|  |  |
| --- | --- |
| Input Contribution | |
| Meeting ID\* | SDS#47 |
| Title:\* | OGC-to-oneM2M Data Mapping Issue |
| Source:\* | Ingo Friese, Deutsche Telekom AG, Ingo.Friese@telekom.de |
| Date:\* | 2020-12-03 |
| Input related to\* | SDS-2020-0141 |
| Intended purpose of  document:\* | Decision  Discussion  Information  Other <specify> |
| Impacted other TS/TR(s) |  |
| Decision requested or recommendation:\* | Add this to the TR0065 V 0.0.1 |
| Template Version: January 2020 (do not modify) | |

**oneM2M Notice**

The document to which this cover statement is attached is submitted to oneM2M. Participation in, or attendance at, any activity of oneM2M, constitutes acceptance of and agreement to be bound by terms of the Working Procedures and the Partnership Agreement, including the Intellectual Property Rights (IPR) Principles Governing oneM2M Work found in Annex 1 of the Partnership Agreement.

# Introduction

This contribution discusses the OGC-to-oneM2M Data Mapping Issue.

The following text explains what SensorThings API is and how it is working. Background section was already agreed but it was copied into the document for providing better context. However, some editorials have been corrected in the subsequent text too. The changes are marked by revision marks.

### -----------------------Start of corrections 1-------------------------------------------

# 5 Background

SensorThings API (STA) is a standard of the Open Geospacial Consortium (OGC). It provides a framework for communication and exchanging data between sensors and applications. The standard is divided in two parts. SensorThings API Part 1 is dedicated to sensing and was published in 2016. Part 2 deals with tasking and was published in 2019. There is an OGC certified Open-Source SensorThings API Server available (FROST Server). It supports the OGC SensorThings API Part 1: Sensing. It also includes preliminary actuation/tasking support. For the description of entities SensorThings API uses ISO 19156:2011 data model and JSON as data serialization format. The communication is REST-based and uses HTTP and also CoAP, MQTT, 6LowPAN.

## 5.1 SensorThings API Architecture

A typical STA-based architecture works in Client/Server mode. The sensor device pushes its data to the SensorThings Server via HTTP POST request. The SensorThings Server has also a MQTT broker. An interested application can subscribe to the MQTT-Broker in order to get notified about new sensor events.

The data in the SensorThings server are organized according to ISO 19156 (see Figure 1: ISO 19156 data model).

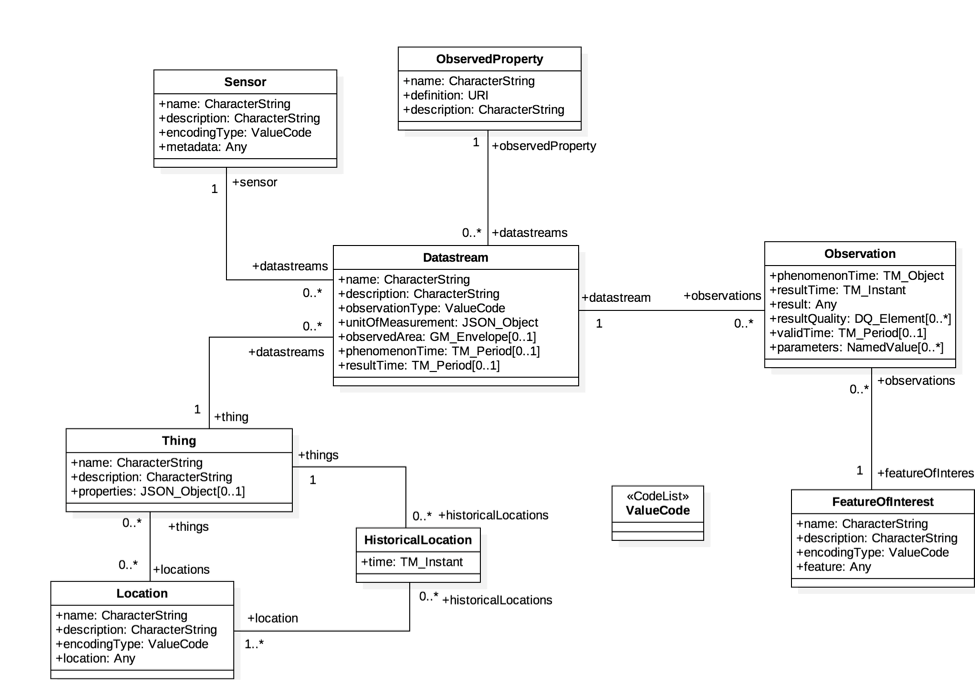


Figure 1: ISO 19156 data model

In the SensorThings data model an event or sensor data are called ‘observations’. Before a sensor is able to push an observation to the server it needs at least a ‘thing’- and a ‘datastream’ entity. This has to be created beforehand. One ‘thing’ might have different ‘sensors’, one ‘location’ or many ‘historical locations’.

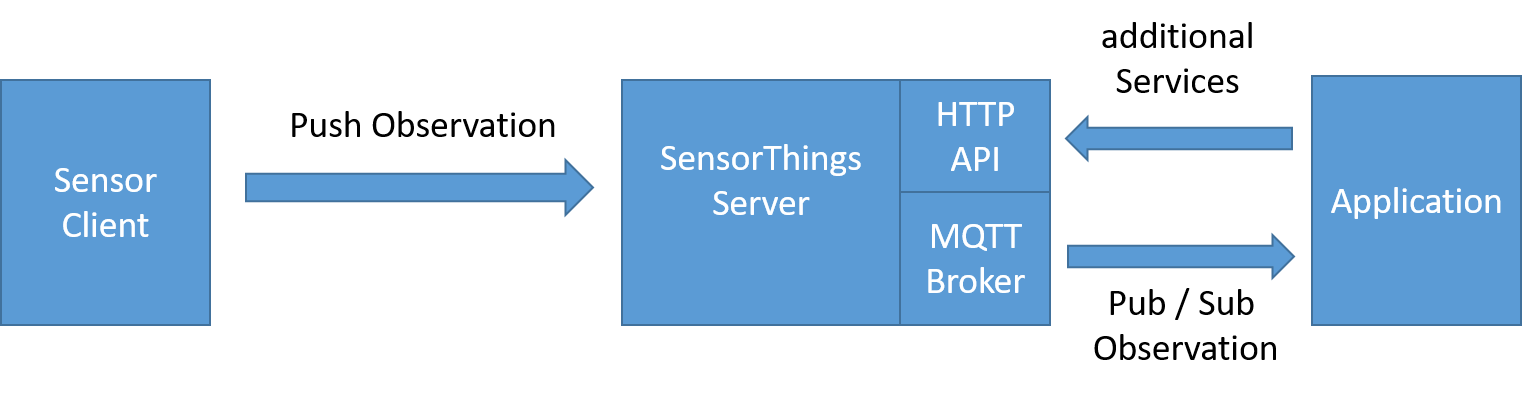


Figure 2: Exemplary STA message flow

## 5.2 SensorThings API example use-case

The message flow in Figure 2 can be explained by using an example of an application that wants to use data of EV-Charging Stations in a city.

1. In order to get all relevant observations belonging to the cities EV-Charging Stations, the application needs to know all regarded ‘datastreams’. Therefore the application need to send a request with filter parameter to the HTTP-API of the server e.g.:  
   https://sta-example-server-address.com/v1.0/Things?$filter=substringof(“Charging”,name)&$count=true&$expand=Datastreams

As a response the server provides a list of all data streams belonging to a charging station.

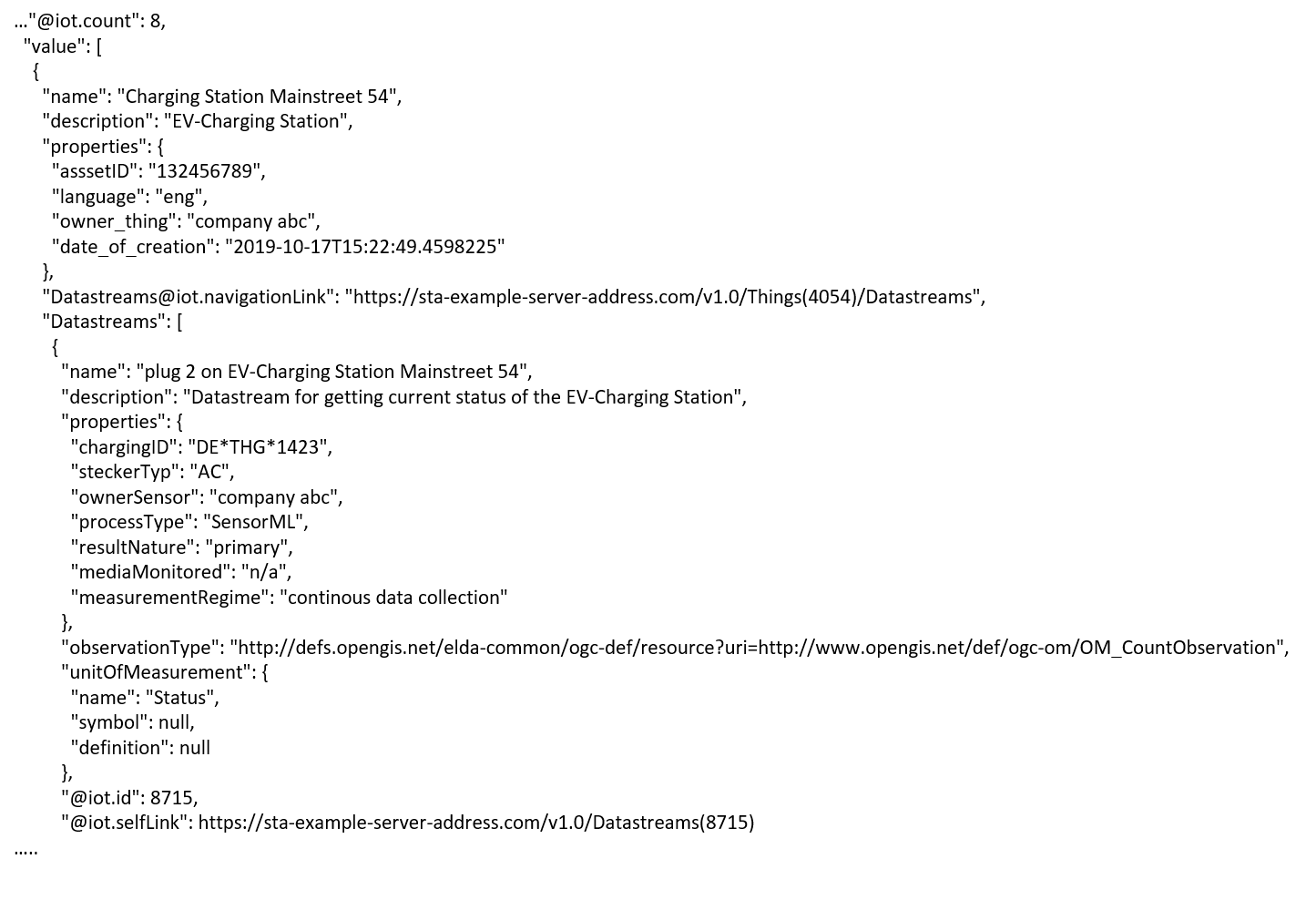


Figure 3: One exemplary entry of the result list of a filter request

1. The application can now subscribe to these ‘datastreams’. Figure 3 shows one entry of the result list. It represents an EV-Charging Station as a thing including the regarded data-stream in the last line of the entry.
2. As soon as the sensor (EV-Charging Station ) changes its status e.g. from “available” to “charging” it pushes an observation to the server.
3. The application gets the observation through a notification that is send by the MQTT-Broker. An example of an observation is shown in figure 4. In the result field is shown the current status of the EV-Charging Station “charging”.

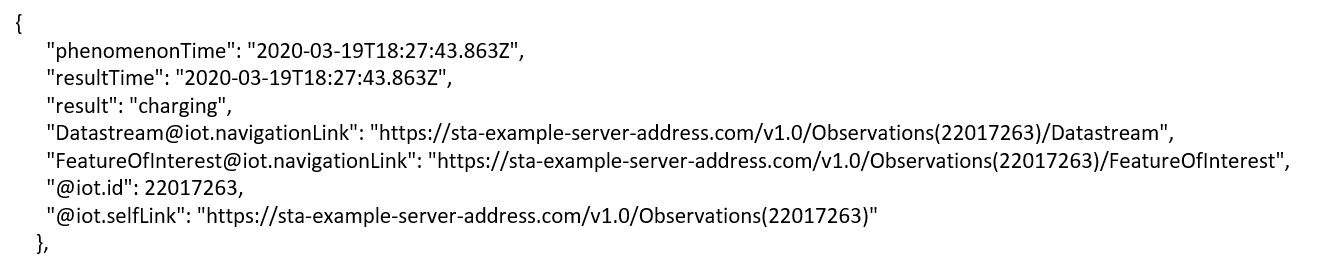


Figure 4: Exemplary STA observation

### -----------------------End of corrections 1-------------------------------------------

### -----------------------Start of change 1-------------------------------------------

# 6 Architecture Model of OGC/STA to oneM2M interworking

Figure 1 shows an architecture approach for an Interworking Proxy Entity (IPE) between oneM2M and OGC Sensor Things API. The IPE is located between a oneM2M CSE and an OGC/SensorThings API (STA)-Server.

The basic interworking enables applications that are connected to an oneM2M-based system to get data from sensors that are connected to an OGC/STA server. Furthermore, an application that is connected to an OGC/STA server is able to get data from sensors that are connected to an oneM2M-based system.

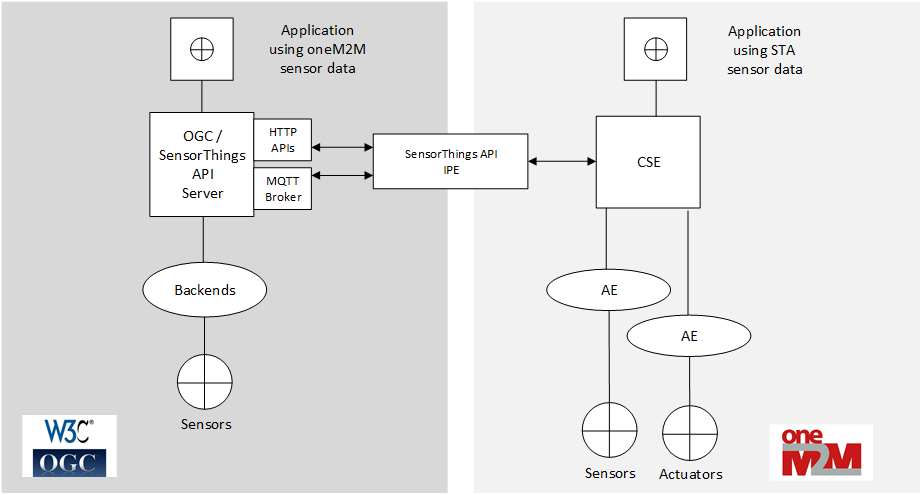


Figure 5: IPE architecture overview

## 6.1 OGC/STA-to-oneM2M Data Model Mapping

According to TS-0033 a representation of a non-oneM2M Proximal IoT function/device in a oneM2M-specified resource instance is to be synchronized with the entity that it represents.

This actual means that the OGC/STA data model has to be represented in the hosting CSE. The data in the OGC/STA server are organized according to ISO 19156 (see Figure 1: ISO 19156 data model).

The oneM2M structure for data models is a tree-structure where data is organized in containers or trees of containers. The OGC/STA data model has loops and is not hierarchical. Thus, it cannot mapped one-to-one to oneM2M or vice-versa.

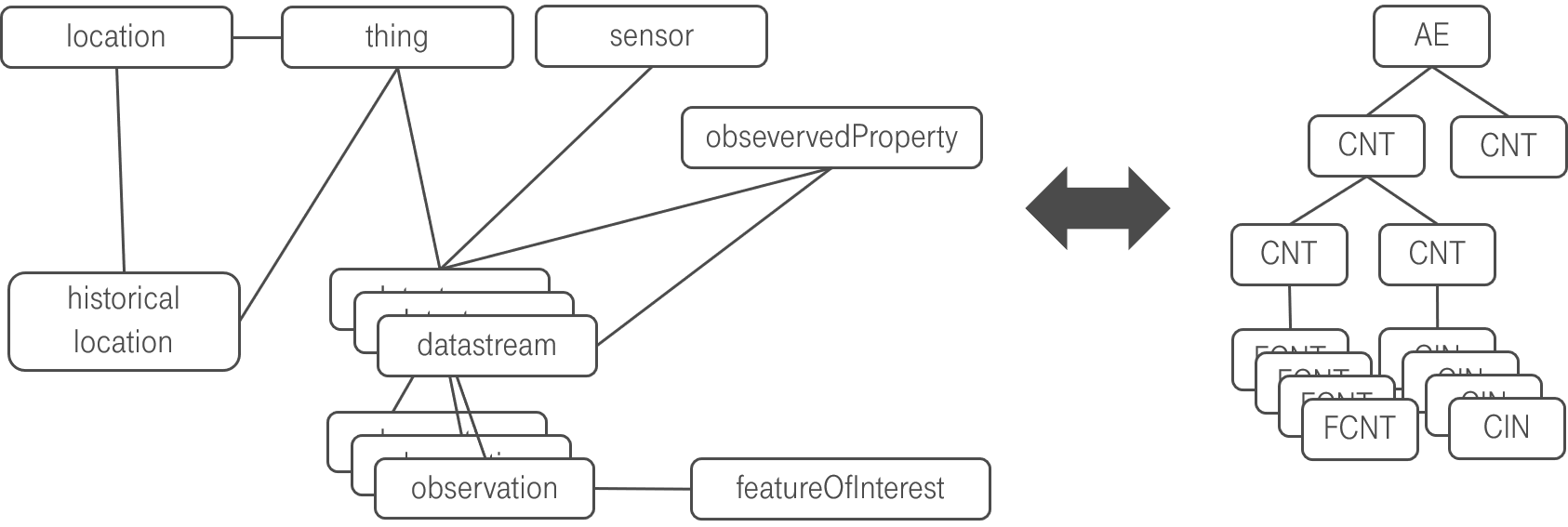


Figure 6: OGC data model cannot directly mapped to oneM2M

## 6.2. Architecture Approaches

The fact that the data model cannot directly be mapped has consequences for the architecture of an interworking between both standards. In the following sub chapters, we are going to discuss three different approaches and their advantages and disadvantages.

### 6.2.1 “Flat Data Model” Approach

One approach is to create for every single group of entities in OGC data model, a separate oneM2M container entity. Figure 7 shows a oneM2M data model that could represent an OGC/STA - IPE in a hosting CSE.

The top of a tree may be an <ae>. Below there are <containers> representing the dedicated parts of the ISO 19156 data model. They may be all at the same level and represent a flat data model representation.

There may be, for example, one <container> where all incoming <observations> are stored by the IPE. There may be another <container> where the <Datastream> object(s) for this <Observation> are stored. There may be also <containers> for <Locations>, <Sensors>, <Things> etc.

The OGC/STA -<Observation> itself may be represented as an <flexContainer>, under a <container> called “observations”.

The OGC/STA -<DataStream> itself may also represented as an <flexContainer> under a <container> called <Datastream>. In figure 7 the <flexContainer> are represented as grey boxes.

This way a oneM2M data structure represents all of the entities from the ISO 19156 data model.

The relationships between the ISO 19156 entities can be represented using dedicated custom attributes defined inside a <flexContainer> specialization.

Figure 7 shows the <flexContainer> with the resource name <rn> “ObservationXYZ” that has an attribute “<STAdatastream>” with the value “Datastream123”. This Attribute points to a <flexContainer> representing the <Datastream> object with the <resourceName> ”dataStream123” located in the DataStream <container>. This way the attribute represents inherent the relationships from the OGC/STA data model.

Actually, two kinds of relationships are described in this approach:

* The first kind are the oneM2M specific <ae>-to-<container>-to-<flexContainer> relationships (grey lines).
* The second kind of relationships are OGC specific and are expressed as attributes inside the <flexContainer> entities. Here for example a <Observation>-to-<Datastream> relationship is described (blue line).

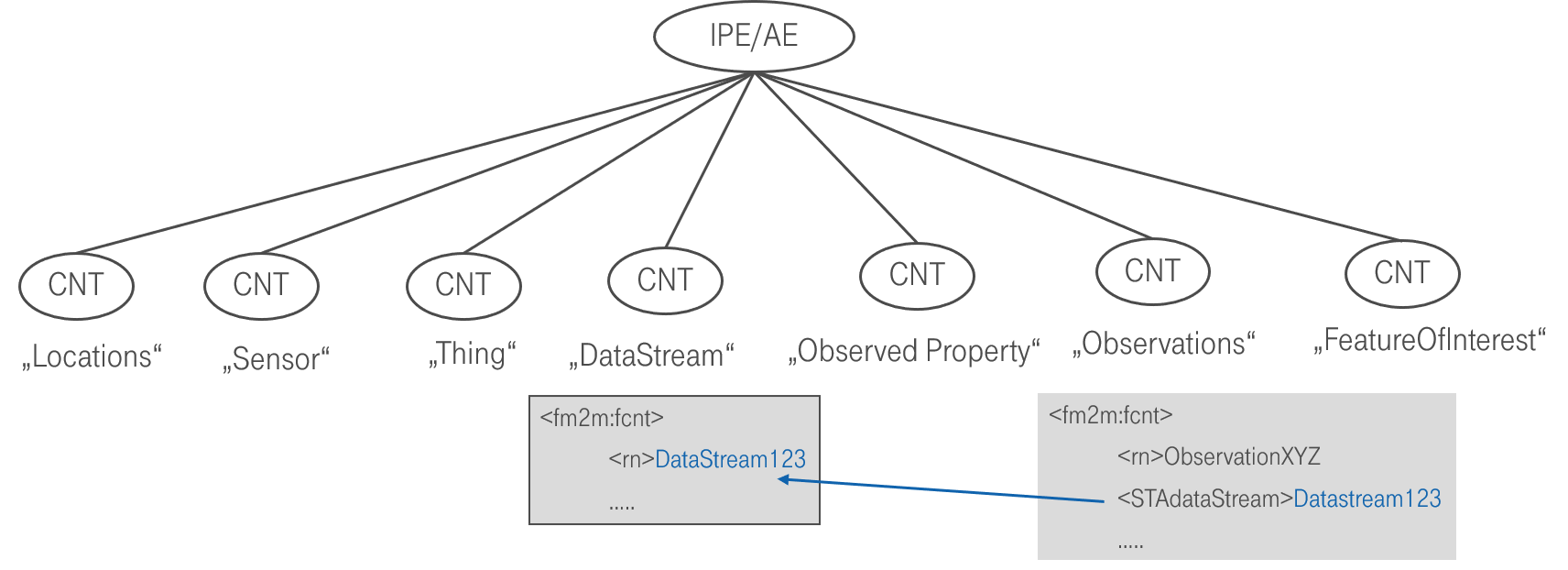


Figure 7: The flat data model with inherent connections

#### 6.2.1.1 Missing Relationship Management

The inherent relationships (blue line in figure 7) describing the OGC/STA data model can’t be managed by the CSE by oneM2M capabilities as of today. Only the relationships between <ae> and <container> can be managed (grey lines) by the CSE.

E.g in case a certain <flexContainer> representing a <Datastream> is deleted, all related <flexContainers> representing an <Observation> need to be deleted too. This is how it’s handled in an OGC/STA Server, but this wouldn’t be possible within oneM2M’s hosting CSE.

#### 6.2.1.2 Discussion of “Flat Data Model” Approach

This approach has advantages and disadvantages:

1. Although the data relationships can be described in the hosting CSE, they can’t be managed by the CSE. This may lead to inconsistent data and synchronization issues
2. This approach would be within the framework given in TS-0033 because there is a representation of a non-oneM2M Proximal IoT function

Note: This approach is rather a theoretical one and shows the issue of a missing oneM2M relationship management in the CSE.

### 6.2.2 “On-the-fly” Approach

The “On-the-fly” approach forwards every request from the oneM2M side to the IPE. The IPE translates the request “on-the-fly” towards OGC/STA and retrieves the response. The OGC response is translated towards oneM2M and goes through the CSE back to the requesting application. In the hosting CSE, a <remoteCSE> entity is used to forward the requests to the IPE.

Figure 8 shows the oneM2M-to-OGC/STA direction. A <contentInstance> is created at the hosting CSE pointing towards a <remoteCSE> endpoint. The request is forwarded to the remote CSE, in this case the IPE. The IPE constructs an <Observation> creation request and copies the ‘content’ attribute of the <contentInstance> to the ‘result’ attribute of the <Observation> and sends it to the OGC/STA server.



Figure 8: Gateway oneM2M-to-OGC/STA direction

Figure 9 shows the OGC/STA-to-oneM2M direction. OGC/STA does not define a publish / subscribe mechanism on HTTP protocol level. Hence the IPE has to subscribe to the MQTT-Broker of the OGC/STA server. The OGC/STA server publishes its new <Observation> via the MQTT broker. The IPE creates a <contentInstance> create request and copies the ‘result’ attribute of the <Observation> to the ‘content’ attribute of the <contentInstance>. Alternatively, a <flexContainer> may be created or updated. The <container> may be created beforehand at the hosting CSE where the IPE <contentInstance> or <flexContainer> resources are stored. All interested applications may subscribe to this container.



Figure 9: Gateway OGC-to-oneM2M direction

This approach is simple and sufficient in case its only about translating <Observation> to <contenInstances> and vice-versa. In case of other parts of the OGC data model like <Sensor> or <Location> required, additional requests from an interested application are needed.

#### 6.2.2.1 Optional Additional Data Requests

Every OGC/STA <Observation> has an ID. With this ID the OGC/STA server may be asked for further parts of the data model. This <Observation> ID needs to be copied from OGC/STA to the oneM2M representation. Based on this <Observation> ID a oneM2M application may request, through the IPE as a remote CSE, the remaining parts of OGC/STA data model.

This additional information may be encoded in the URL. In this case the IPE is able to retrieve the requested information. In some case, such as retrieving location information, the IPE would be required to perform several requests, before a response can be delivered to the CSE and a subscribed oneM2M application (see Figure 10).

Figure 10 shows the message flow where an application requests the <Location> of a received <Observation> with an ID “4711”. An oneM2M application might request a <contentInstance> on dedicated endpoints e.g. “../observation4711/location”. The request may be forwarded via <remoteCSE> mechanism and the IPE performs several requests towards the OGC/STA server, in order to get the requested <Location>. The OGC/STA <Location> may be translated into a dedicated attribute of an <flexContainer>.

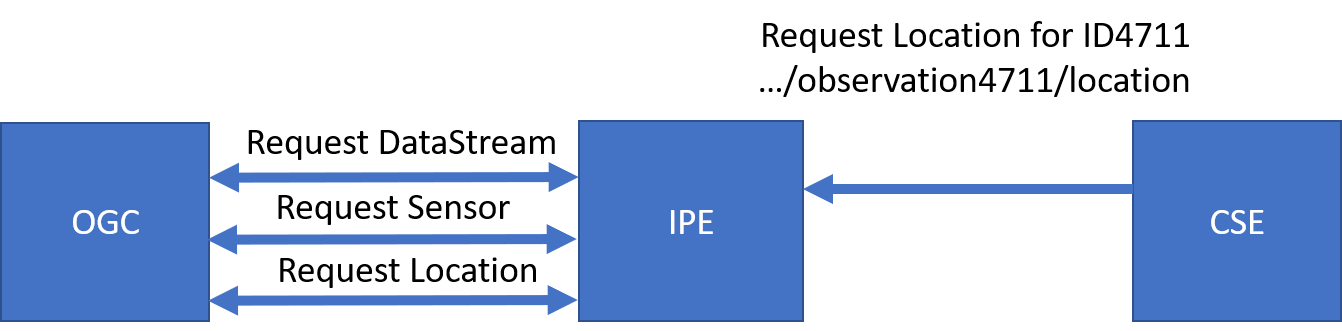


Figure 10: Requesting additional parts of the OGC data model

#### 6.2.2.2 Discussion of “On-the-fly” approach

The “On-the-fly” approach has a number of disadvantages:

1. This approach is outside the framework of TS-0033, because there is no representation of a non-oneM2M Proximal IoT function
2. The oneM2M client application requires insights about the structure of the OGC data model, in case it wants to request additional data to an <observation>
3. It needs a dedicated URL scheme for additional data requests. This schema has to be known by the application.
4. The observation ID has to be copied from the <Observation> to the oneM2M data structure for example a <flexContainer> in case the oneM2M client application wants to get additional data.

On the other hand, the approach has also advantages:

The IPE would not be required to copy the OGC/STA data model into the hosting CSE. The data would always accurate, because they stay in the authoritative source, the OGC/STA server. This approach avoids synchronization effort and possible errors.

Conclusion:

The “On-the-fly” approach would be a very flexible solution, in case only single value data need be exchanged. In case data get more complex, the oneM2M application requires knowledge about the OGC / STA data structure.

This approach is outside the framework of TS-0033.

### 6.2.3 “Specific Device” Approach

Another architectural approach would be to focus the design of the IPE on a specific device type.

The data model may be described according to TS-0023 Home Appliance Information Model. TS-0023 describes a templating tool for describing heterogenous devices and their functionalities using a Smart Device Template (SDT). SDT offers a generic and flexible modeling structure for non-oneM2M devices.

The first step in an OGC/STA interworking scenario may be to register the IPE to the hosting CSE as an <*ae*> resource. This <ae> resource is a parent for dedicated <*flexContainer*> resource specializations that represent each device connected to the OGC/STA server (for example an EV-Charging station).

OGC/STA devices may be modelled as SDT Devices. Mapping of the SDT Device model to oneM2M resources is performed according to the general mapping procedure described in clause 6.2.2 of TS-0023. A SDT Device component is mapped to a specialization of a <*flexContainer*> resource with an associated 'DeviceClass ID' (e.g. "org.onem2m.home.device.tv") *containerDefinition* attribute.

Figure 11 shows an example of a OGC/ STA device:*[* *[deviceElectricVehicleCharger]],* which is modelled as *a <flexContainer>* resource specialization derived from the corresponding SDT Device component. The model of *[[deviceElectricVehicleCharger]]* follows the schema described in clause 5.5.18 of TS-0023.

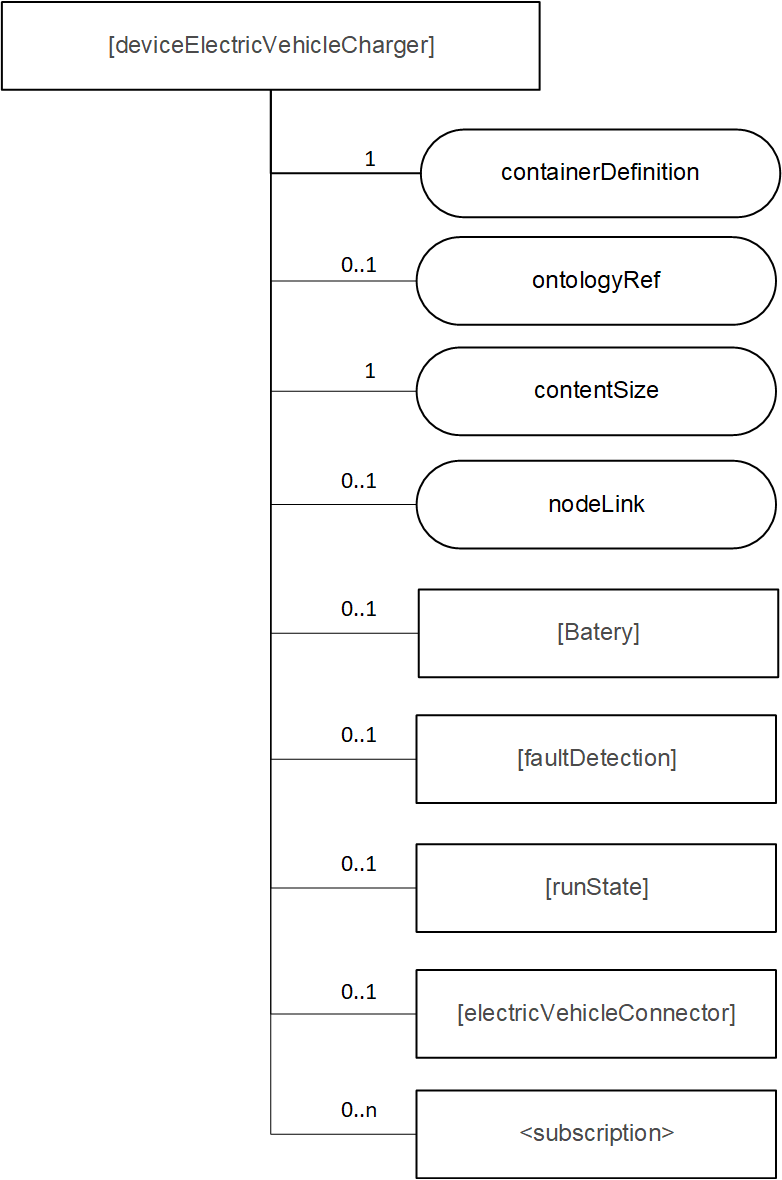


Figure 11: [deviceElectricVehicleCharger] example resource representing a OGC/STA device

The <flexContainer> representation based on SDTs allow the design of a desired data model for supported device types. The IPE is responsible for ensuring changes in the OGC data model are mapped to an update of the appropriate <flexContainer>.

#### 6.2.3.1. Communication Schema

In this approach the IPE subscribes to the MQTT message broker of the OGC/STA server, to receive all desired changes in the data model of a certain OGC/STA device. In this case, apart from <Observations> all changes, such as <Location>, are published to the IPE (Figure 12). The IPE may subscribe or filter out only changes affecting the <flexContainer> and sends respective UPDATE messages to the CSE.

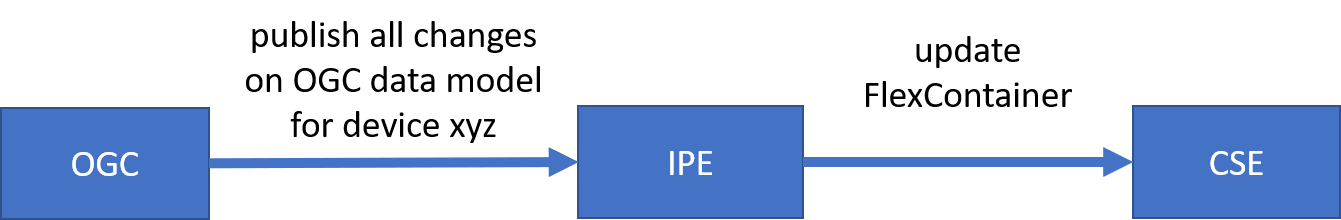


Figure 12: OGC/STA-to-oneM2M direction

The IPE subscribes to the <flexContainer> resources in the hosting CSE. If there are changes to the <flexContainer> from an application the CSE will send a <notification> message to the IPE. The IPE assigns the appropriate messages to update the OGC data model (Figure 13.)

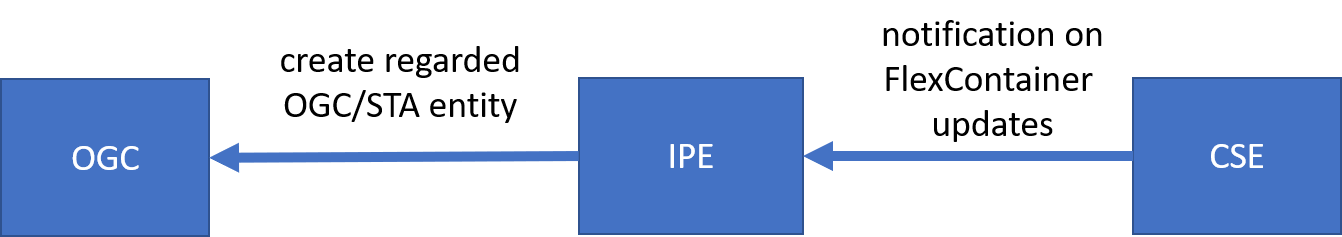


Figure 13: oneM2M-to-OGC/STA direction

#### 6.2.3.2 Discussion of the “Specific Device” Approach

The disadvantage of this approach would be inflexibility. The SDT data model and its mapping to the OGC model has to be designed beforehand, specifically for the device. Even if there are tools to create SDT <flexContainer> from a certain device automatically, the mapping to or from the OGC/STA data model may still be highly individual because the “properties” field in the OGC/STA data model can be filled with arbitrary data in JSON-Format.

In this case it would be beneficial to have somethings similar for SDTs, a data structure for arbitrary attributes. It is very likely that SDT not always describe all features, information and attributes of a complex device. To have something like a ‘property’ in SDT could ease the process of translating foreign data model into oneM2M. In our OGC / STA example use case “ev-charging” there is a foreign index like “chargingID” (see Figure 3) that could be defined as an attribute in a “property” structure. In SDT 3.0 “Properties” were already discussed as an addition to “Action”, “Datapoint” and “Event”. They should be used for non-functional data. An alternative approach for attributes that are not defined by a SDT is the “Label” attribute.

As a consequence, this approach enables no OGC/STA IPE for general use currently. It rather more enables a OGC / STA IPEs for specific devices e.g. “EV-Charging Stations” of “Company XYZ” in “Version 1.23”.

However, this approach is also beneficial. Compared to the approaches discussed before, in this case the client application does not need to have any knowledge about the OGC data model. A client application may only rely on oneM2M specifications and is still able to read data coming from a sensor that is connected via OGC / STA.

This approach is fully compliant with TS-0033.

### -----------------------End of change 1-------------------------------------------