How W3C Web of Things and OGC SensorThings API can work together and benefit from each other

W3C Workshop on the Web of Things, 3-5 June 2019, Munich, Germany

Michael Jacoby (Fraunhofer IOSB), Hylke van der Schaaf (Fraunhofer IOSB), Josh Lieberman (Open Geospatial Consortium), Kathi Schleidt (DataCove e.U.)

Introduction

OGC SensorThings API[1] (STA) and W3C Web of Things (WoT) are both standards with the goal to improve interoperability in the Internet of Things (IoT). Although STA has its focus on and origin in the sensing domain and WoT in the web domain, they fit together quite well. This position paper presents some initial findings on how they relate and how they could benefit from each other. It is intended to act as a starting point for further discussion and action.

What is STA and how does it work?

OGC SensorThings API (STA) provides an open, geospatial-enabled and unified way to interconnect IoT devices, data and applications over the Web. Currently, it consists of two parts: Part I covers the sensing domain and Part II the actuation/tasking domain. Especially the sensing part is based on and motivated by existing, proven-working and widely adopted open standards from the sensing community such as OGC Sensor Web Enablement (SWE) and ISO/OGC Observations and Measurements. It picks up the main ideas of OGC Sensor Observation Service (SOS) and OGC Sensor Planning Service (SPS) and adapts them to fit into the connected world by applying the principles of the Web (HTTP, JSON, and hypermedia). Since STA was officially published in 2016, it has gained a lot of attention, especially in the environmental sensing community. It is currently used in many different scenarios and projects, most of them in the context of smart cities and environmental sensing.

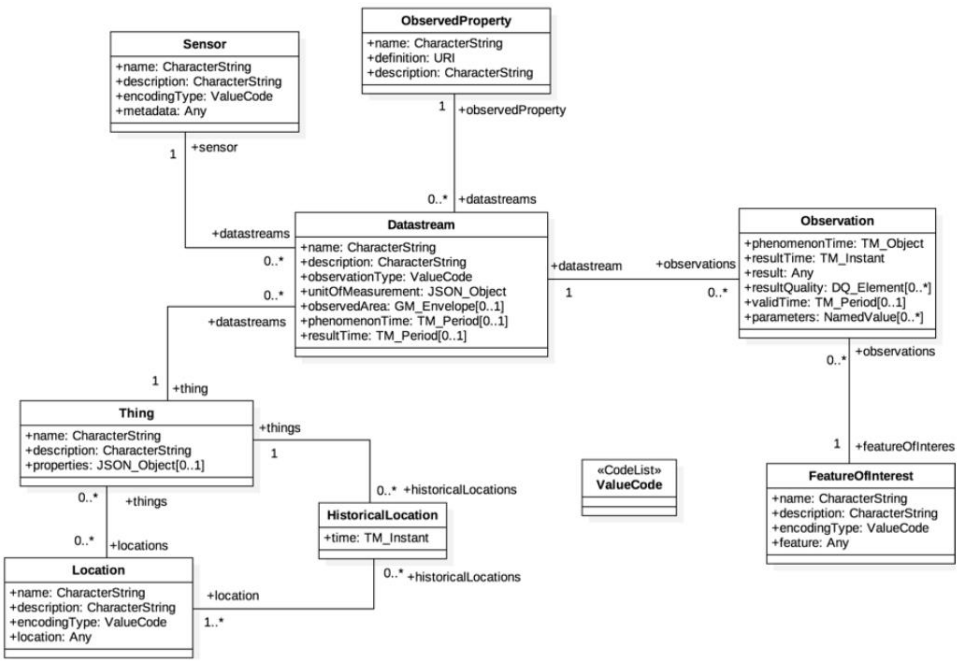

Figure 1: SensorThings API data model.

[1] http://docs.opengeospatial.org/is/15-078r6/15-078r6.html

From a technical perspective, STA is heavily influenced by the OASIS Open Data Protocol 4.0[2] (OData). It defines a data model as shown in Figure 1 and a REST-based API describing how to manipulate (collections of) instances of these classes, called entities, and how to query them. The syntax of the REST-base API is based on OData. For example, to fetch all entities of an entity type one would call `HTTP GET http://example.org/STA/v1.0/Things` and to access a single entity (with ID 1) `HTTP GET http://example.org/STA/v1.0/Things(1)`. As result a JSON description of the requested resource(s) is returned, e.g.

```
{
   "@iot.id": 1,
   "@iot.selfLink": "http://example.org/STA/v1.0/Things(1)",
   "Locations@iot.navigationLink": "Things(1)/Locations",
   "Datastreams@iot.navigationLink": "Things(1)/Datastreams",
   "HistoricalLocations@iot.navigationLink": "Things(1)/HistoricalLocations",
   "name": "Oven",
   "description": "This thing is an oven.",
   "properties":
   {
      "owner": "Noah Liang",
      "color": "Black"
   }
}
```

By default, related entities (e.g. the Datastreams related to a Thing) are not inlined in the response but rather referenced via a so-called navigation links pointing to the location where to fetch these related entities. Manipulation of data works as know from the web via `HTTP POST/PATCH/PUT/ DELETE`. Subscribing to entity or property changes is possible via MQTT.

For finding resources, STA provides a powerful query language based on the OData query language plus additional functions especially for geospatial queries. The query is encoded using URL parameters, e.g.

```
http://example.org/STA/v1.0/Observations?$filter=result ge 10
```

to find all observations with value >= 10. To enhance expressivity of the query language and to reduce the number of HTTP calls needed to fetch a desired result, the query language supports deep filtering as well as the $expand and $select operators. Deep Filtering means one cannot only filter on properties of the current entity type but also on types that can be reached via navigation links, e.g.
`http://example.org/STA/v1.0/Observations?$filter=Datastream/observedProperty/ name eq 'temperature'`

The $select operator allows returning only the requested properties of an entity, e.g.
`http://example.org/STA/v1.0/Observations?$select=resultTime,result`

returns an array of all Observation entities but each only represented by resultTime and result like this

```
{ "value": [
      {"resultTime": "2019-04-01T10:20:00-07:00", "result": 42},
      ...
]}
```

The $expand operator however allows to expand/inline the content of a navigationLink. They can combined and nested in arbitrary depth, allowing to retrieve exactly the information needed, e.g. calling

[2] https://www.odata.org/

```
HTTP GET http://example.org/STA/v1.0/Observations?filter=result qe 10 and
Datastream/ObservedProperty/name eq 'temperature'&$select=resultTime,result&
$expand=Datastream($select=name,description)
```

which returns the following JSON

```
{ "value": [
   {
      "resultTime": "2019-04-01T10:20:00-07:00",
      "result": 42,
      "Datastream": {
         "name": "...",
         "description": "...",
      }
   },
   ...
]}
```

How do WoT and STA relate?

In general, WoT and STA are rather complementary than competitive. STA is a solution tailored especially to the sensing and actuation domain. This manifests in the fact that STA provides a domain-specific information model and is tied to concrete communication protocols (HTTP and MQTT). WoT however tries to bring together different existing APIs for the IoT by providing a meta-level description of things, their properties, services, events, and how they relate to other things. Therefore, STA can be seen as a perfect candidate API to be described by WoT.

A special issue is that they use a partially different paradigm to describe their view of the world. WoT uses a (almost) strictly state-based paradigm, meaning things only have a current state reflected by the current values of their properties which might change over time. STA also uses this paradigm a lot, e.g. for Things and Sensors. STA also uses an observation-based paradigm meaning that Things do not only have a state but there are lots of observations over time which may (in-)directly correspond to a state of a Thing. This approach is closer to how the real world works but also often more complex than needed when one is only interested in a more abstract view on things.

How can WoT benefit from STA?

Expose data in STA using WoT Thing Description

Thing Description (TD) should be able to describe even complex IoT APIs. Trying to describe STA using TD would be a real-world scenario to evaluate expressivity of TD. Furthermore, this could yield additional requirements for TD. Some conceptual work in this area has already been done and we identified at least two aspects where the current expressivity of TD might not be enough. The first one is about linking between Things or collections of Things, as well as manipulation of those links, especially adding and removing elements from collections. The second aspect is how to describe recursive datatypes. This is something very common in STA as you can see in the STA data model in Figure 1.

It is possible that these issues are already addressed in the latest version of TD. However, when going into the details, it is likely that even more issues can be identified which could help to improve the expressivity and usability of TD.

Embrace the (geospatial) nature of data

WoT is a web-based standard for interoperability in IoT. It focuses on providing a clean and easy-to-use description for Things, but at the same time neglect the (geospatial) nature of things. From a theoretical

and technical point of view, this is reasonable, but for any real-world application, the geospatial aspects of data are essential. This will become more evident the moment someone wants to search a ThingDirectory for Things nearby or within a given geospatial area. Furthermore, the state-based paradigm used to describe things might not be suited well enough to fit many real-world applications as state-transitions do not happen instantly and historic values are quite important in many scenarios. Therefore, from the experience gathered through multiple years of using STA in real-world use cases, we would advocate to re-think if adding support for those requirements to the standard would be an option.

Powerful resource discovery and efficient resource access

To our understanding, WoT will provide a web-based API for resource registration and discovery, called ThingDirectory, in the future. The query language to be used in such discovery requests is not yet determined, but SPARQL or CoRE Link Format seem to be promising candidates[3]. OData/STA offers a query language that allows combining data and metadata in one query, e.g. find sensors by observed property, location and latest sensor value. This has proven as very powerful and useful in real-world applications. Therefore, we propose to consider adding something similar for the ThingDirectory. As a ThingDirectory, unlike a STA Server, does not store the actual property values of a thing but only descriptions how to access them, the STA approach cannot be used directly.

Another strength of the OData/STA query language is the possibility to reduce the number of requests a client has to make to fetch all the data needed by providing the possibility to describe the desired output data by the means of $filter and $expand. Especially $expand helps to dramatically reduce the number of requests needed to fetch the desired information across multiple linked objects/collections of objects.

Although it is unclear if these features are applicable in the WoT universe and if so, if they would/should be part of the web-based API or rather part of a client library, we find this issue important enough to list them here to trigger a discussion.

How STA can benefit from WoT?

Adding more detailed semantics

Real-world use cases often need to add a domain- or application-specific information model on top of the STA model. The support for this in STA version 1.0 in rather limited but will be improved in the upcoming version 1.1 by adding a generic property called *properties* to all entities that can be used to store any additional custom data. Exposing entities of a STA server using WoT TD would be beneficial regarding the semantic expressivity in two aspects. First, TD makes use of semantic web technologies and therefore provides a more standardized and powerful way to semantically describe and annotate things. Second, it would make it possible to expose data collected with a STA server using any custom domain- or application-specific information model. Additionally, because of the different paradigms used, it would allow exposing data from an STA server in a state-based model, which in some scenarios is highly desirable. An example how to expose existing observation-based STA data in a state-based manner based on a custom information model using TD is shown below. In the example, the latest observation of a custom property of a certain thing is exposed as its property state. This example could be further extended, e.g. by means of adding a way to set a property value via TD which then results in creation of a new observation. Furthermore, information how to subscribe via MQTT could be added to the TD.

---

[3] https://w3c.github.io/wot-architecture/#terminology

```
{
   "@context": { "ex": "http://example.com/myModel/" },
   "name": "MyThing",
   "@type": ["Thing", "ex:myCustomType"],
   "id": "http://example.com/STA/v1.0/Things(1)",
   "properties": {
      "myCustomProperty": {
         "@type": "ex:myCustomProperty",
         "label": "myCustomProperty",
         "type": "number",
         "readOnly": true,
         "forms": [
            {
               "href":
"http://example.com/STA/v1.0/Observations?$filter=Datastream/Thing/id eq 1
and Datastream/ObservedProperty/name eq 'myCustomproperty'&
$orderBy=resultTime desc&$top=1&$select=result",
                  "mediaType": "application/json"
        }]
    }
  }
}
```

Interoperability and federations

Exposing a STA server using TD would also simplify integration with other devices and services. It would not only enhance interoperability between STA and other soft-/hardware but also make it possible to integrate different instances of STA server and expose their entities in a unified way. From our experience, some real-world scenarios require multiple STA servers that with the current features of STA cannot be integrated. This gap could be closed by adding TD on top.

Conclusion and Next steps

We have shown that WoT and STA are rather complementary than competitive and have the potential to both benefit from each other in multiple ways. As a first step, we would propose to try to describe entities of a STA server using TD in a more complete way than in the above example. This could be done in cooperation between the WoT and the STA group. Additional cooperation could focus on the design of the query language for ThingDirecotry and the integration of spatial data into WoT.