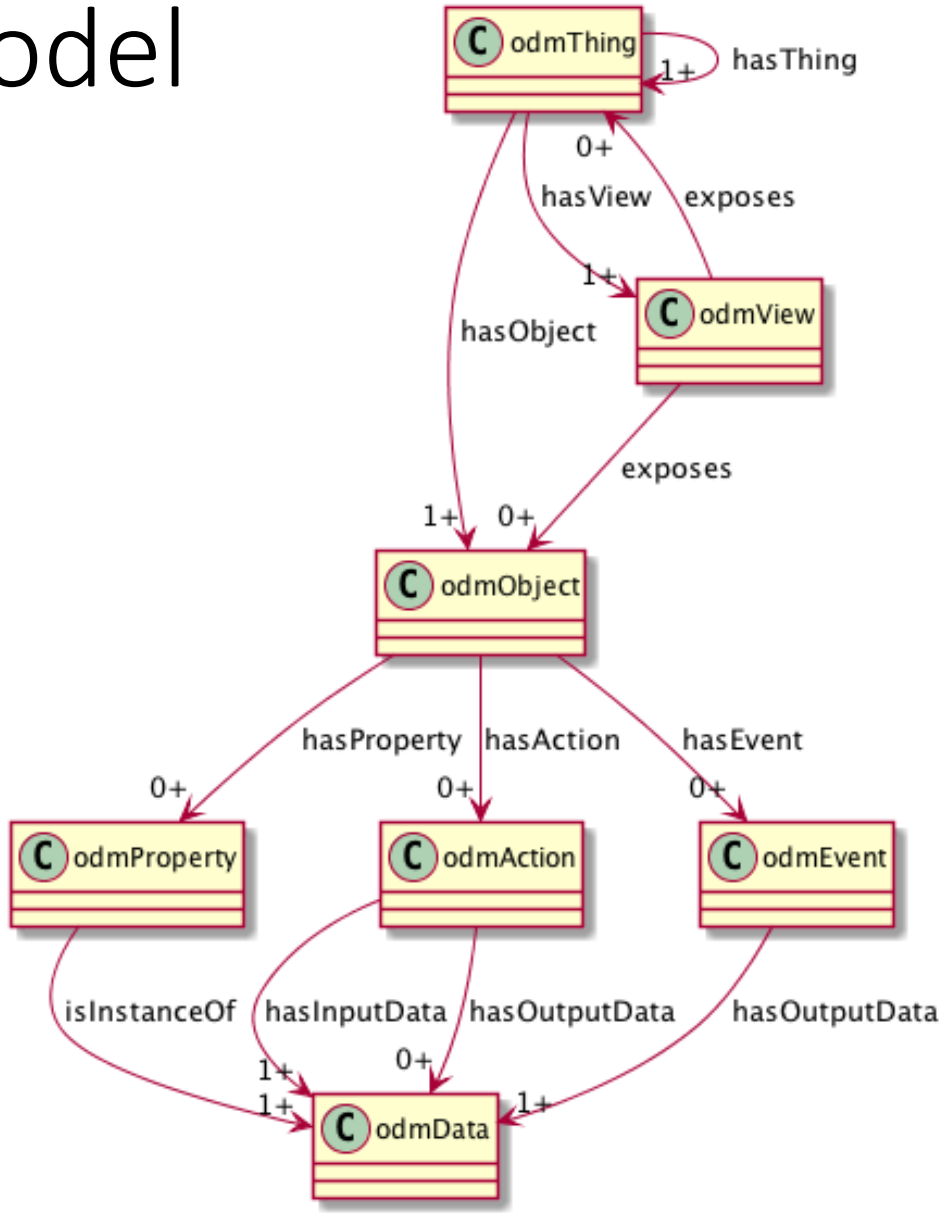
- Abstract meta-model for IoT device affordances, behavior, and context
  - Decoupled from network bindings, protocol-agile
  - Common categories for affordances
  - Common categories for constraints
  - Common format for definitions
- Initial focus on affordances to normalize device-facing interactions across SDOs and vendors
- Behavioral and contextual models also are needed but not in the initial scope

ODM Architecture

- Meta-model
- ODM Classes
  - odmData
  - odmProperty
  - odmAction
  - odmEvent
  - odmObject
  - odmThing
  - odmView

ODM Meta-Model

- Thing Class to compose Objects
- View (Interface) Class to virtualize affordances
- Reusable Objects
  - Property, Action, and Event Affordances
- Reusable Data Types



odmProperty

- Elements that represent the state of a connected thing – direct affordance to instances of Data Types
- Read and Write meta-operations on Data elements
- Read meta-operation returns the representation of state
- Write meta-operation uses a supplied representation to update state
- For example, the operational mode of a thermostat

odmAction

- Affordance, usually to control a physical world effector that is associated with a connected thing
- Also emulates function calls
- Invoke meta-operation with zero or more input data parameters
- Output data returns information including status of long running actions
- E.g. locking or unlocking a door lock

odmEvent

- An affordance to obtain happenings associated with the connected thing, often to receive asynchronous or unsolicited notification messages
- Subscribe meta-operation to map to most protocols, e.g. CoAP Observe, MQTT Subscribe, HTTP long poll or eventSource
- Could be notifications of state changes, also alerts and alarms
- Output data contains state or application messages

odmData

- Reusable definitions for data types – JSON Schema
- May use the same odmData definition for odmProperty as for odmAction input data, odmEvent output data
- Defines a semantic type, e.g. temperatureData, and basic data type (number, string, boolean), with additional constraints (enum, minimum, maximum, number of decimal places, etc.), and associations with quantities and units
- Well known types for date, time, URL, UID, etc.

odmObject

- A collection of Properties, Actions, and Events
- Work together to perform some discrete function
- On/off switch, light dimming, color control, door lock/unlock
- odmObject is the main point of commonality for interoperability
- Similar functional odmObjects have similar affordances
- Defines minimum required set of affordances

odmThing

- A collection of odmObjects and odmThings that work together in a complementary way
- A light control thing may have on/off switch control, dimming control, and color control objects
- A product thing may have several light things and other types of things, allowing for nested modular composition patterns

odmView (Interface)

- odmView is for defining virtual interfaces to a thing
- For example, an interface may have limited functionality traded off for small data size over the air
- odmViews are composed of odmObjects and odmThings that are defined elsewhere in the model, with additional qualities to constrain and define the behavior of the interface

SDF Language Design Review

- Overview – Functional structure
- Definitions and Declarations
- References using JSON Pointer
- Processing model – namespaces and files
- High Level Composition

SDF Design Overview

- JSON based DSL – JSON Schema validation
- Associates semantic terms with type definitions of ODM classes
- Example odmObject definition for a simple binary (on/off) switch control
  - The odmObject for "Switch" object has three affordances:
  - odmProperty for state "value" with a defined string enum allowing "on" and "off" values
  - odmActions for "on" and "off" (that implicitly act on the "State" Property)

SDF - Simple Definition Format

```
{
  "info": {
    "title": "Example file for ODM Simple JSON Definition Format",
    "version": "20190404",
    "copyright": "Copyright 2019 Example Corp. All rights reserved.",
    "license": "http://example.com/license"
  },
  "namespace": {
    "st": "http://example.com/capability/odm#"
  },
  "defaultNamespace": "st",
  "odmObject": {
    "Switch": {
      "odmProperty": {
        "value": {
          "type": "string",
          "enum": ["on", "off"]
        }
      },
      "odmAction": {
        "on": {},
        "off": {}
      }
    }
  }
}
```

Simple example – Info and namespace definitions

keywords

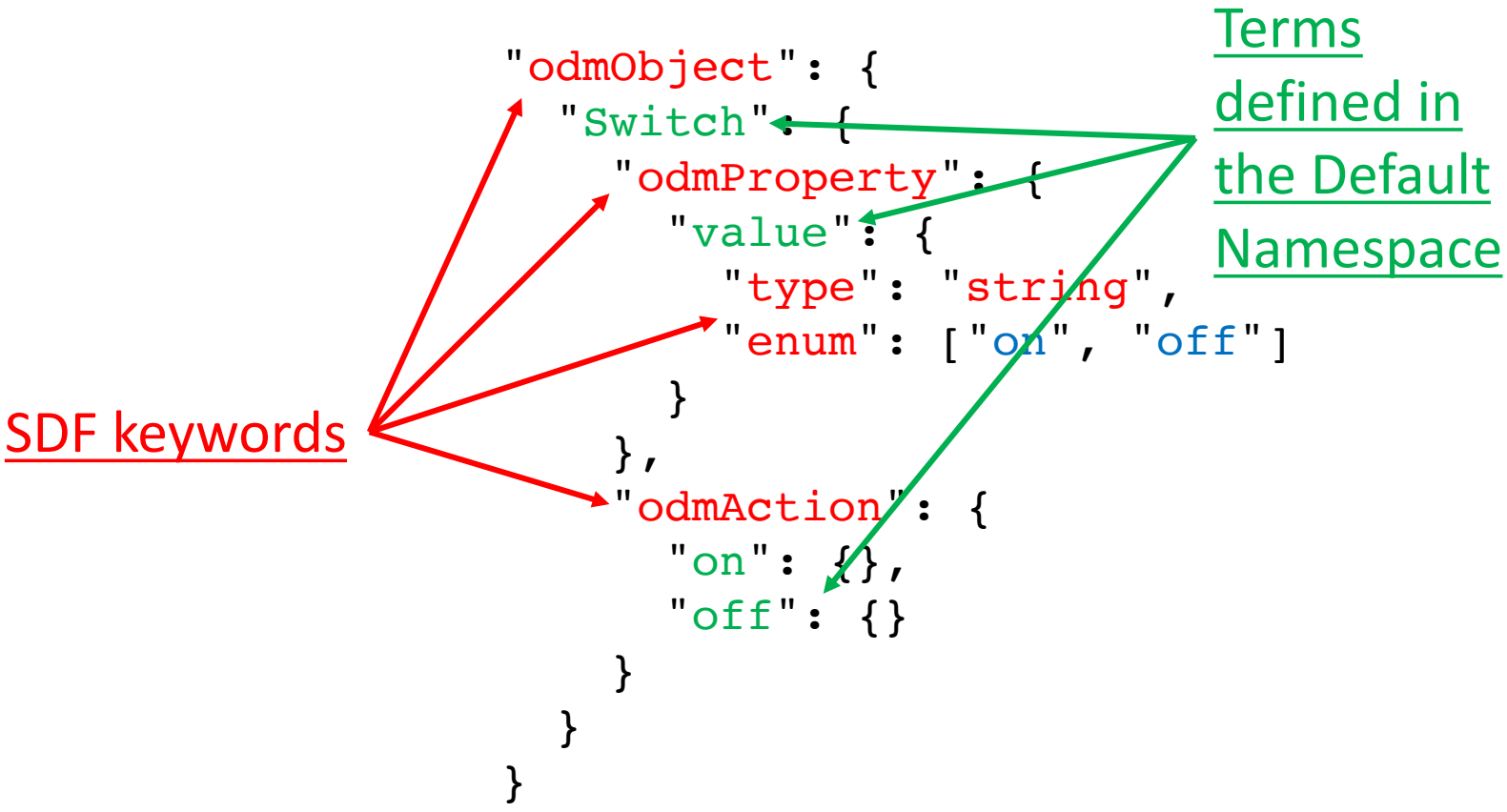
File Information

```
"info": {
  "title": "Example file for ODM Simple JSON Definition Format",
  "version": "20190404",
  "copyright": "Copyright 2019 Xcorp, Inc. All rights reserved.",
  "license": "http://example.com/license"
},
```

curies resolved

```
"namespace": {
  "ocf": "http://example.org/ocf/odm",
  "st": "http://example.com/capability/odm"
},
"defaultNamespace": "st",
```

Definitions



Definitions

- A definition consists of a defined term and a map of it's defined qualities
- JSON Schema syntax is used for odmProperty and odmData constraint qualities

```
"value": {
  "type": "string",
  "enum": ["on", "off"]
}
```

Declarations

- A Declaration in SDF is some use of a defined term
- Usually in another definition, through reuse of definitions
- A declaration can also be an inline definition, within another definition
- In the above example, "value" is a definition with its own declared qualities, as well as a declaration within the "Switch" definition
- Are statements about qualities in a definition also declarations?

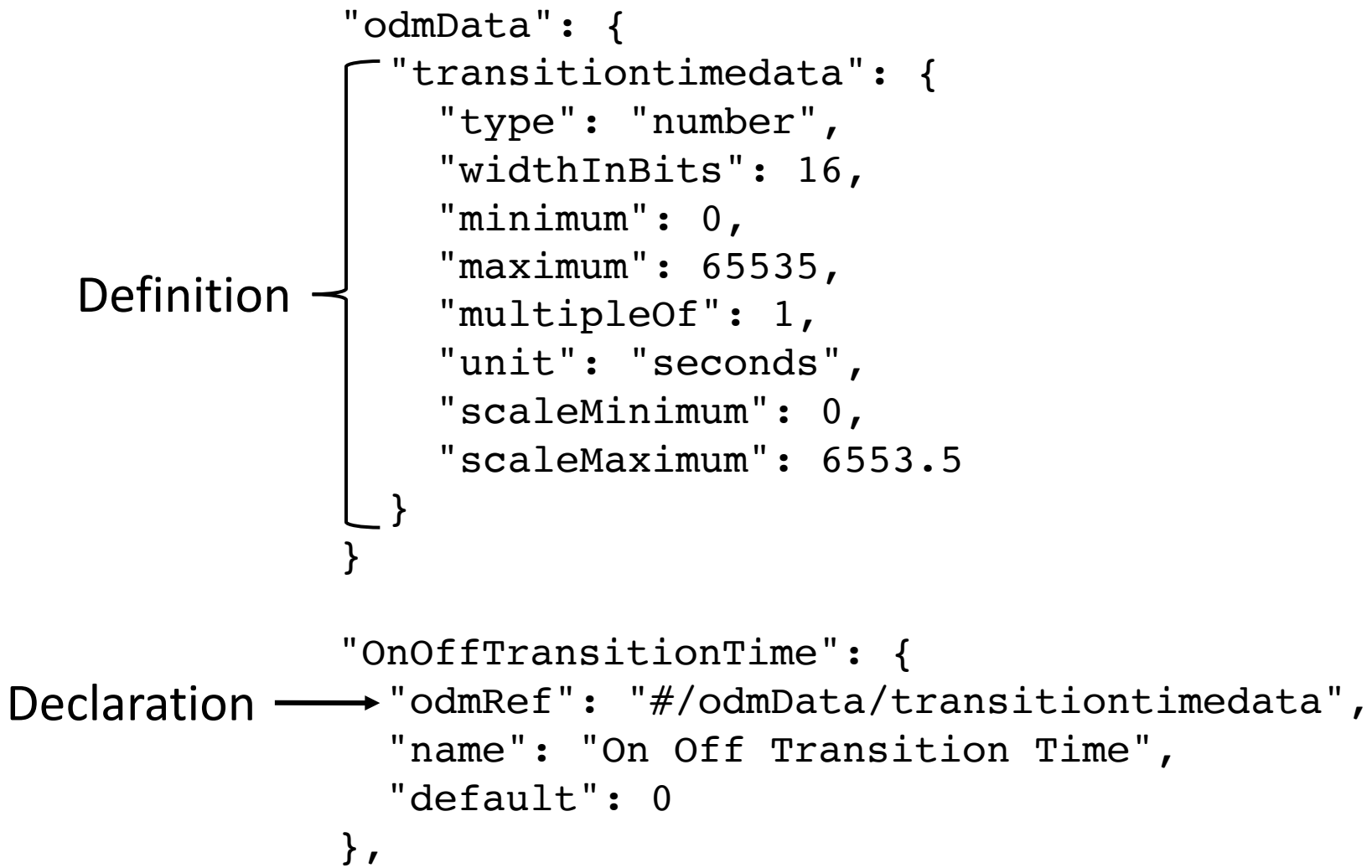
SDF References

- The defined qualities, and the semantic identity, of a definition can be re-used in a new definition
- For example, a common definition for Transition Time Data can be used for timing parameters of different Actions in a lighting control model
- Reuse of definitions in SDF models is achieved through references, using JSON Pointer (RFC6901)
- Some examples use JSON Relative Pointers (I-D)

odmRef keyword

- Functions in a similar way as #ref in JSON schema
- Can be thought of as copying the qualities of the referenced definition into the current definition
- Additional qualities may be defined, e.g. semantic tagging, in the current definition

odmRef Example



Processing Model – Files and Namespaces

- Multiple SDF files are expected to be submitted to populate a namespace
- The defaultnamespace declaration in each SDF file determines the destination namespace location of the definitions that are in the file
- Lookup operations on the namespace will behave as if there is one file that contains all of the definitions in that namespace
- Accepting a definition file into a namespace is agreeing to roll it into the single file image

Data Type Definition

```
{
  "info": {
  "title": "Example ODM Data Type definition",
  "version": "20190504",
  "copyright": "no copyright",
  "license": "not licensed"
  },
  "namespace": {
    "zcl": "http://example.com/zcl/odm#"
  },
  "defaultnamespace": "zcl",
  "odmData": {
    "transitiontimedata": {
      "type": "number",
      "widthInBits": 16,
      "minimum": 0,
      "maximum": 65535,
      "multipleOf": 1,
      "unit": "seconds",
      "scaleMinimum": 0,
      "scaleMaximum": 6553.5
    }
  }
}
```

Example use of definition from a namespace

- Reference doesn't need to know about file names or how a definition was contributed
- Namespace prefix in the reference is expanded to a URL prefix before JSON Path processing

```
"namespace": {
  "zcl": "http://example.com/zcl/odm#"
},

"MoveToTiltTransitionTime": {
  "odmRef": "zcl:odmData/transitiontimedata",
  "name": "Move To Tilt Transition Time",
  "default": 0
},
```

Other use of JSON Pointer in SDF

- Indicate sub-sets of definitions that are required or designated as input or output data

```
"odmAction": {
  "MoveToLevel": {
    "name": "Move to Level",
    "odmRequired": [
      "0/odmData/level",
      "0/odmData/transitiontime"
    ],
    "odmInputData": [
      "0/odmData/level",
      "0/odmData/transitiontime"
    ],
```

```
"odmData": {
  "level": {
    "name": "Level",
    "type": "number",
    "widthInBits": 8,
    "minimum": 0,
    "maximum": 254
  },
  "transitiontime": {
    "name": "Transition Time",
    "odmRef":
"#/odmData/transitiontimedata"
  }
```

High Level Compositions

- odmThing
  - Contains definitions of odmObjects and, recursively, odmThings
- odmView (Interface)
  - Models a set of already-defined odmObjects and odmThings that are composed into a virtual interface
  - Defined within an odmThing, can only refer to other elements defined or declared within the odmThing in which the odmView is defined
  - Can define overlapping or disjoint functional subsets of an odmThing, or expose specialized representations for compactness, privacy, etc. – e.g. a read-only view

Thing Example

```
"odmProduct": {
  "SKU_19934774": {
  "productTypeListing": "Compound Vacuum Gauge",
  "required": [
    "0/odmObject/3300~10"
  ],
  "odmObject": {
    "3300/0": {
      "odmRef": "#/odmObject/genericSensor",
      "odmProperty": {
        "minimumRange": {
        "const": -120000
        },
        "maximumRange": {
          "const": 120000
        },
        "applicationType": {
          "const": "Vacuum Gauge"
        },
        "sensorType": {
          "const": "absolutePressure"
        },
        "units": {
          "const": "Pa"
        }
```

odmView Example

```
"odmView": {
  "oic.if.s": {
    "isDefaultView": true,
    "odmComponent": [
      "#/odmObject/oic.r.temperature/odmProperty/temperature",
      "#/odmObject/oic.r.temperature/odmProperty/units"
    ]
  },
  "oic.if.a": {
    "odmComponent": [
      "#/odmObject/odmProperty/temperature",
      "#/odmObject/odmProperty/units"
    ]
  },
  "oic.if.rw": {
    "odmComponent": [
      "#/odmObject/odmProperty/temperature",
      "#/odmObject/odmProperty/units",
      "#/odmObject/odmProperty/range",
      "#/odmObject/odmProperty/step",
      "#/odmObject/odmProperty/precision"
    ]
  },
```

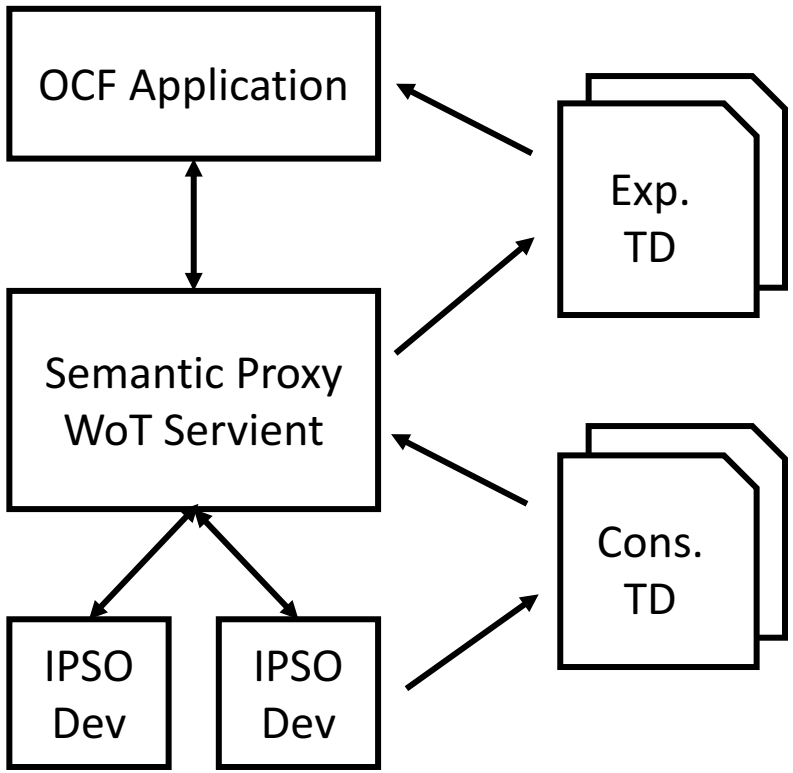
Next steps

- More updates to the language in progress
- Model convergence across vendors, SDOs
- Demonstration based on translation and gateway
- Public announcement soon, timing discussion
- Semantic Proxy project – W3C WoT integration

Semantic Proxy – W3C Web of Things Integration

- "Thing Description" associates semantic identifiers for Properties, Actions, and Events with affordance descriptions consisting of data schemas and protocol bindings
- Protocol bindings associate network operations with meta-operations in the semantic model
- "Incoming" Consumed TD and "Outgoing" Exposed TD have the same affordances in the semantic model, and customized data schemas and protocol bindings

Semantic Proxy - Schematic



- Exposed Thing TDs have OCF protocol binding
- Consumed Thing TDs have IPSO protocol binding
- Both TDs have the same meta interactions and operations defined in OneDM models

Semantic Proxy – RDF Converter

- Create RDF statements from OneDM definitions for use in semantic tooling
- iotschema style RDF definitions are aligned with the meta-model
- WoT Thing Description can use iotschema definitions for annotation
  - WoT TD only has Thing and affordance (P/A/E) classes
- odmObject maps to iotschema Capability
- odmThing and odmView don't directly map but can extend iotschema Capability

References

One Data Model SDF and Model work in progress

- https://github.com/one-data-model/language
- https://github.com/one-data-model/playground

Semantic Proxy and W3C WoT

- https://github.com/tum-ei-esi/virtual-thing
- https://www.w3.org/TR/2019/CR-wot-thing-description-20191106/
- https://www.w3.org/TR/2019/CR-wot-architecture-20191106/