# Semantic Proxy

Semantic Interoperability Demonstration using OneDM SDF, iotschema RDF, and W3C WoT TD

November 17, 2019

# Semantic Proxy

- A Proxy for Semantic Interoperability
  - Proxy to enable translation from device protocol to application protocol
  - Provides for many-to-many mapping of application protocols to device protocols
  - many-to-one and one-to-many through a common semantic model
  - Could implement a "universal" IoT gateway

# Meta Operations of Abstract Affordances

- Property
  - value = Read(), Write(value)
- Action
  - status response(s) = Invoke(parameters)
- Event
  - event occurrence responses = Subscribe()

# Semantic Proxy – Protocol Binding

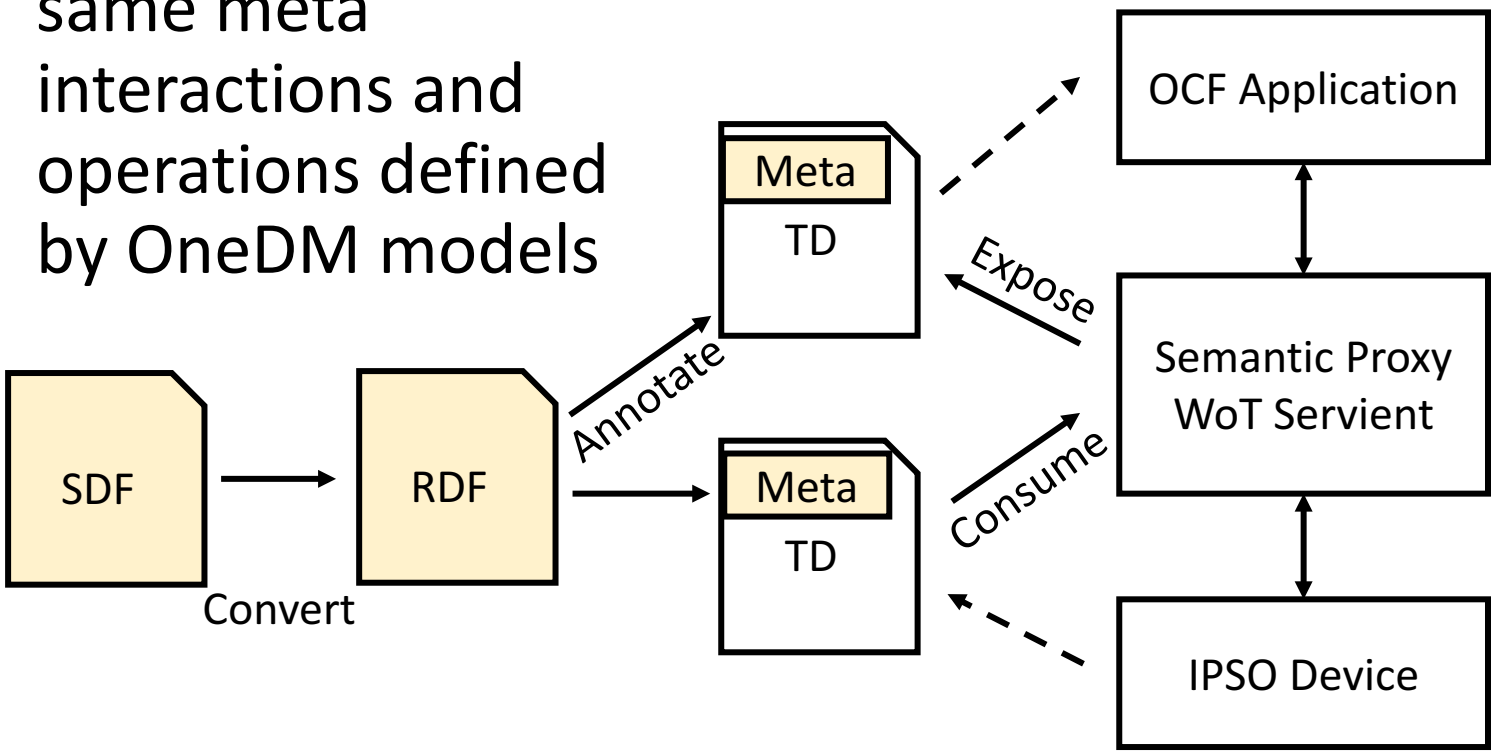
- Uses a common semantic model to connect applications to things over diverse network protocols and communication patterns
- Proxy maps the meta-model operations to network messages in the target protocol using protocol bindings
- Flavors of REST, Pub/Sub, RPC messages
- Example using W3C Web of Things Architecture

# 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

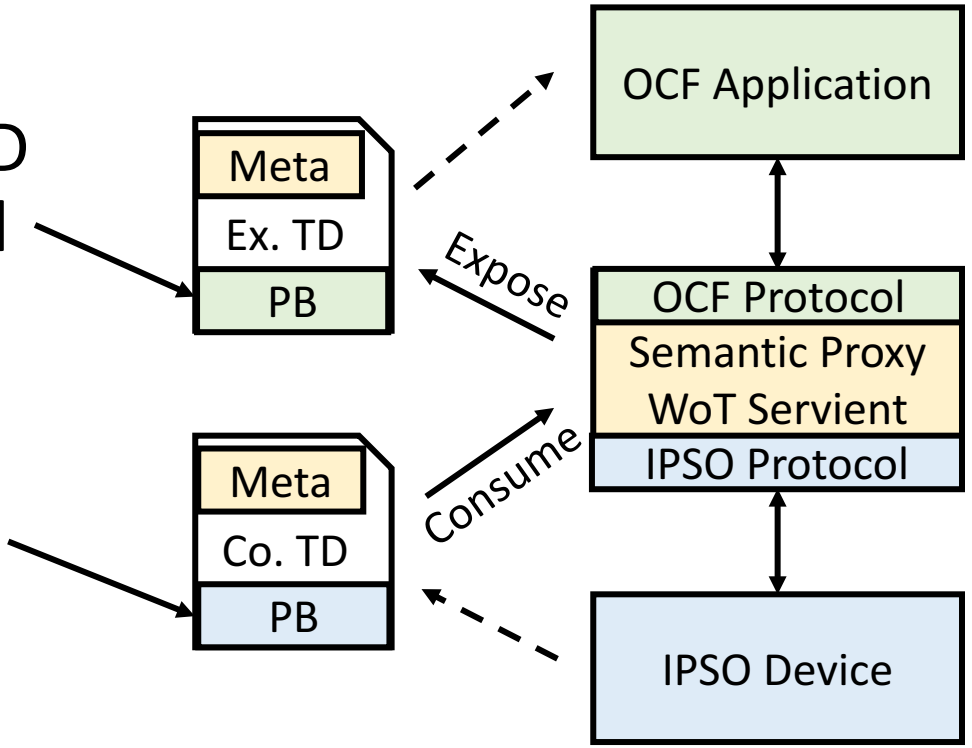
# Semantic Proxy - Schematic

- Both TDs have the same meta interactions and operations defined by OneDM models



# Semantic Proxy - Schematic

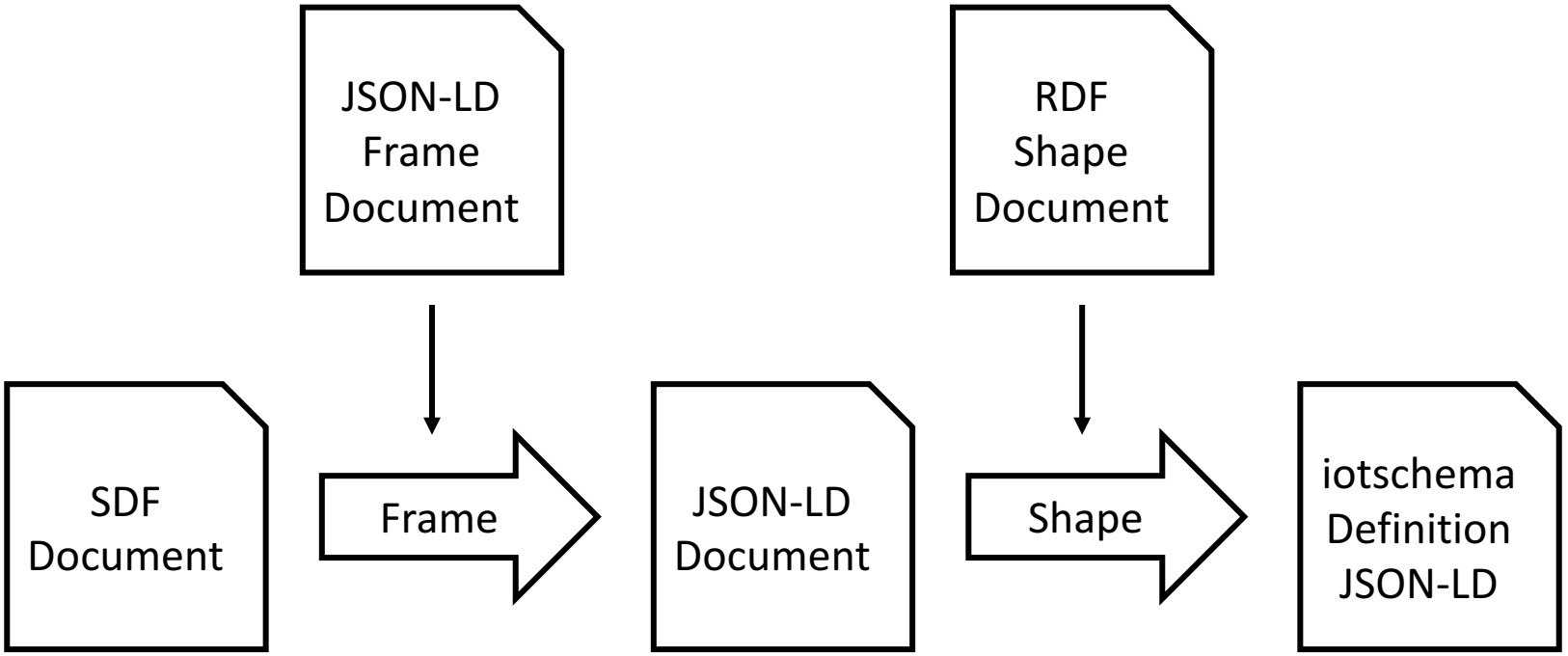
- Exposed Thing TD has OCF protocol binding
- Consumed Thing TD has IPSO protocol binding



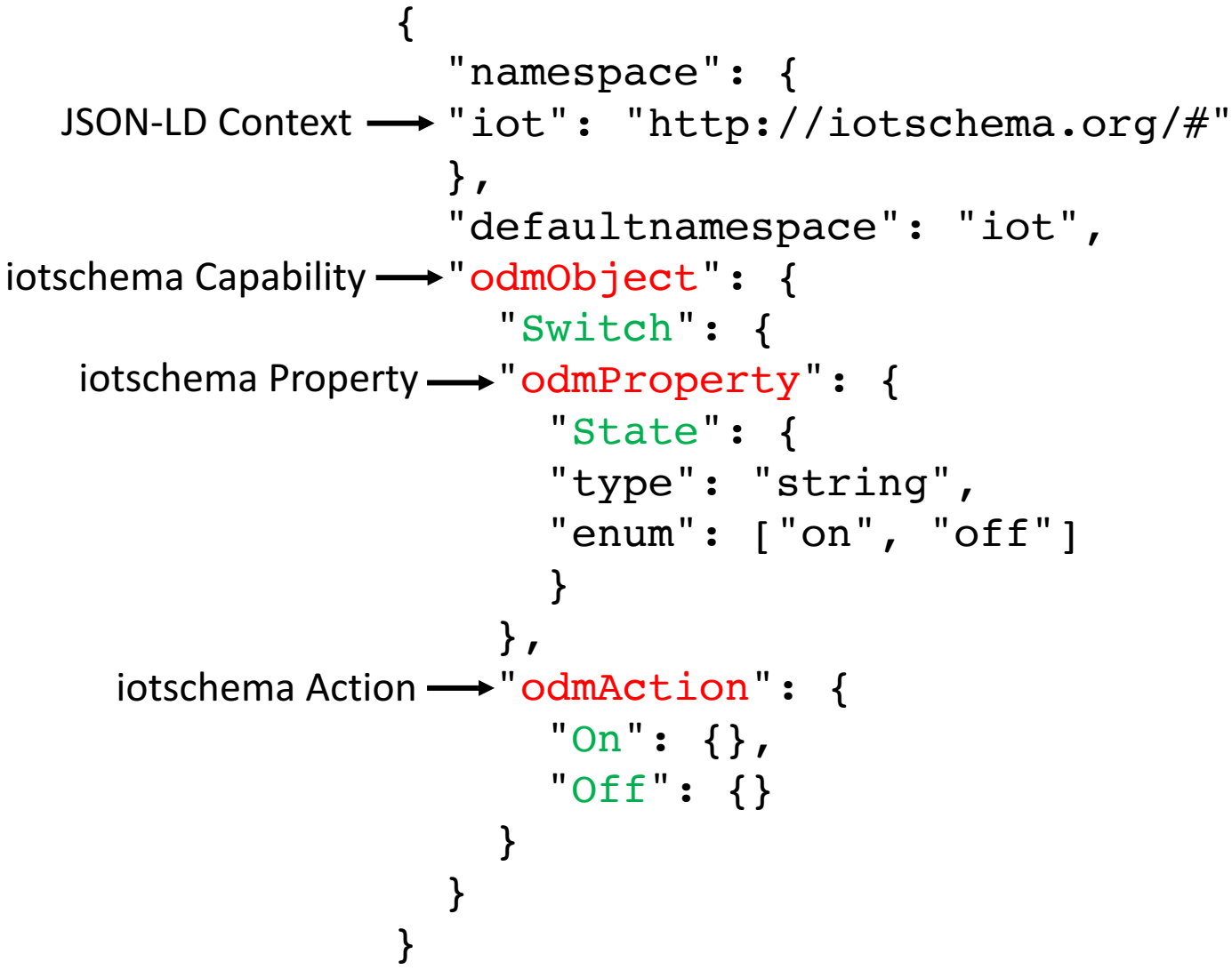
# Semantic Proxy – RDF Converter

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

# Convert SDF Documents to an iotschema style Definitions



# OneDM SDF Example Mapping




```
{
  "namespace": {
    "iot": "http://iotschema.org/#"
  },
  "defaultnamespace": "iot",
  "odmObject": {
    "Switch": {
      "odmProperty": {
        "State": {
          "type": "string",
          "enum": ["on", "off"]
        }
      },
      "odmAction": {
        "On": {},
        "Off": {}
      }
    }
  }
}
```

# Expected Result – Object maps to Capability

```
{
  "@id": "iot:SwitchCapability",
  "@type": "rdfs:Class",
  "rdfs:label": "SwitchCapability",
  "rdfs:subClassOf": {
    "@id": "iot:Capability"
  },
  "iot:providesInteractionPattern": [
    {
    "@id": "iot:SwitchStateProperty",
    "@id": "iot:SwitchOnAction",
    "@id": "iot:SwitchOffAction"
    }
  ]
}
```

# Expected Result - Actions

```
{
  "@id": "iot:SwitchOnAction",
  "@type": "rdfs:Class",
  "rdfs:label": "SwitchOnAction",
  "rdfs:subClassOf": {
    "@id": "iot:Action"
  }
},
{
  "@id": "iot:SwitchOffAction",
  "@type": "rdfs:Class",
  "rdfs:label": "SwitchOffAction",
  "rdfs:subClassOf": {
    "@id": "iot:Action"
  }
}
```

# Properties and Data Types

```
{
  "@id": "iot:SwitchStateProperty",
  "@type": "rdfs:Class",
  "rdfs:label": "SwitchStateProperty",
  "rdfs:subClassOf": {
    "@id": "iot:Property"
  },
  "iot:providesOutputData": {
    "@id": "iot:SwitchStateData"
  }
},
{
  "@id": "iot:SwitchStateData",
  "@type": "rdfs:Class",
  "rdfs:label": "SwitchStateData",
  "rdfs:subClassOf": {
    "@id": "schema:PropertyValue"
  },
  "schema:propertyType": {
    "@id": "schema:String"
  }
}
```

# Rexpected Result - Enums

- How do we describe enums in iotschema style RDF?

# Enum Data Item and Type in RDF

```
{
  "@id": "iot:SwitchStateData",
  "@type": "rdfs:Class",
  "rdfs:label": "SwitchStateData",
  "rdfs:subClassOf": {
    "@id": "schema:PropertyValue"
  },
  "schema:propertyType": {
    "@id": "iot:SwitchStateType"
  }
}
{
    "@id": "iot:SwitchStateType",
    "@type": "rdfs:Class",
    "rdfs:comment": "Enumeration of Switch states",
    "rdfs:label": "SwitchStateType",
    "rdfs:subClassOf": {
      "@id": "schema:Enumeration"
    }
},
```

# Enum Data Values in RDF

```
{
    "@id": "iot:SwitchOnState",
    "@type": "iot:SwitchStateType",
    "rdfs:comment": "Switch On State",
    "rdfs:label": "SwitchOnState"
},
{
    "@id": "iot:SwitchOffState",
    "@type": "iot:SwitchStateType",
    "rdfs:comment": "Switch Off State",
    "rdfs:label": "SwitchOffState"
}
```

# Semantic Proxy – WoT TD


```
{
  "@context": [
    "https://www.w3.org/2019/wot/td/v1",
    {"iot": "http://iotschema.org/"}
  ],
  "@type": [ "Thing", "iot:SwitchCapability" ],
  "properties": {
    "SwitchStateProperty": {
      "@type": ["iot:SwitchStateProperty", "iot:SwitchStateData"],
      "type": "string",
      "enum": ["on", "off"],
      "writeOnly": false,
      "readOnly": false,
      "observable": true,
      "forms": [
        {
         "href": "/SwitchCapability/properties/SwitchStateProperty",
         "op": ["readproperty", "writeproperty", "observeproperty"],
         "contentType": "application/json"
        }
      ]
    }
  },
```

# Semantic Proxy – WoT TD

```
"actions": {
  "SwitchOnAction": {
    "@type": ["iot:SwitchOnAction"],
    "input": {},
    "forms": [
      {
        "href": "/SwitchCapability/actions/SwitchOnAction",
        "op": ["invokeaction"],
        "contentType": "application/json"
      }
    ]
  },
  "SwitchOffAction": {
    "@type": ["iot:SwitchOffAction"],
    "input": {},
    "forms": [
      {
        "href": "/SwitchCapability/actions/SwitchOffAction",
        "op": ["invokeaction"],
        "contentType": "application/json"
      }
    ]
```

# Semantic Proxy – Semantic API

- Names of affordances are resolved through Semantic Discovery
  - PropertyName=discover(FilterParameters)
- Applications use meta-model affordances and operations
  - data=readProperty(PropertyName)
  - writeProperty(PropertyName, data)
  - result=invokeAction(ActionName, parameters)
  - data=subscribeEvent(EventName)
- Supports modular, declarative programming models – Node-RED

# 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/