iot.schema.org Community Teleconference

Michael Koster

Darko Anicic

Jason Koh

Agenda

1. Agenda bashing

2. Updates:

- Brief report out from the WISHI Semantic Interoperability hackathon
- https://github.com/t2trg/wishi/wiki/IETF-102-Hackathon

3. Semantic API

- Using iot.schema.org definitions and JSON Schema to annotate data
- Semantic API using iot.schema.org definitions as resource selectors

4. RDF Shapes constraint generator - Darko Anicic

5. Proposal for BRICK - Jason Koh

6. Community - contribution process:

- Discuss the process for accepting new definitions
- Contribution agreement (since we now have external contributions)
- Incoming area (folder) for proposals
- Validation check (automated CI process on PRs for contributions?)
- Design Review
- Example: Incoming contributions from SmartThings

7. Materials for admin workstream (ongoing)

- Charter for W3C WoT CG
- iot.schema.org One Pager

WISHI Hackathon

- July 14th and 15th at IETF101, Montreal
- 8 participants
- Devices from OCF, OMA LWM2M
- Protocols CoAP, HTTP, MQTT
- Used W3C Thing Description and iot.schema.org definitions

Technical Components (1)

- Mediatypes
  - CoRE Link-Format and Web Linking (RFC6690, RFC8288)
  - WoT Thing Description
  - OMA LWM2M
  - SenML
  - JSON
- Protocols
  - HTTP
  - CoAP
  - MQTT
  - DNS-SD

Technical Components (2)

- Software Components
  - Thingweb - node-wot
  - Thingweb - Thing Directory
  - CoRE Resource Directory
  - Node-RED
- Some Bridged Ecosystems
  - OCF
  - LWM2M
  - IKEA Lighting
  - Philips Hue

Projects

- IPSO/LWM2M mapping using WoT Thing Description and iot.schema.org
- OCF mapping using WoT Thing Description and iot.schema.org
- RD Implementation
- W3C Wot Protocol Bindings to CoAP+DTLS devices
- Semantic wrapper for W3C WoT Scripting API
- DNSSD Integration

Example Semantic Annotation

```
{
  "@context": [
    "http://w3c.github.io/wot/w3c-wot-td-context.jsonld",
    "http://w3c.github.io/wot/w3c-wot-common-context.jsonld",
  {"iot": "http://iotschema.org/"}
  ],
  "base": "coap://example.net:5683/",
  "@type": [ "Thing", "iot:TemperatureCapability" ],
  "name": "Temperature Sensor",
  "interaction": [
    {
      "name": "Temperature",
      "@type": ["Property", "iot:Temperature"],
      "outputData":  {
        "type": "object",
        "field": [
          {
            "name": "temperature",
            "@type": ["iot:TemperatureData"],
            "type": "number",
            "minimum": -50,
            "maximum": 100,
            "unit": "Celsius"
          }
        ]
      }
```

Some Results

- Breakout discussion on high level work items/areas
- Demonstrated interoperation between generic clients and diverse devices
- Closed 44 issues with node-wot implementation and moved to Eclipse Foundation
- Got RD implementation up to speed and ready to integrate Thing Directory functionality
- Demonstrated automatic interaction with diverse CoAP+DTLS servers
- Report in progress

Semantic API

For abstract interaction over diverse ecosystems, adaptation is needed for Transfer Layers, Serialization Formats, and Data Types

1. Adaptation to Transfer Layer formats is provided by Forms element processing
2. **DataInstance class library allows automatic adaptation to diverse serialization formats (OCF, LWM2M, SenML) by embedding a Data Item dictionary that contains Semantic Annotation**
3. Adaptation to Data Type and Scale + Units may be provided by a DataItem adaptation class

DataInstance Abstract Class

- DataInstance is a representation in some mediatype, which is also a transfer layer payload
- Described by a DataSchema
- Contains one or more DataItem as dataProperty
- Actions and Events exchange DataInstance representations
- Instance of an Interaction Property is an instance of DataInstance
- Interaction Property may also be a instance of a DataItem, providing get() and set() decorators

Semantic annotation and Schemas

- DataItems (variables) in DataInstance Schemas are identified by Semantic Annotation in "@type" values
- Schemas are used to validate and interpret incoming payloads
- Schemas are used to construct outgoing payloads
- Schema validator can be extended to emit a dictionary of DataItems that can be referenced using the Semantic Annotation
- Library can be used to create a Semantic API wrapper for the WoT Scripting API

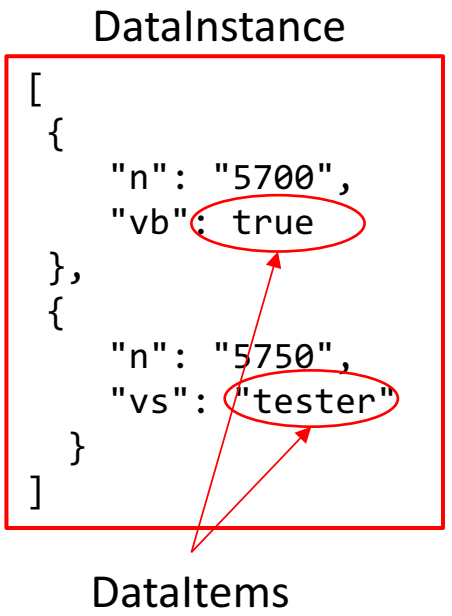
Example Schema with Annotation

```
{
  "type": "array",
  "allOf": [
    {
      "contains": {
        "type": "object",
        "properties": {
          "n": {
            "type": "string",
            "const": "5700"
          },
          "vb": {
            "type": "boolean",
            "@type": "iot:SwitchData"
          }
        },
        "required": ["n", "vb"]
      }
    },
    {
      "contains": {
        "type": "object",
        "properties": {
          "n": {
            "type": "string",
            "const": "5750"
          },
          "vs": {
            "type": "string",
            "@type": "iot:ApplicationTypeData"
          }
        },
        "required": ["n", "vs"]
      }
    }
  ]
}
```

DataInstance Dictionary

- Contains a JSON Pointer and sub-schema for each DataItem in a DataInstance
- Example for a SenML DataInstance

```
[
  {
    "path": "/0/vb",
    "@type": "iot:BinarySwitchData",
    "type": "boolean"
  },
  {
    "path": "/1/vs",
    "@type": "iot:ApplicationTypeData",
    "type": "string"
  }
]
```



Semantic API Examples

```
// Semantic Lookup returns instances capable of semantic lookup
thing = local-directory.lookup-by-simple-template;
light = thing( {"@type": ["iot:Light", "BinarySwitchCapability"] } )
switch = light.property( {"@type": "iot:BinarySwitch"} )
rgbcolor = light.property( {"@type": "iot:RGBColor"} )
turnon = light.action( {"@type": "iot:TurnOnAction"} )
setlevel = light.action( {"@type": "iot:SetLevelAction"} )

// read() function with and without DataItem filter
>>> console.log( switch.read( {"@type": "iot:BinarySwitchData"} ))
true

>>> console.log( switch.read() )
[{ "@type": "iot:BinarySwitchData", "value": true },
  { "@type": "iot:ApplicationTypeData", "value": "tester" }]

// write() function
switch.write( {"@type": "iot:ApplicationTypeData", "value": "Light"} )
```

Semantic API Examples (2)

```
// Write of multiple DataItems in a structured DataInstance
rgbcolor.write( [
  {"@type": "iot:RedColorData", "value": 255},
  {"@type": "iot:GreenColorData", "value": 255},
  {"@type": "iot:BlueColorData", "value": 255} ] )

// invoke() function
turnon.invoke()

setlevel.invoke( [{"@type": "iot:LevelData", "value": 170},
{"@type": "iot:TransitionTimeData", "value": 100}] )

// chained semantic references
>>> console.log( thing({"@type": ["iot:Light","BinarySwitchCapability"]})
.property({"@type": "iot:BinarySwitch"})
.read({"@type": "iot:BinarySwitchData"}) )
true
```

(Presentations)

Contributions

- Discuss the process for accepting new definitions
- Contribution agreement (since we now have external contributions)
- Incoming area (folder) for proposals
- Validation check (automated CI process on PRs for contributions?)
- Design Review
- Example: Incoming contributions from SmartThings