Comparison of Description Languages for the Internet of Things

Abstract—**Number of IoT devices increased dramatically in the last years and it is now above the human population. This many connected devices created a lot of questions. How to interoperate, control or maintain different devices, how to get most from one device are the first questions. There are many IoT platforms, with different device description approaches to these problems. However, it is not clear which platform has advantages in describing the device or its functionality. Is there any platform that fits for all or is there a better solution in different cases? Describing is a key factor here in order to interoperate, control and maintain devices, it is a must to know devices capabilities in every aspect. In this work, we have compared the description models of popular platforms with a new standard of World Wide Web Consortium, Web of Things (WoT) Thing Description, and analyzed the expressiveness of the descriptions. After the comparison, it is found that, every platform have their advantages by defining IoT devices, in different scenarios.**

Index Terms—**Internet of Things(IoT),**

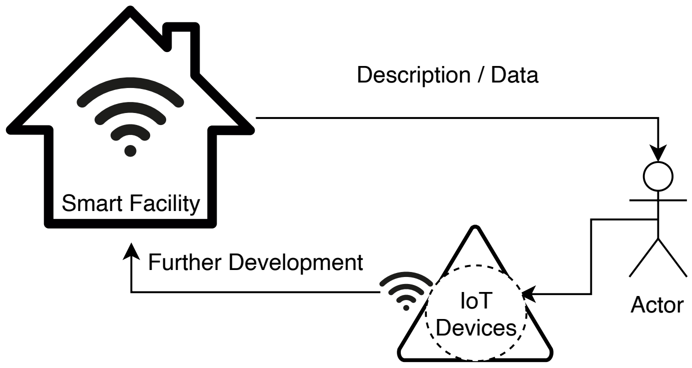

Fig. 1. Comparison Scenario. The developer has only data from device descriptions or data coming from devices. The task is to add more devices in this kind of a setup

I. Introduction

Despite the Internet of Things(IoT) being a relatively new concept, which is assumed it was born in between 2008-2009 [1], there are more than 8 billion of IoT devices [2]. There are numerous platforms available, including from big companies, trying to solve needs of this many devices. The needs and problems of IoT are evolving since the market is not settled yet. All the platforms in the market have different approaches to the problems, which generally differ from the target group of the platform, like industry, smart home, automotive, and the general needs of this group.

In this work, some of these popular platforms are compared. Main point of this comparison is the expressiveness of the descriptions of devices connected to the platforms. Pricing, scalability and other features are out of this works context. Only the features, which have influence on the descriptions, will be covered to explain their effect on description.

To fairly compare different platforms, they will all be compared in one scenario with the aim to understand how expressive they are for a person who did not use this platform before. To compare the expressiveness, all the descriptions will be compared with an emerging standard, the Web of Things (WoT) Thing Description (TD).

In section 1A the comparison scenario and in section 1B the comparison method, is detailed. In section 2 Thing Description and its properties are explained. AWS IoT, Azure IoT, Mozilla WebThing and Oracle IoT and their description methods are examined and compared with TD in subsections of section 3 respectively.

A. Comparison Scenario

All the comparisons will be made within one scenario. A developer is given the job to add new devices to an already existing setup within one platform. This developer does not have the code for the setup, only the data or descriptions of the existing devices. While adding new devices, it is required to conserve the existing system functionality and add some new functionality to the setup with the new devices. For example, a smart home owner, who has smart lights and temperature sensors, buys new controllers for the heating and cooling system of his/her house. The home owner wants from this developer to add the controller to system so that, he/she can control the temperature of the home from the Internet and also create scenarios like ”if the temperature is below a degree, turn on the heater”. Even though this example has only a few devices included, this example can also be thought in a bigger scale for a building, factory or a network of factories.

B. Comparison Method

For the comparison method, it will be assumed that a developer has already done a similar scenario for another system, in which all the devices had a TD. For the new application case, it will be questioned, which parts of the job is easier or harder to do, or cannot be done, compared to the case with a TD.

II. Base Scenario with Thing Description

In this section, we introduce the TD standard to be able to make further comparison. TD is a JSON-LD file, which includes four main components: textual data about the Thing itself, a set of Interaction Affordances that indicate how the Thing can be used, schemas for the data exchanged with the Thing for machine-understandability, and Web links to express any formal or informal relation to other Things or documents on the Web.

The Interaction Model of W3C WoT defines three types of Interaction Affordances with the following specific names: properties, actions and events, which can be seen in the Listing 1 in lines 9, 15, 20 respectively. Properties are used for getting the current value of sensing and controlling parameters. Actions are used for invocation of physical processes. Events are used for the push model of communication where notifications, discrete events, or streams of values are sent asynchronously to the receiver [3]. Thus, by having just a TD, developer can have information about device and how to get or set data, trigger actions, and receive events from the device. In our further comparisons, these three Interaction Affordances will be important as a reference. TD also has a field called `description` available for every single affordance, which gives ability to explain more about this affordance and it gives clues for a developer for further development.

```
 1  {
 2      "@context": "https://www.w3.org/2019/wot/td/
            v1",
 3      "id": "urn:dev:ops:32473-WoTLamp-1234",
 4      "title": "MyLampThing",
 5      "securityDefinitions": {
 6          "basic_sc": {"scheme": "basic", "in":"
                header"}
 7      },
 8      "security": ["basic_sc"],
 9      "properties": {
10          "status" : {
11              "type": "string",
12              "forms": [{"href": "https://mylamp.
                    example.com/status"}]
13          }
14      },
15      "actions": {
16          "toggle" : {
17              "forms": [{"href": "https://mylamp.
                    example.com/toggle"}]
18          }
19      },
20      "events":{
21          "overheating":{
22              "data": {"type": "string"},
23              "forms": [{
24                  "href": "https://mylamp.example.
                        com/oh",
25                  "subprotocol": "longpoll"
26              }]
27          }
28      }
29  }
```

Listing 1. An example TD from [1]which has 3 Interaction Affordances.

III. Comparisons

In this section the description methods and related features to this descriptions are explained for popular IoT platforms. After the explanation, the description systems are compared with TD to explain their advantages and disadvantages.

[1]https://www.w3.org/TR/wot-thing-description/

A. *Amazon Web Services IoT*

Amazon has a different approach than the TD, which is called Device Shadow Service (DSS). With DSS, Amazon puts its Web Services (AWS) as a middle man. Every request, which is sent directly to the device in the case of TD, is done to the AWS with DSS. Amazon has a copy of all properties on the cloud and when the user requests data, he/she gets the shadow of the device from the AWS. With this approach, device's workload is reduced and in the case of a sensor device, the device only needs to update the shadow on the cloud. The data requests from other devices or systems are answered from AWS, which has typically more resources than an IoT device. This way, it is assured that the device will not be overloaded by answering the requests and it can easily do the intended job. In the case of a failure of a device, the last valid data can be used from AWS/DSS, which assures the continuity of the system.

Device Shadows uses 3 commands to interact with the device, `get`, `update` and `delete`. All the three commands can be used both as RESTful API or Pub-Sub via MQTT [4]. The commands does not effect directly interaction affordances like the TD. Property update or get for DSS is straightforward and acts similarly with TD's Property Affordance. By updating a property, an action can be instantiated. And by using registering MQTT topics, user can be informed whenever a value is changed. So DSS is capable of representing functionality of 3 Interaction Affordances of TD.

DSS also gives some more information because it is MQTT based and MQTT is asynchronous in nature. It is not guaranteed that messages will be received or received synchronously. So it has a version field and a timestamp field for the document and timestamp for each field. Listing 2 shows an example return of a request. Version field is used for distinguishing the recentness of a request. If a request version is equal or below the version of shadow device, it is rejected, which prevents an old request to change the device state, but it also means that some requests might not be executed.

```
{
    "state": {
        "desired": {
            "lights": {"color": "RED"},
            "status": "ON"
        },
        "reported": {
            "lights": {"color": "GREEN"},
            "status": "ON"
        },
        "delta": {
            "lights": {"color": "RED"}
        }
    },
    "metadata": {
        "desired": {
            "lights": {
                "color": {
                    "timestamp": 123456
                }
            }
            "status": {
                "timestamp": 123456
            }
25            },
26            "reported": {...},
27            "delta": {...}
28        },
29        "version": 10,
30        "timestamp": 123456789
31    }
```

Listing 2. A reworked Shadow Device from [2]

Shadow Device also have 3 versions of same data field, which are desired, reported and delta (line 3, 7, 11 in Listing 2). Reported field shows the last values which is updated from device to AWS. Desired shows the values in which an update is requested but not instantiated yet (possible future values). Delta shows which value is about to change, (update request is received by physical device and change process is began).

The description advantage of DSS against TD is the extra default fields timestamp, desired and delta. `timestamp` gives information about when was the last change of field. With the help of `desired` and `delta` fields, Device Shadow does not only gives current value information but also information about process and possible future. A downside is the not separated affordances unlike TD, the purpose of fields are not clear, another documentation is needed to explain if a field is used for a action or if it is read only for a server. And because the shadow things main purpose is not describe the thing, it lacks any description field which may help on later development.

B. *Azure IoT*

Microsoft has its own IoT solution with its Web service Azure. Azure IoT's approach is built on the simulated devices. Devices can be simulated with JavaScript files, where every possible failure and a new device integration can be tested without physically connecting the device and precautions for extreme cases can be handled beforehand. Its solution for description is called Device Model Scheme (DMS).

DMS has 3 main fields for interactions: `Properties`, `Telemetry`, `CloudToDeviceMethods` (Line 9-14-27 in Listing 3). It also has one field for `Simulation` which has simulation files to simulate output values for `Telemetry` objects, other fields which describes devices name description and communication protocol. [5]

In Azure IoT `Properties` has a different meaning than TD. They are static fields which describes devices location and type, which will be used to sort devices in the user interface.

`Telemetry` is a similar solution to properties in TD. The main difference is Azure assumes these values will be updated regularly, so it has an `Interval` to give information about its update interval. In Telemetry, data fields can be bundled and the output template is given in `MessageTemplate`, to make the values also code readable.

`CloudToDeviceMethods` is similar to Actions in TD. Rather than giving input fields or a form to reach, it gives a script file which can be called from the IoT Hub. These methods can be called by an HTTPS POST from the cloud through the hub with payload.

Azure IoT also supports Device Twin, which is similar with Device Shadow of AWS IoT without the `Delta` field. It is specifically generated and used for the long running tasks of the backend or the device.

Direct simulation is not describable by default in TD, which can be useful in development process, but it does not give any extra information about the description. DMS has `Interval` in `Telemetry` field, which is an advantage for DMS and which TD does not have. Message template is a field missing in TD but TD has a output field which gives all output keys and value types. There is no difference if output is a JSON object. DMS has description field only for the device, not for telemetry or methods, which is a disadvantage for developers who use the system later on. Method field having script files may be useful for updating the functionality by modifying the script files but it does not give enough information about inputs and outputs. Lastly, there is no events field, which could be implemented through telemetry fields on the server, but it can not be understood without detailed extra documentation or testing.

```
1   {
2       "SchemaVersion": "1.0.0",
3       "Id": "drone",
4       "Version": "0.0.1",
5       "Name": "Drone",
6       "Description": "Simple drone.",
7       "Protocol": "AMQP",
8       "Simulation": {...},
9       "Properties": {
10          "Type": "Drone",
11          "Firmware": "1.0",
12          "Model": "P-96"
13      },
14      "Telemetry": [{
15          "Interval": "00:00:05",
16          "MessageTemplate": "{"velocity":"${
                velocity}","acceleration":"${
                acceleration}","position":"${
                latitude}|${longitude}|${altitude
                }"}",
17          "MessageSchema": {
18              "Name": "drone-event-sensor;v1",
19              "Format": "JSON",
20              "Fields": {
21                  "velocity": "double",
22                  "velocity_unit": "text",
23                  ...
24              }
25          }}],
26      "CloudToDeviceMethods": {
27          "RecallDrone": {
28              "Type": "JavaScript",
29             "Path": "droneRecall-method.js"
30          }
31      }
32  }
```

Listing 3. An example DMS from [3], with Method and Telemetry data

C. *Mozilla WebThing*

Mozilla WebThing is an open source implementation of Web of Things standards at the W3C[6]. The difference

[2]https://docs.aws.amazon.com/en_us/iot/latest/developerguide/using-device-shadows.html

[3]https://docs.microsoft.com/en-us/azure/iot-accelerators/iot-accelerators-device-simulation-advanced-device

is, their approach is built upon a gateway device. To use Mozilla WebThing, all devices should be connected to a gateway device. Every request made from the client is made to gateway and gateway communicates with the device, so direct interaction with the device is not possible. For setups with few devices, adding an extra gateway device may increase complexity of the setup compared to TD.

WebThing supports HTTP and WebSocket protocols. In some cases, having a continuous connection with the gateway may be useful with web socket. With this connection it can be guaranteed that the gateway is alive and responding and message will be delivered if it does not break while sending.

Even though, Mozilla claims that, they are an implementation of WoT, few keywords are different than TD. For example WebThing uses links, rather than forms. Because security measures are covered in gateway, WebThing does not give any information about security. One other problem is they are only using RESTful API, which means that it is not possible to get async events.

In WebThing, to prevent any possible timeout for action request, response is always same as the request with http code 201 (created). Chaining actions through result of action requests are not possible, because actions does not give any valuable information in response like TD. To follow the action it is needed to make another request to get action queue, which is a feature of gateway which logs every action and their status. Because actions returns the same response type, and property request returns the property object with filled values, WebThing does not use output field. In TD responses may vary and it can be describable in the output field.

As a result, it can be said that Mozilla WebThing is a subset of TD rather than an implementation. It only considers a use case with a gateway, with HTTP and WS protocols. It does not allow to connect to the device directly. Also some fields are missing, because it is necessary to connect every device to a gateway, with WebThing and the job of those fields are covered in gateway.

D. Oracle Device Model (ODM)

Oracle also has its own solution for IoT in business cases. It has a wide variety of programming languages and request protocols to create IoT enabled devices integrated with their cloud service [7]. To create a Device Model, the Oracle Internet of Things Cloud Service Management Console provides step by step fields. Main fields of ODM are `Custom Attributes`, `Actions` and `Alerts and Custom Messages`.

`Custom Attributes` is similar to TD's properties, which also includes `Writable` and `Description` fields. It allows for developers to know read-only properties and understanding the aim of the property. Both fields exists also in TD. `Writable` field is described as `readOnly` or `writeOnly` in TD. `Actions` are same with TD and `Alerts and Custom Messages` is similar to `Events` field in TD[8].

ODM has `description` field for every field possible, which makes the aim of fields more understandable for developer. It has all three Affordances that TD has, which makes ODM really similar to the TD. The similarities makes it easier to convert TD to ODM and vice versa. Code for such a conversion is publicly available [4]. Even though using gateways are possible with Oracle, it is not forced. So direct access to device is possible, and having a direct library support even for mobile operating systems is an advantage for ODM.

IV. Comparison Results

In Section 3, four popular platforms description models are compared to TD. In this section the comparisons are summarized. The summarized information can also be seen in Table I.

Our first point was the interaction models and 3 Interaction Affordances of TD was our base. Oracle and Mozilla are able to describe all 3 affordances, with separate and understandable fields. Azure IoT has only similar fields for Action and Properties of TD but it lacks Events as a field, but the same usage can be done with listening to the changes of Telemetry field. In this particular comparison AWS IoT was worst performing with only one field for all. Even the usage can be done, it needs to be documented well.

Second comparison was over the description field. Mozilla and Oracle also has description field for all affordances as in TD, while Azure only has a description for device and AWS has none.

In TD, it is possible to describe a field is read-only, for example measurement of a sensor, which is also possible for Oracle and Mozilla. In Azure or AWS it is not possible. AWS also allows to trigger actions by setting a property and not knowing if this property is allowed to set or if it initializes any action creates confusion for further development.

AWS and Azure is built on an async nature, which gives some advantages to those two platforms in terms of descriptiveness. AWS has 3 fields for the same property, which explains, the current status, what is about to change and what will be the last result. In Azure, it has 2 fields, which describes now and end result. TD and the other 2 platforms do not have this feature. Knowing future values also prevents other devices to send the same ending request over and over again in the changing process, which is especially advantageous for processes taking long time. The same feature gives also timestamp and versions to control the order, which is adds extra information about the device. However this async nature may also cause some processes to be not initiated because another request is processed earlier.

Azure IoT assumes data properties are updated in an interval, so it has a field called interval by default, which is useful for sensing devices and other models do not have it by default. Azure also supports simulation in its description, which is useful for testing a system without having a physical device. It may be also useful, if simulation files are available in further development stage to see the usual working of the device and test the extreme cases.

[4]https://github.com/w3c/wot/tree/master/plugfest/2019-tpac-fukuoka/tools/Oracle

One other difference between those platforms is how they access to the devices. There are 3 main type of access, with middle man (a service between devices and the client), gateway in the middle and direct access. All have different advantages. Web services buffers the data from the device and serves it to the requests, which reduces the requests that devices answers and its workload, which is especially important for big systems with lots of requests. Gateway devices may seem an extra devices, which adds up more complication. But because the Internet access is done from this device only, the other devices in a system do not necessarily need to be internet capable, any connection to gateway is enough like USB or BlueTooth. It allows to build a system with less capable, cheaper devices. Direct access may be important for time important or synchronous applications. And because it does not need a server connection or an extra gateway, it is easier to create a simple system.

TABLE I
PLATFORM COMPARISON

Feature	***AWS***	***Azure***	***Mozilla***	***Oracle***
Interaction Affordances	Only 1	Only 2	All 3	All 3
Description Field	None	Only Device	Everywhere	Everywhere
Read only fields	N.A.	N.A.	Available	Available
Future Information	Near Future, End Result	End Result	N.A.	N.A.
Device Access Type	Web Service	Web Service	Gateway	Web Service or Gateway

V. CONCLUSION

All 5 description models, has their own advantages for different application types. The differences comes from their target group, and assumptions of usages, which also effects the device access type.

The most descriptive models for further development are TD, Oracle ODM and Mozilla WebThing, because they have description fields for every field. For applications with many devices, AWS IoT and Azure IoT has useful extra description features, such as simulation, interval and future values.

REFERENCES

[1] D. Evans, "The Internet of Things How the Next Evolution of the Internet Is Changing Everything", CISCO, Tech. Rep., 04 2011. [Online]. Available: https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf
[2] "Market inside for internet of things", 2018. [Online]. Available: https://iot-analytics.com/wp/wp-content/uploads/2018/08/State-of-the-IoT-Update-2018-Q1Q2-August-2018-SAMPLE.pdf
[3] "Web of things (wot) thing description". [Online]. Available: https://www.w3.org/TR/wot-thing-description/
[4] "Device shadow service for aws iot - aws iot", 2019. [Online]. Available: https://docs.aws.amazon.com/en_us/iot/latest/developerguide/iot-device-shadows.html
[5] Dominicbetts, "Device schema in remote monitoring solution - azure". [Online]. Available: https://docs.microsoft.com/en-us/azure/iot-accelerators/iot-accelerators-remote-monitoring-device-schema
[6] "Mozilla webthings". [Online]. Available: https://iot.mozilla.org/docs
[7] "Oracle internet of things cloud service - tutorials", Aug 2019. [Online]. Available: https://docs.oracle.com/en/cloud/paas/iot-cloud/tutorials.html
[8] "Developing applications with oracle internet of things cloud service", Aug 2019. [Online]. Available: https://docs.oracle.com/en/cloud/paas/iot-cloud/iotgs/create-new-device-model.html