One Data Model
Semantic Definitions for Connected Things

July 17, 2020

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
- July 2020: oneDM public
  - SDF standardization in IETF (ASDF BOF created)
  - Playground contains 200+ SDF contributed models from

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
- 2-way translation between OCF OAS2.0 models and the SDF language
- Initial SDF models for Zigbee cluster available
- Initial SDF models for Bluetooth available
- 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

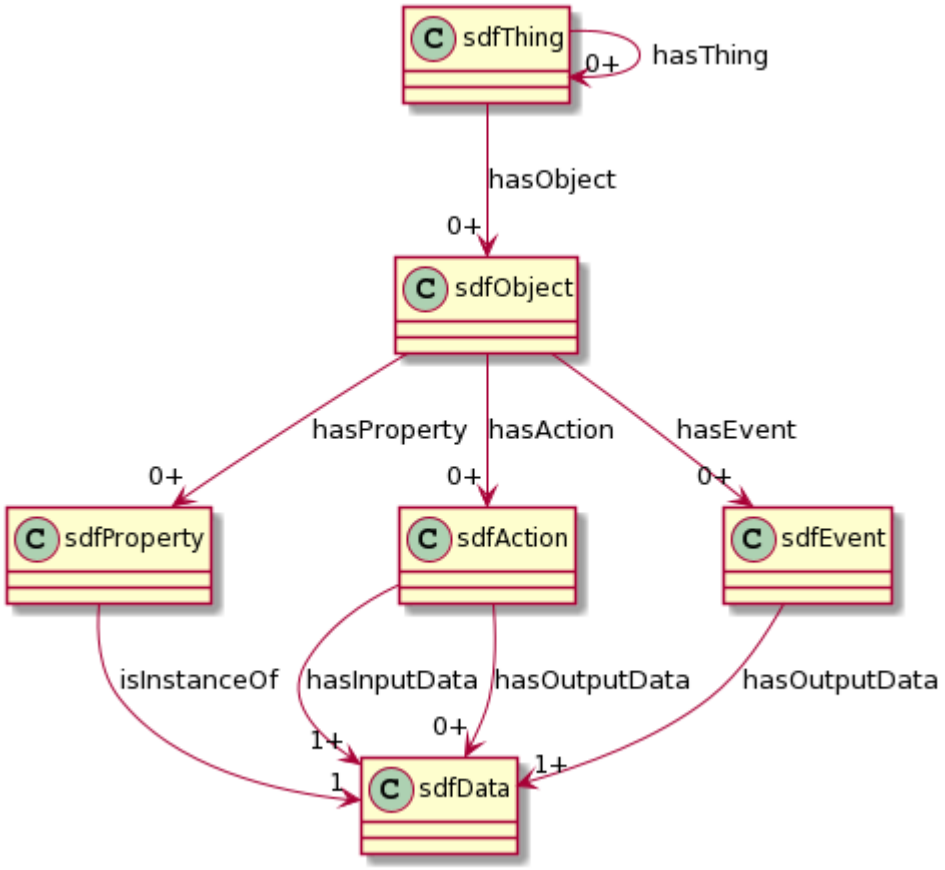
- Abstract meta-model for IoT device affordances, behavior, and context
  - Decoupled from network bindings, protocol-agile
  - Common categories for affordances
  - Common categories for constraints
  - Common format for definitions
- Initial focus on affordances to normalize device-facing interactions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

SDF Architecture

- Meta-model
- SDF Classes
  - sdfData
  - sdfProperty
  - sdfAction
  - sdfEvent
  - sdfObject
  - sdfThing

SDF Meta-Model

- Thing Class to compose Objects
- Reusable Objects
  - Property, Action, and Event Affordances
- Reusable Data Types



sdfProperty

- 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

sdfAction

- 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

sdfEvent

- 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

sdfData

- Reusable definitions for data types – JSON Schema
- May use the same sdfData definition for sdfProperty as for sdfAction input data, sdfEvent 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.

sdfObject

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

sdfThing

- A collection of sdfObjects and sdfThings 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

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 sdf classes
- Example sdfObject definition for a simple binary (on/off) switch control
  - The sdfObject for "Switch" object has three affordances:
  - sdfProperty for state "value" with a defined string enum allowing "on" and "off" values
  - sdfActions for "on" and "off" (that implicitly act on the "State" Property)

SDF - Simple Definition Format

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

Simple example – Info and namespace definitions

keywords

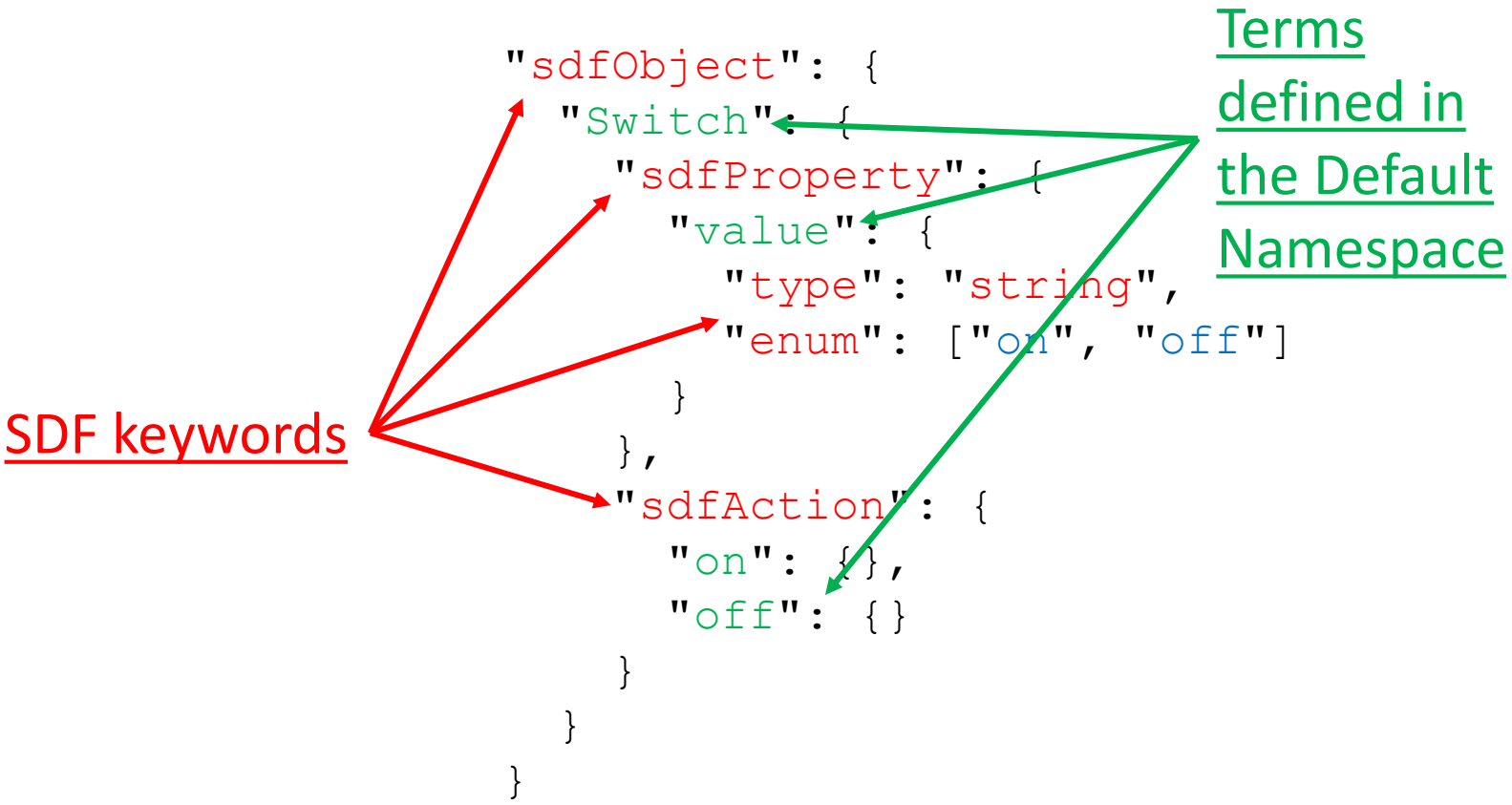
File Information

```
"info": {
  "title": "Example file for sdf 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/sdf",
  "mynamespace": "http://example.com/capability/sdf"
},
"defaultNamespace": "mynamespace",
```

Definitions



Definitions

- A definition consists of a defined term and a map of it's defined qualities
- JSON Schema syntax is used for sdfProperty and sdfData 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 syntax (RFC6901)

sdfRef 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

sdfRef Example

Definition

```
"sdfData": {
  "transitiontimedata": {
    "type": "number",
    "widthInBits": 16,
    "minimum": 0,
    "maximum": 65535,
    "multipleOf": 1,
    "unit": "seconds",
    "scaleMinimum": 0,
    "scaleMaximum": 6553.5
  }
}
```

Declaration

```
"OnOffTransitionTime": {
  "sdfRef": "#/sdfData/transitiontimedata",
  "name": "On Off Transition Time",
  "default": 0
},
```

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 sdf Data Type definition",
  "version": "20190504",
  "copyright": "no copyright",
  "license": "not licensed"
  },
  "namespace": {
    "zcl": "http://example.com/zcl/sdf#"
  },
  "defaultnamespace": "zcl",
  "sdfData": {
    "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/sdf#"
},

"MoveToTiltTransitionTime": {
  "sdfRef": "zcl:sdfData/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

```
"sdfAction": {
  "MoveToLevel": {
    "name": "Move to Level",
    "sdfRequired": [
      "#/sdfData/level",
      "#/sdfData/transitiontime"
    ],
    "sdfInputData": [
      "#/sdfData/level",
      "#/sdfData/transitiontime"
    ],
```

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

Next steps

- Standardize the language at IETF
- Update the language, e.g. more featurs
- Model convergence across vendors, SDOs
- Demonstration based on translation and gateway