Shaping Device Descriptions to Achieve IoT Semantic Interoperability

Aparna Saisree Thuluva\textsuperscript{1,2}, Darko Anićić\textsuperscript{1}, Sebastian Rudolph\textsuperscript{2}

\textsuperscript{1}Siemens AG - Corporate Technology, \textsuperscript{2}TU Dresden, Germany.

aparna.thuluva@siemens.com, darko.anicic@siemens.com, sebastian.rudolph@tu-dresden.de

Abstract. The Internet of Things (IoT) promises easy integration of connected physical devices at a large scale. For this purpose use of semantic technologies are widely acknowledged, as they enable devices to understand the meaning of data instead of merely exchanging it. W3C WoT Working Group is standardizing Thing Description (TD) as a machine-readable interface of a thing. Further on, iot.schema.org specifies semantics for capabilities of things. Using these models it is possible to describe devices. However, they do not constrain semantics of IoT devices. For example, iot.schema.org provides one capability for a class of devices. Often same-class devices, produced by different manufacturers, differ in certain feature or data they offer. In order to represent device variants we propose to extend iot.schema.org Capabilities with RDF Shapes. In our approach, TDs for device variants are automatically generated. It also enhances thing discovery, semantic interoperability, validation of TDs, and accelerates IoT application development.

1 Introduction

The Internet of Things (IoT) promises techniques which will enable integration of connected physical devices with less effort, and as a consequence this will enable new added-value services at a large scale. The integration of two or more physical devices should not only happen at the level of the data exchange but also understand the meaning of exchanged data. Instead of mere interoperability, the IoT devices are required to achieve semantic interoperability. Only then, it would be possible for machines to discover devices relevant for an added-value service, and further to understand their capabilities, to integrate and process their data, and to ultimately create a new value.

The promise of IoT has not been delivered yet. The current focus in the IoT is merely to connect heterogeneous devices to a common IoT platform. Once connected, devices are capable to exchange data. But semantics of data and capabilities of devices, which generate this data, are not described in a machine-interpretable form. What is generated by each IoT platform is essentially a data silo that yet needs to be integrated. This situation hinders the potential of IoT to enable added-value services at large scale.
To improve this, W3C Web of Things (WoT) Working Group\(^1\) is developing a new standard which addresses the challenge of heterogeneous IoT platforms, communication protocols, and data formats. Among others, WoT Group proposes the so-called protocol binding to bring disparate IoT platforms and devices to the Web as a common application layer. Once connected to the Web, devices can be accessed and programmed with a standardized Application Programming Interface (API). However, the most important aspect of the upcoming standard is provision of Thing Description\(^2\) (TD). TD is a machine-readable description of an IoT thing (e.g., a physical device or virtual thing residing in a cloud). It abstracts an IoT thing in terms of its interactions and data it consumes or produces. Interactions, data and the thing itself may be marked-up with semantic terms from an ontology or a schema. iot.schema.org is a community effort\(^3\) which is developing exactly one such IoT schema to be used with TD. iot.schema.org abstracts the functionality of IoT devices via so-called Capabilities. Capability is a semantic model to formally describe traits of a class of physical devices (e.g., thermostat provides temperature data and it can be set to a target temperature). The schema is developed as an IoT extension to the widely-used schema.org.

W3C TD and Capabilities from iot.schema.org offer a solid basis for achieving the IoT semantic interoperability. For example, consider an air-condition whose functionality should be semantically described and exposed over an IoT API. For this purpose one can use Aircondition Capability\(^4\) to mark-up a TD of the device. TD is then used to generate a code-skeleton of the thing. This can be done, for example, with the W3C WoT reference implementation\(^5\). Once the code-skeleton has been implemented for the air-conditioner, the device can be exposed as an IoT thing, i.e., accessible over a Web API and discoverable thanks to its TD. Such an IoT thing can also be easily integrated with other things since its capability, interactions, and data are all semantically described and machine interpretable. According to iot.schema.org Aircondition Capability, the semantic description include interactions (e.g., *Temperature*, *TargetTemperature*, *RunMode*, *WindStrength* etc.), domain-related information (e.g., smart building, smart home, mobility, industry etc.), input and output data of each interaction, their data types, units etc.

With W3C TD and iot.schema.org Capabilities in place we are already in place to create semantically described IoT things and to realize them in an efficient manner. This is unfortunately not so easy in practice. For example, iot.schema.org always provides one Capability per class of devices. It implies that all air-conditioners in the world should be described with the same Capability. In the real world, air-conditioners from different manufactures differ, e.g., in the temperature range they operate in or in the running modes they support.

---

\(^1\) [https://www.w3.org/WoT/WG/](https://www.w3.org/WoT/WG/)

\(^2\) [https://www.w3.org/TR/2017/WD-wot-thing-description-20170914/#type-system-object](https://www.w3.org/TR/2017/WD-wot-thing-description-20170914/#type-system-object)

\(^3\) [https://github.com/iot-schema-collab/iotschema](https://github.com/iot-schema-collab/iotschema)

\(^4\) [https://github.com/iot-schema-collab/iotschema/blob/master/Aircondition.jsonld](https://github.com/iot-schema-collab/iotschema/blob/master/Aircondition.jsonld)

\(^5\) [https://github.com/thingweb/node-wot](https://github.com/thingweb/node-wot)
The goal of this work is to extend semantics of iot.schema.org Capabilities in such a way that it enables specification of variants of Capabilities, and automate the generation of TD for so-extended device Capabilities. In particular, we propose to use Shape Expressions (ShEx) [1] to specify constraints on IoT Capabilities. Shapes help to precisely define complex data of Capabilities and also to model various device variants. Then, TD templates can be generated from a Capability and its shapes. This is a very important extension to iot.schema.org Capabilities and W3C TD as it ensures rich semantic descriptions for IoT device variants, while keeping the overall process easy, e.g., with respect to automatically-generated TD, TD validation, and automatically-generated user interfaces for IoT things.

The main contributions of this paper are the following: (1) we describe how to specify ShEx constraints on IoT Capabilities in order to precisely define complex data of interactions and to model several device variants; (2) as Thing Description uses JSON Schema [2] to model input and output data of interactions, we create mapping between ShEx and JSON Schema; (3) we provide a prototypical implementation to generate semantically enriched Thing Description templates from an IoT Capability and its shapes.

The remainder of this paper is structured as follows. Section 2 discusses the background of our approach and outlines related work on W3C WoT, iot.schema.org and ShEx. In Section 3, we describe in detail about the modeling of an IoT Capability and describe the procedure to specify ShEx constraints on Capabilities. Section 4 presents the mapping between ShEx and JSON Schema. In Section 5 we present the prototypical implementation of the tool to generate semantically annotated Thing Description templates from an IoT Capability and its shapes. In Section 6 we provide a discussion about pros and cons of our approach. We conclude in Section 7 and present ideas for future work.

2 Background & Related Work

W3C Web of Things Working Group is developing a standard to create interoperability between physical things on the Web. For this, WoT Group is developing a protocol binding to enable interoperability between various protocols such as OCF, OPC-UA [3], BacNet [4] and so on. Apart from the protocol binding, WoT Group also proposes Thing Description, which is a platform independent abstract description of a physical device. Thing Description describes a device in terms of its interactions such as Properties, Events and Actions. A Thing Description is serialized in JSON-LD [5] format. Further on, Thing Description uses JSON Schema [2] to model syntactical constraints on data. JSON Schema provides simple data types such as integer, number, string, boolean and complex data types such as object, array and enumeration.

In order to provide semantic discovery and interoperability between devices, Thing Description is supposed to be extended with external (domain specific) ontologies and schemas [6, 7]. Many ontologies are developed for IoT, Domain-
independent ontologies such as Semantic Sensor Networks (SSN) [8], QUDT\(^6\) for Quantities and Units of measurement, IoT Ontology [9] and so on. Domain ontologies such as SAREF\(^7\) for smart building domain, eCl@ss [10] for industry domain and so on. More recently, the community work on iot.schema.org has started. It models a physical device in terms of its capabilities which are represented as TD interactions (such as Properties, Events and Actions). iot.schema.org provides RDFS semantics and reuses definitions from existing semantic models such as SSN, QUDT etc. In order to model domain features of a physical device, iot.schema.org normalizes semantics from existing standards such as OneM2M\(^8\), OpenT2T\(^9\), OCF\(^10\), IPSO Objects\(^11\).

**RDF Shape Languages** defines constraints on RDF graphs, which can be used for purposes such as generating and validating data, or generating user interfaces [11]. There are two RDF Data shape languages: SHACL\(^12\) and ShEx\(^13\). Both SHACL and ShEx can be used to express constraints on RDF graphs. JSON Schema can be used to model constraints on JSON data and to validate it. But JSON Schema cannot be used to validate RDF graphs. In Thing Description we have the need to validate both, the terms specified in RDF graphs and data specified with JSON Schema. In our approach we use RDF Shapes for both purposes. ShEx has several serialization formats such as ShExC, which is compact representation of ShEx Shapes. ShExJ\(^14\) is a JSON syntax, which serves as abstract syntax. And ShExR is a RDF syntax generated from ShExJ. In the remaining part of this paper, we use ShEx to model constraints on IoT Capabilities. ShEx is chosen as it provides intuitive and compact syntax. Moreover, it is tailored to be used with JSON-LD, which is the serialization format of Thing Description.

### 3 ShEx Constraints Modeling on an IoT Capability

We use ShEx as a toolkit to model constraints on IoT Capabilities. An IoT Capability represents capabilities of a class of physical devices. However, in the real-world there exists many variants in a class of physical devices. These variants can be expressed by modeling shape constraints on an IoT Capability. These shapes can be used for multiple purposes such as the following: (1) a shape models different variants in a class of physical device;, (2) a shape enhances interoperability by acting as a shared model and documentation of a device variant, which helps to exchange and understand device descriptions between

---

\(^6\) http://qudt.org/1.1/schema/qudt#

\(^7\) https://w3id.org/saref

\(^8\) http://www.onem2m.org/

\(^9\) https://github.com/openT2T/translators

\(^10\) https://oneiota.org/

\(^11\) https://github.com/IPSO-Alliance/pub/tree/master/reg

\(^12\) https://www.w3.org/TR/shacl/

\(^13\) https://www.w3.org/2001/sw/wiki/ShEx

\(^14\) https://shexspec.github.io/spec/
multiple parties; (3) a shape can be used to generate semantically enriched TD templates from an IoT Capability shape.

In this section, we will consider the air-conditioner example presented in section 1 and present an IoT Capability for an air-conditioner. After that, we will describe a procedure to model ShEx constraints on an IoT Capability and show how different variants of a physical device (eg., an air-conditioner) can be created by simply changing few constraints in a shape.

3.1 IoT Capability

iot.schema.org Capability model is alligned with W3C WoT model presented in [7], which models a physical device in terms of its Interaction patterns such as Properties, Events and Actions. Therefore, an IoT Capability provides a semantic description of a physical device in terms of its Interaction patterns and their input and output data. Here we present an IoT Capability defined for an air-conditioner. Due to space constraints, we present a part of the Capability in Listing 1.1. The complete specification of the Aircondition Capability, its Interaction patterns and data can be found online. This Capability is created by normalizing the Aircondition descriptions from OneM2M, OpenT2T, OCF. Among others, the Aircondition Capability provides the following interaction patterns (1) Temperature, which indicates the current temperature on the air-conditioner, (2) TargetTemperature, which is used to set temperature on an air-conditioner, (3) RunMode, which specifies operating modes of an air-conditioner. These interaction patterns are further described in terms of its input and/or output data. The example shown in Listing 1.1 presents the input and/or output data of each interaction patterns of the Aircondition Capability. The IoT Capability model further defines the data of an Interaction pattern in detail as presented in the next section.

Data Description The Data of an IoT Capability’s Interaction pattern is well-defined in terms of the value type, units of measurement, minimum and maximum range of the data. Lines 40-55 in Listing 1.1 shows the definition of data for each interaction pattern of the Aircondition Capability. Let us consider TemperatureData in this example. The definition states that an air-conditioner provides temperature data, whose value is of type schema:Float. It further defines that, there exists minimum and maximum range for temperature on an air-conditioner. Further on, it defines that temperature on an air-conditioner can be measured in TemperatureUnit, which is either iot:Celsius, Kelvin or Fahrenheit. Similar, RunModeData defines a set of pre-defined modes of operation of an air-conditioner. Data description for all interaction patterns of the Aircondition Capability can be found online.

---

15 https://github.com/iot-schema-collab/iotschema
16 https://github.com/iot-schema-collab/iotschema/blob/master/unit.jsonld
17 https://github.com/iot-schema-collab/iotschema/blob/master/interaction-patterns.jsonld
We have seen that an IoT Capability provides a well-defined semantic description for a class of physical devices. In the next section, we will describe a procedure to model constraints on an IoT Capability, in order to precisely define the data and, to model various device variants.

3.2 Modeling of ShEx Constraints

An IoT Capability itself is not sufficient to precisely model the definitions for multiple device variants. This can be achieved by modeling Shape constraints on a Capability. That is, RDF Shapes can describe multiple sub-graphs on an IoT Capability to represent variants of a device.

Shape constraints on an IoT Capability are specified at two levels: (1) on the level of data of an Interaction pattern. It precisely defines the complex data of an Interaction pattern, (2) on the level of a Capability in order to constrain the interaction patterns in a device variant. Listing 1.2 shows a part of ShEx constraints modeled on the Aircondition Capability in ShExC format (complete shape can be found online).

Constraints on Complex Data In this section, we will show the procedure for modeling constraints on data of an Interaction pattern. We will consider TemperatureData of Aircondition Capability shown in Listing 1.1 for this purpose. A set of ShEx constraints on the Aircondition Capability are shown in Listing 1.2. The <TemperatureData> shape defines constraints on TemperatureData. The shape expresses that, minimum range of TemperatureData should be 5.0 and maximum range should be 40.0. Further on, it also specifies that a physical device, which implements this shape should provide TemperatureData in Celsius unit. Similarly, <RunModeData> shape in Listing 1.2 models constraints on RunModeData. It expresses that an air-conditioner which conforms to this shape should provide CoolMode and/or EnergyOrPowerSavingMode operating modes. In this manner, many shapes can be modeled on an IoT Capability, each shape can acts as a specification of a variant in a class of physical devices.

Constraints on Interaction patterns In this section, we consider the Aircondition Capability to show modeling of constraints on Interaction patterns using ShEx. The Aircondition Capability shown in Listing 1.1 defines that a physical device of type air-conditioner should implement all six interaction patterns as specified in the Capability. However, in the real world, all the air-conditioners need not implement all these six functionalities. Therefore, using shapes we can specify several variants of air-conditioners by constraining the Interaction patterns an air-conditioner should implement. Consider the example shapes shown in Listing 1.2. In the example <AirConditioner> shape describes the constraints on interaction patterns of the Aircondition Capability. It means that a physical

18 https://github.com/aparnasai/iotschema/tree/iotschema-TDGenerator/Shape%20Expressions
device, which implements Aircondition Capability and this shape should implement BinarySwitch, RunMode, TargetTemperature and Temperature interaction patterns. Therefore, it represents a variation of air-conditioners. In the similar manner, another shape with a different set of interaction patterns can be defined on the Aircondition Capability representing another variation of air-conditioners. Further on, such a shape created for a Capability can be shared and it acts as a documentation representing a device variant.

In this section we have shown that it is necessary to specify constraints on IoT Capabilities, interaction patterns and Data. We also presented how to create Shape constraints to precisely define and constrain complex data and Interaction patterns of an IoT Capability.

4 Mappings between ShEx and JSON Schema

We use ShEx constraints specified on an IoT Capability to generate TD templates. A TD is serialized in JSON-LD format. Moreover, JSON Schema is used to model the data of Interaction patterns in a TD. Therefore, when the templates are generated from Shapes, the constraints on data of Interaction patterns should be converted to JSON Schema. For this purpose, we have created a mapping from ShEx to JSON Schema. In this section, we will present this mapping.

ShEx defines RDF Shapes as Node constraints, Triple constraints and Shape expressions. Here, we will present different elements of ShEx and a possible conversion to JSON Schema. The table in Figure 1 shows this mapping. In the table all the examples use default prefix ":" for URIs. This default prefix can be replaced by other prefixes depending on the use case. ShEx uses XML schema datatypes. In this examples we use \texttt{xsd} prefix for XML datatypes. Firstly, the table presents elements of a Node constraint such as \texttt{xsFacet} (StringFacet, NumericFacet) and \texttt{ValueSet} and their conversion to JSON Schema. Inside each facet we also present the constraints that are applicable to it. For example, constraints such as \texttt{MinInclusive}, \texttt{MaxInclusive}, \texttt{MinExclusive}, \texttt{MaxExclusive}, \texttt{TotalDigits}, \texttt{FractionDigits} are presented in \texttt{NumericFacet} as they are applicable only to numeric RDF literals. In some cases, there is no JSON Schema equivalent for ShEx. \texttt{TotalDigits}, \texttt{FractionDigits} in \texttt{NumericFacet} and \texttt{flags} in \texttt{StringFacet} are such examples. The ShEx Shape Expression \texttt{And} can be converted to a JSON Schema \texttt{allOf} if the shape is combining more than one shapes as shown in row 6(a) of Figure 1. Alternatively, \texttt{AND} can be converted to a JSON Schema \texttt{object}, if it combines more than one Triple constraints as shown in row 6(b) of Figure 1. Triple expression \texttt{EachOf} is also converted to JSON Schema following same approach as \texttt{And} shape.

This mapping from ShEx to JSON Schema is used as a part of TD template generator. It generates input and output data of a TD interaction pattern from an IoT Capability shape.
<table>
<thead>
<tr>
<th>Concept</th>
<th>ShEx</th>
<th>JSON Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. xsFacet: stringFacet</td>
<td><code>valueType : xs:string</code> MinLength X MaxLength Y pattern &quot;regex&quot; (xs:integer</td>
<td>`valueType : { &quot;type&quot; : &quot;string&quot; } minLength: X, maxLength: Y, pattern: &quot;regex&quot; )</td>
</tr>
<tr>
<td>String Length:</td>
<td>(valueType xs:string Length X )</td>
<td>String Length: `valueType : { &quot;type&quot; : &quot;string&quot; } minLength: X, maxLength: Y )</td>
</tr>
<tr>
<td>2. xsFacet: Numeric Facet</td>
<td><code>valueType : xs:integer</code> MinInclusive X MaxInclusive Y FractionDigits M</td>
<td>`valueType : { &quot;type&quot; : &quot;integer&quot; } minInclusive: X, maxInclusive: Y, fractionDigits: M )</td>
</tr>
<tr>
<td>MinInclusive &amp; MaxInclusive:</td>
<td></td>
<td>`valueType : { &quot;type&quot; : &quot;integer&quot; } minExclusive: X, maxExclusive: Y )</td>
</tr>
<tr>
<td>MinExclusive &amp; MaxExclusive:</td>
<td></td>
<td>`valueType : { &quot;type&quot; : &quot;integer&quot; } minExclusive: X, maxExclusive: Y )</td>
</tr>
<tr>
<td>3. Boolean</td>
<td><code>booleanLiteral : &quot;true&quot;</code></td>
<td>`booleanLiteral : { &quot;type&quot; : &quot;boolean&quot; }</td>
</tr>
<tr>
<td>4. Value list</td>
<td><code>value : [ &quot;X&quot;, &quot;Y&quot; ]</code></td>
<td><code>value : [ &quot;X&quot;, &quot;Y&quot; ]</code></td>
</tr>
<tr>
<td>5. Cardinality</td>
<td></td>
<td><code>value : [ &quot;X&quot;, &quot;Y&quot; ]</code></td>
</tr>
</tbody>
</table>

**Fig. 1:** Mapping between ShEx Schema and JSON Schema
5 From IoT Capability Shape to Thing Description Template

We have developed a prototypical implementation of TD template generator which has two parts: (1) it generates semantic marked-up TD Interaction templates from an IoT Capability shape, (2) generates input and output data for Interaction patterns using the mappings presented in Section 4. The prototype has been developed in javascript and it is available online.\textsuperscript{19} The mappings are done at two levels. (1) mapping from IoT Capability model to WoT TD model. This is a direct mapping as IoT Capability model follows WoT TD model. This mapping generates semantically enriched Interaction templates of a TD, (2) mapping from ShEx to JSON Schema. This mapping generates input and output data of an Interaction Pattern.

The tool is built on top of shex.js, which is a javascript implementation of ShEx.\textsuperscript{20} A Shape of an IoT Capability in ShEx compact format (ShExC) is given as input to shex.js which verifies the shape and converts it into ShEx JSON format (ShExJ).\textsuperscript{21} ShExJ file is given as input to the tool. This file is converted to JSON-LD format (as TD is serialized in JSON-LD format) based on different elements and types declared in it. These conversions are made recursively and gives the output as a TD template.

The example shape of Aircondition Capability presented in Listing 1.2 is used to ensure that the prototype can work and do the transformation as expected. This example includes different elements of ShEx as \textit{NumericFacet}, \textit{ValueSets}, \textit{EachOf} shape, \textit{And} shape and so on. Although it is a small example, it has the structure of a typical ShEx Shape and the prototype can convert it properly. The resulting semantically enriched TD template is shown in Listing 1.3.

A generated TD template have some blanks that should be filled with the attributes of a physical thing. For example, \textit{link} section of the Interaction patterns should be filled with the \textit{href} of the Interaction, that is the link where an Interaction can be accessed and the \textit{mediaType} which indicates the format an Interaction pattern’s data. After filling this information, the template becomes a TD which can be validated using thingweb-playground tool.\textsuperscript{22}

The resulting TD is semantically enriched with the concepts of an IoT Capability. This enhances discovery of devices and semantic interoperability between devices.

6 Discussion

Creating IoT Capabilities, marking-up Thing Descriptions with them and semantic discovery with such Thing Descriptions has be tested in W3C WoT Plugfest

\textsuperscript{19} https://github.com/aparnasai/iotschema/tree/iotschema-TDGenerator
\textsuperscript{20} https://github.com/shexSpec/shex.js
\textsuperscript{21} https://shexspec.github.io/spec/#shape-expressions-shex
\textsuperscript{22} https://github.com/thingweb/thingweb-playground
during the WoT meeting in November, 2017. The results were very encouraging, Thing Descriptions from various manufacturers were easily discoverable based on terms from iot.schema.org. The discovery process is important as it precedes the development of a new application. An efficient discovery significantly reduces the time required for application development. The work in this paper further simplifies the application development by automating the generation of semantically enriched Thing Descriptions. Moreover, it enables specification of variants of Thing Descriptions to capture differences in physical devices of the same class. Shape Constraints are created based on well-defined semantic terms from iot.schema.org, rather than on strings (as it the case in JSON).

iot.schema.org tend to over specify capabilities of physical devices, as they need to capture all possible features of one class of devices. On the other hand, it is desirable to be able to constrain such specification for devices that do not support all features. Before RDF Shape languages it was a challenge to constrain RDF graphs under the Open World Assumption of OWL. On the other hand, in IoT it is critical to restrict the interface of a device to exact requirements of the device (e.g., input/output data provided in certain range). Only then, interoperability with that device is possible. This is the reason why we propose RDF Shape languages to be used for constraining over specified IoT Capabilities. So, RDF Shape languages bring a Close World Assumption flavor to the interpretation of semantics of IoT Capabilities, which is important for building clients that interact with IoT devices.

The process of semantic mark-up of Thing Descriptions is error-prone. By automating this process, we avoid these errors. We believe that this work is a good contribution to both iot.schema.org and W3C WoT communities, as it provides an engineering solution for a real world problem, and it helps for the adoption of the technologies developed by these communities.

On the other hand, the limitation of this approach is that in some cases, there is no mapping from ShEx to JSON Schema, which may lead to loss of information in generated Thing Descriptions.

7 Conclusions & Future Work

In summary, we have addressed the problem of modeling device variants by extending IoT Capabilities with RDF Shape constraints. In this manner, we can constrain the semantics of IoT devices which is very important for semantic interoperability between IoT devices. We proposed to use ShEx Shape language to model constraints on IoT Capabilities. We have presented the procedure to model constraints on IoT Capabilities. These constraints express variants in a class of physical devices. Then, the IoT Capabilities along with shape constraints can be used to automatically generate semantically marked-up Thing Description templates. For this purpose, we proposed a mapping from ShEx to JSON Schema in order to convert input and output data of a Capability’s interactions to TD

interactions data (as TD uses JSON Schema to model interactions data). We also provided a prototypical implementation of TD generator from an IoT Capability and its shape constraints.

In the future, we will do further investigate on using SHACL for expressing constraints on IoT Capabilities. We will check how it could differ from using ShEx for this purpose.
Listing 1.1: Air Conditioner Capability, its Interaction patterns and Data
Listing 1.2: Shape Expressions on Air Conditioner Capability, Interaction patterns and Data
Listing 1.3: Air Conditioner Thing Description Template

```json
{
  "name": "AirConditioner",
  "@type": ["Thing", "AirConditioner"],
  "interaction": [
    {
      "name": "RunMode",
      "inputData": { "type": "string",
                     "enum": ["CoolMode", "EnergyOrPowerSaverMode"] },
      "outputData": { "type": "string",
                      "enum": ["CoolMode", "EnergyOrPowerSaverMode"] },
      "@type": ["Property", "RunMode"],
      "writable": true,
      "link": [{"href": "," ,"mediaType": ""}] },
    {
      "name": "Temperature",
      "outputData": { "type": "object",
                      "properties": {
                          "TemperatureData": {
                              "type": "float", "minimum": 5, "maximum": 35
                          },
                          "unit": "Celsius"},
                      "@type": ["Property", "Temperature"],
                      "writable": false,
                      "link": [{"href": "," ,"mediaType": ""}] },
    {  
      "name": "Target Temperature",
      "inputData": { "type": "object",
                     "properties": {
                         TemperatureData": {
                             "type": "float", "minimum": 5, "maximum": 35
                         },
                         "unit": "Celsius"},
                     "outputData": { "type": "object",
                                     "properties": {
                                         TemperatureData": {
                                             "type": "float", "minimum": 5, "maximum": 35
                                         },
                                         "unit": "Celsius"} },
                     "@type": ["Property", "TargetTemperature"],
                      "writable": true,
                      "link": [{"href": "," ,"mediaType": ""] }
      }
  ]
}
```

Bibliography