Prepared by: Petri Laari (petri.laari@ericsson.com)

Version: 1.0 (22.8.2023)

Extending ecosystem support in SDF

1 Introduction

The main goal of this work is to extend the ecosystem support in Semantic Definition Format (SDF) [7] with new ecosystems. In this work we took Bluetooth Mesh models as a new ecosystem to verify the capabilities of the current SDF specification. Bluetooth is one of the largest ecosystems of IoT devices, thus it is essential that SDF can support the features that are used in Bluetooth for the device specifications. The main target is to provide support for onboarding of Bluetooth devices into systems that use SDF for cross-ecosystem device support.

Bluetooth Mesh models are a new set of models that have been created when the Bluetooth mesh networking was introduced. The Bluetooth Mesh profile specification defines the fundamental requirements for nodes joining the Bluetooth mesh network. The Bluetooth Mesh model specification, in turn, defines further the basic functionality, together with the states that the node maintains and messages that a device uses to communicate with other nodes in the mesh network.

Before introducing the mesh models, the Bluetooth specified "services" and "characteristics" to define the Bluetooth nodes and their capabilities. Some of the Mesh models have adopted also selected characteristics.

In this document, we propose a way to model the Bluetooth nodes using the Semantic Definition Format (SDF) and list some open questions for further discussion. We also introduce shortly the JavaScript PoC implementation for making automatic conversions from the Bluetooth Mesh models to SDF. This work covers the conversion of Bluetooth states and messages into SDF properties, actions, and events.

At the time of writing, the Bluetooth Mesh models are not available as machine-readable definitions, e.g., in YAML or JSON. The model definitions are in various documents provided by the Bluetooth community and for the purpose of this work, we generated YAML definitions for selected Mesh models that can be used with the PoC implementation for testing the automatic conversion from a Bluetooth Mesh model to SDF.

2 Bluetooth Node

2.1 Bluetooth Node definition

A Bluetooth node is defined in Bluetooth Mesh model Overview specification as depicted in Figure 1.

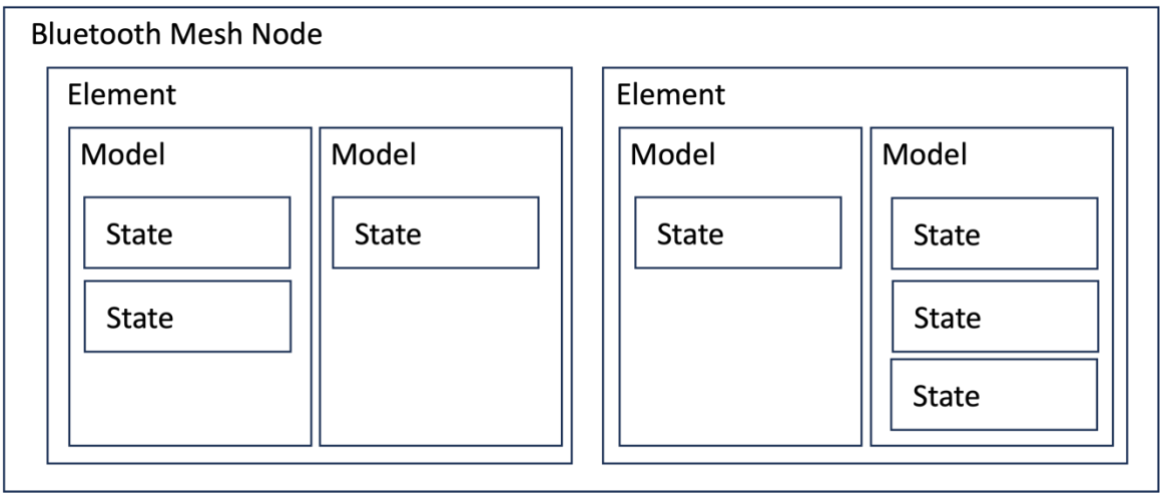

Figure 1 Bluetooth Node definition

Each Bluetooth node (device) consists of one or more elements, where each element is addressable from outside, e.g. with an IP address. Elements further consists of one or more Mesh models, defining the functionality of the element. The following Figure 2 (Bluetooth White Paper: Building a Sensor-Driven Lighting Control System Based on Bluetooth Mesh [5]) shows a luminaire with two elements, one for the light itself and one for the controller. The controller takes the input from various sensors and modifies the lighting state accordingly using the state binding between Light Lightness Linear state and Light LC Linear Output State. The light element can also be addressed directly from outside, using for example an on/off switch that manually can operate the light omitting the controller element.

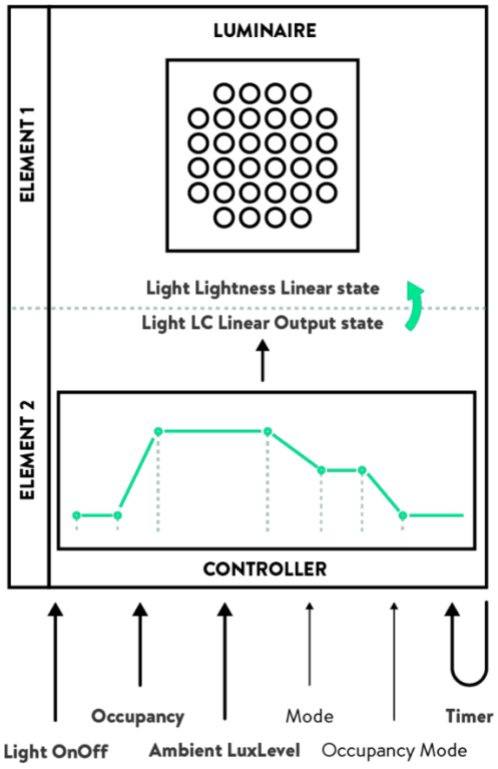

Figure 2 Lighting system consisting of two elements

Each of the Models consists of one or more States, describing the information that is stored on the device. For example, on lighting system, the Light CTL (Color-tunable Light) Temperature State maintains the color temperature value. The Models also specify the messages that the model can send and receive to communicate with other Models, for example for controlling the lighting color temperature, the client can send Light CTL Temperature Set, containing various parameters, such as the target temperature value and the execution delay at the server.

2.2 States

2.2.1 State information

"A state is a value representing a condition of an element" [1], section 2.3.1. This can be, for example, an on/off state defining if the status of the element is on or off. If a state contains multiple values, it is defined to be a composite state [2] section 1.4.1.2. A composite state is a way to combine multiple states and it is not a technical feature on the device, thus it is not needed to be mapped to the SDF and sdfProperties. A single state maps roughly to an sdfProperty in SDF.

Value	Description
0x0000	Light is not emitted by the element.
0x0001–0xFFFE	The perceived lightness of a light emitted by the element.
0xFFFF	The highest perceived lightness of a light emitted by the element.

Figure 3 Light Lightness Actual state

In Figure 3, The Light Lightness Actual state's possible values are depicted as defined in the Bluetooth Mesh model documentation. Figure 4 and Figure 5 show two ways how to convert that state definition with uint16 into SDF Property. The first one uses sdfChoice to describe the possible values, maintaining the detailed semantics of the smallest and largest value from the Bluetooth specification. The second one uses only one property with minimum and maximum values given.

```
"LightLightnessActual": {
  "label": "Light Lightness Actual",
  "description": "The Light Lightness Actual state ...”,
  "writable": true,
  "type": "integer",
  "sdfChoice": {
    "Light is not emitted by the element":
      {"const": 0},
    "The perceived lightness of a light emitted by the element": {
      "minimum": 1,
      "maximum": 65534
    },
    "The highest perceived lightness of a light emitted by the element":
      {"const": 65535}
  }
}
```

Figure 4 Using sdfChoice for representing uint16 to maintain semantics from the Bluetooth specification.

```
"LightLightnessActual": {
  "label": "Light Lightness Actual",
  "description": "The Light Lightness Actual state ...",
  "writable": true,
  "type": "integer",
  "minimum": 0,
  "maximum": 65535
}
```

Figure 5 Using Integer with min and max values to represent uint16.

2.2.2 Bound States

Relations between states are called state bindings. A binding defines the connection between two states, i.e., when one state value changes, it may indicate that the bound state must be recalculated. Bindings can be unidirectional or bi-directional. Bindings can be also *conditional* i.e., some other state may control the binding by enabling it or disabling it. How this is achieved is left for the Bluetooth device implementors. States can be bound within a node between various Elements and Models.

In the SDF class level definition, it may be enough to specify e.g. with sdfRelation defined in “Extending Relation Information for Semantic Definition Format (SDF)” internet-draft [6] the potential connection between two states. The details of the actual recalculation of state values are not considered to be part of the class-level definition but may be defined for example in the mapping file.

In the following Figure 6, the Generic Power Actual state is bound to the Generic OnOff state. When the Generic Power Actual state value goes to zero, the Generic OnOff value is set to off (Boolean false) and when it changes to some non-zero value, the Generic OnOff value is set to on (Boolean true).

```
{
  "sdfObject": {
    "GenericPowerLevelServer": {
      "label": "Generic Power Level Server Model",
      "description": "Power level server, Mesh model specification, 3.3.6",
      "sdfProperty": {
        "GenericPowerActual": {
          "label": "Generic Power Actual",
          "type": "integer",
          "minvalue": 0,
          "maxvalue": 65535,
          "sdfRelation": {
            "description": "when powerlevel goes to zero, the genericOnOff state
                            is set to off, and when it goes > 0, the state is set
                            to on. ",
            "relType": "https://example.com/bluetooth/boundstate",
            "target": "#/sdfObject/GenericPowerLevelServer/sdfProperty/GenericOnOff"
          }
        },
        "GenericOnOff": {
          "description": "Generic OnOff - status of the device",
          "type": "boolean"
        }
        ...
      }
    }
  }
}
```

Figure 6 Bluetooth Bound state example in SDF

2.3 Models

Bluetooth Mesh models are used to describe the basic functionality of the nodes in a mesh network. Mesh models include generic models defining functionality that is standard across device types, and models, such as lighting control, sensors, and time and scenes, to support key mesh scenarios.

Mesh models can be either clients or servers. The client models do not have states, but the server models have as they maintain the node related information. There are also two types of server models: the normal server model cannot access all the configuration information on the node, but the *setup server* can do that, thus there is a possibility to create different access policies.

3 Bluetooth Mesh Node and SDF

3.1 Semantic Definition Format

SDF is a metamodel and a format for describing capabilities of IoT devices. An SDF file contains one or more descriptions of ways to interact with a device using a protocol to retrieve data or change state. There are several key SDF use cases:

- **Device and system modeling:** An IoT system is described using SDF. At its simplest, an SDF document represents an IoT device with its associated interaction capabilities, such as what properties does it have, what actions can be done on the device and what kind of telemetry feeds can be set up. In a more complex scenario, the composition capabilities of SDF are used to build up and describe the various sub-components and how they relate to each other, resulting in a full ecosystem-independent description of the system.
- **Model translation:** The initial OneDM use case was to use SDF as an intermediate format in the translation from one ecosystem (e.g., OCF) to another (e.g., IPSO). This operation relies on data model mapping mechanisms contained in translators to work. The model translation can either be statically performed at system design time, or dynamically incorporated into a semantic gateway. The primary benefits of using SDF as intermediate format are that it reduces the number of necessary crosswise translations from NxN to Nx1 (i.e., every ecosystem needs to translate only to/from SDF instead of each other) and then streamlines those translations via common toolchains.
- **Common system description schema:** An SDF document representing a device or system can also be used as documentation to be used for e.g., automated creation of database schemas representing the device or system; or automated API generation towards the device or system.

3.2 Mapping a Bluetooth Mesh node to SDF

To enable conversion between the Bluetooth Mesh and SDF models, the Bluetooth Node model has to be mapped on high level to the SDF elements. The structure of the Bluetooth Node definition is not one-to-one with the SDF elements, thus there is a need to consider different alternatives for the mapping. The Bluetooth node has four levels of definitions as described in the previous subsection, while SDF has three: sdfThing, sdfObject, and a selection of affordances (sdfProperties, sdfActions, and sdfEvents).

In the following Figure 7, potential mapping between Bluetooth Mesh and SDF concepts are described. A Bluetooth node, containing four defined hierarchical elements (states, model, element, and node), can be mapped to SDF using sdfThing nesting. This is achieved by creating an sdfThing from the Bluetooth Node and further including sdfThing(s) that are created from the Bluetooth elements. The assumption is that these sdfThings, based on the elements that are addressable in Bluetooth, can be addressed using the SDF names. Further definitions of the concept are left for further study. The sdfThings created from the elements consists of sdfObjects that are based on the Bluetooth models, which further are consisting of sdfProperties, sdfActions, and sdfEvents matching the States and messages in the Mesh models.

However, later in this document, according to the selected approach in the activity, we do not consider the node and element entities, but we focus only on the models, states, and communication between the models, i.e., mesh model messages.

Bluetooth Mesh	SDF (nesting sdfThing)
Node	sdfThing
Element	sdfThing (nested)
Model	sdfObject
State	sdfProperty

Figure 7 Mapping Bluetooth Node to SDF

4 Example: Light Lightness Server model

In the following, we show some of the Mesh models and how they extend other models. Figure 8 shows the construction of a Light Lightness Server model. The root models on the left are models that are not extending any other models. These are further extended by other models towards the right. In the following subsections, we describe the yellow-marked models in more detail.

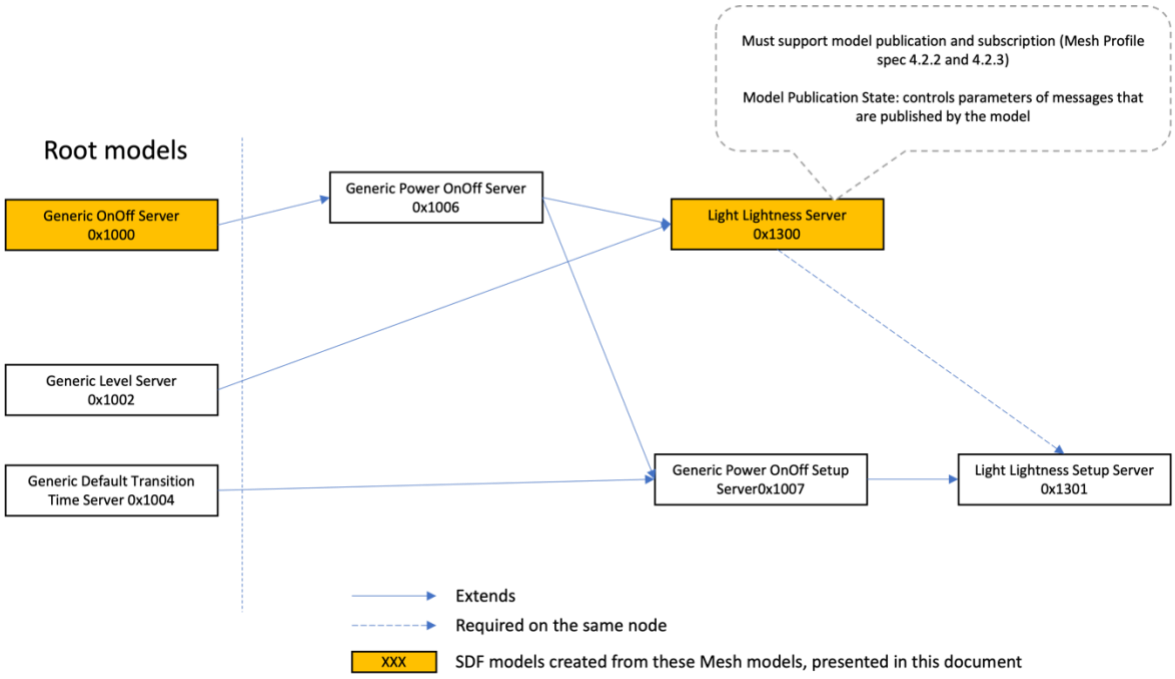

Figure 8 Light Lightness Server

4.1 Generic OnOff Server Model (SIG ID: 0x1000)

Bluetooth Models communicate with each other using messages. The different types of messages are defined in the model specifications.

In the following, the Generic OnOff Server (Mesh Model specification [2], section 3.1.1) translated to SDF is shown as an example. The Generic OnOff is a root model, which does not extend other models. The model consists of one state, Generic OnOff Server describing with zero or one, if the server is off or on. The Generic OnOff Server is described in

Bluetooth Mesh model [2] specification and the full Bluetooth Mesh model in YAML as well as SDF converted version in JSON are presented in Appendixes B, section 9.2 and C, section 9.3.

In the sdfAction section shown, we have defined the interface, how other models can interact with the OnOff server. The message can be *get* or *set*, and it can also be *set unacknowledged*, in which case the no response is expected as a response from the device. For the first two messages, the server responds with the *status* message.

In sdfData, the parameters are listed for both the incoming and outgoing messages. In the following example, the message "Generic OnOff Set Unacknowledged" is shown together with the parameters that the message contains.

The more complete SDF model can be found from Appendix E, Section 9.3

```
 "sdfAction": {
            "Generic OnOff Set Unacknowledged": {
              "description": "Set OnOff status, do not acknowledge",
              "sdfInputData": "#/sdfData/Generic_OnOff_Set_Unacknowledged"
            }
          }
          …
          "sdfData": {
            "Generic_OnOff_Set_Unacknowledged": {
              "type": "object",
              "properties": {
                "OnOff": {
                  "description": "The target value of the Generic OnOff
                                  state: 0 = off, 1 = on, boolean",
                "type": "boolean"
                },
                "TID": {
                  "description": "Transaction identifier",
                  "type": "integer"
                },
                "Transition Time": {
                  "description": "Transition Time, see section 3.1.3
                                  (Mesh Model specification)",
                  "type": "integer",
                  "minimum": 0,
                  "maximum": 255
                },
                "Delay": {
                  "description": " Message execution delay in 5
milliseconds
                                  steps",
                  "type": "integer"
                }
              }
            },
```

Figure 9 Partial SDF model of Generic OnOff Server: sdfAction

4.2 Light Lightness Server (SIG ID: 0x1300)

The Light Lightness Server model extends the Generic OnOff Server model (among others). For the full SDF model, see Appendix D in Section 9.4. This section focuses on the Bluetooth Mesh model requirement, where two model implementations need to reside on the same node. In this case, the Light Lightness Setup Server must exist on the node. Thus, we define the requirement using sdfRelation (defined in an SDF extension Internet-Draft draft-laari-asdf-relations [6])

```
"sdfObject": {
  "lightLightnessServer": {
    "label": "lightLightnessServer",
    "description": "Extends Generic Power OnOff Server model",
    "sdfRelation": {
      "relType": "bt:#/Relationtypes/MandatoryOnSameNode",
      "target": "bt:#/sdfObject/LightLightnessSetupServer",
      "description": "Setup Server needed on the same node",
      "maxItems": 1,
      "minItems": 1
    }
    ...
  }
}
```

Figure 10 Light Lightness Server; another model needed to be implemented on the same node

Another requirement for Light Lightness Server is that it must support also Model Publication state. This state is a composite state that controls parameters of messages that are published by a model. Within an element, each model has a separate instance of Model Publication state. The Model Publication state consists of six individual states. When presented using SDF, these states will be presented as separate sdfProperties.

4.3 Light LC Server & Light LC Setup server model

The Light Lightness Controller (LC) server (Figure 11) controls lightness of an element implementing a Light Lightness Server model through a binding with the Light Lightness Linear state of an element. It extends the Light Lightness Server model and contains also some device Property States.

The Property States in Light LC Server define the configuration of the actions on the server, i.e., how the device should respond to the incoming sensor messages. These property states define for example the delay between the occupancy sensor informing that people are detected in the area and the event when the light is turned on. The device properties for each Mesh model are listed in Bluetooth Mesh Model specification [2] and further definitions can be found from Mesh Device Properties [4]. The device properties that are used for the Mesh models are listed in Section 9.1, Appendix B.

The Light LC Setup server (0x1310) further extends the Light LC Server and allows the control of the property states enabling changes in the behavior of the Light LC Server.

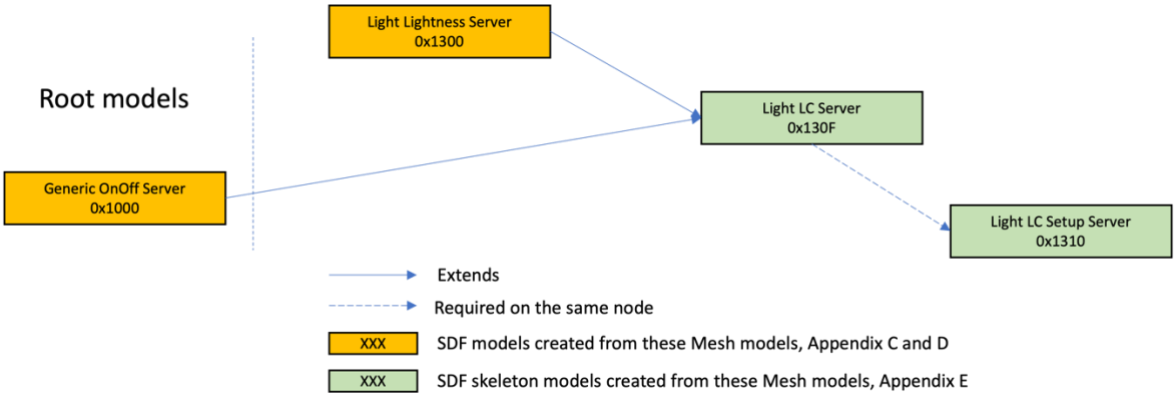

Figure 11 Light LC Server, Light LC Setup Server

5 Mapping details

Most of the mappings are straightforward, but there are some corner cases that may need some more attention, in this subsection we list some of the cases that may need more evaluation.

5.1 Characteristics

Before the mesh models were introduced in Bluetooth, the node operations were described as *services* and *characteristics*. A service represents a feature of a device and characteristics are items of data that belong to a particular service [9]. The way the models are now presented in Mesh models is different from the older version. However, the characteristics can be still used as describing the state information in Mesh models and a couple of them have been defined to be used in some of the Mesh models via the device properties. These are defined as "property states" in the Mesh documentation. The characteristics are defined in YAML files and they can be converted automatically with reasonable fidelity to SDF. The tool that we have, supports this conversion (see section 6).

One characteristic file can contain multiple values defined in different "fields". Each of them can be translated into a separate sdfProperty, containing a single defined value. In this work, we limited the conversion only to Mesh models and all the characteristics that are required for the Mesh models contain each only one value field. Thus, we are not converting the characteristic file itself, but we are including only the required values from the characteristic to the sdfProperty in the conversion.

For further study: There are characteristics that contain multiple fields which may benefit from converting a single characteristic into a separate sdfObject, with multiple sdfProperties. However, this is out of the scope of this work, but the conversion mappings are listed in the conversion table in Figure 12 for completeness.

In this section we have considered only those characteristics, that are used by the Mesh models. They all define only a single value and are trivial when mapping to SDF. Some other characteristics can have more complex structures, e.g., having multiple values or types of struct that are further defined in the Bluetooth documentation.

Characteristics used in the Mesh models map to device property states and in SDF they map to SDF Properties. The following table shows two alternative mappings from the Bluetooth Characteristic to corresponding SDF entities:

BT Characteristic	SDF Entity (to sdfObject), Note: For further study **)	SDF Entity (to a single sdfProperty)	Notes
Characteristic identifier	-	-	URI identifying the characteristic
Characteristic name	sdfObject name	-	This is the name of the characteristic
Characteristic description	sdfObject Description	-	This description goes to the SDF info block
Structure	sdfProperty	sdfProperty	Status/property definitions
Field	Name of sdfProperty definition	Name of sdfProperty definition	Identifying the sdfProperty
Field type	type data quality	type data quality	Type of data (e.g., used by property), see also section 5.3
Field size	minimum/maximum data quality *)	minimum/maximum data quality *)	Depends on the type in BT characteristic
Field Description	Description of the sdfProperty	Description of the sdfProperty	Description of this property. Information about the characteristic, e.g., units, may be included in this field as human-readable text (that needs to be parsed for automated translation).

Figure 12 Mapping BT Characteristic to SDF

*) This conversion depends on the type. For example, if type is uint16 (in SDF “integer”), this converts to SDF data qualities of minimum 0 and maximum 65535. The characteristics used in this work are all numerical types, thus there is no need to define size separately, but it can be calculated from the type, minimum, and maximum qualities.

**) This column demonstrates converting characteristics containing multiple value fields into sdfObject (for further study).

5.2 Extending Model

Bluetooth Mesh defines a set of root models that can be extended by other Mesh models to create models defining more functionality. In SDF, sdfRef can be used to define that this model is extending some other model. This sdfRef:ed model can be copied completely into

this SDF model. In Bluetooth Mesh models, the operation is similar when the extended objects are part of the extending model.

5.3 Data types

There is a wide variety of data types in Bluetooth models. These are not directly mapped to SDF, but they can be represented using “number” or “integer” providing minimum and maximum values for them, e.g., BT Mesh type “uint16” is converted to "integer" with minimum 0 and maximum 65535 in the sdfProperty definition.

Some values that are defined for the models in the Bluetooth specification are scaled in the implementation. That is, there may arrive integer n which is stored in the state and it is further scaled at the application to represent the real value. This applies also to the characteristic, defined in GATT (GSS) specification where the representative values are calculated from the raw stored value using a formula (see Section 9.1, Appendix B for the GATT characteristic calculations that are relevant for the Mesh models). This is a protocol binding issue and requires definitions in the corresponding companion document specifying the protocol specific details.

6 Implementations

6.1 Bluetooth to SDF: bt2sdf.js

This section describes our current implementation for converting Bluetooth models to SDF. *The implementation will be available later in the GitHub*.

The implementation supports various conversions, but the most important one related to this document is the Mesh model YAML to SDF conversion:

```
node bt2sdf.js -y mesh-model-file.yaml
```

NOTE: The conversion software is under development, and the description in this section may change or become invalid. However, the accompanying README file will be up-to-date and more information can be found from there.

As an example, one YAML file that we generated for the “Generic OnOff Server” can be found from the Appendix B, section 9.2.

6.2 Mesh Model YAML file design

For testing the automatic conversion, we created a simple YAML design for the Mesh models. The current example is the generic OnOff Server, which is a root model and does not require other models. The example file is available at the GitHub repository.

6.2.1 Entities in the YAML

The YAML file design currently uses the following entities:

model - this defines that this is a Mesh model

states - the state information from the Mesh model specification

messages - the messages that are either sent or received by the state

fields - these are the parameters of the messages, containing now the type, size and description. The direction is defined either "in" or "out" depending if the parameter is arriving or leaving the state. The field value *Response* provides a reference to another parameter set is used for a response messages. References are used to avoid duplicate definitions in the file.

6.3 Practical issues

- The message parameters in the YAML file are currently simply defined to be "in" or "out" depending on the message direction. This relates to the SDF so that parameters “in” can be mapped to sdfInputData and “out” to sdfOutputData.
- The Mesh specification defines "boolean" as an uint8 value "0" or "1", which is further in the mesh model specification text defined to be actually *boolean*. However,such free-form text description is not a robust option for automatic translation. The current translator implementation searches for a string "boolean" in the description of uint8 fields and translates such fields to SDF boolean type.
- Bluetooth Mesh has different sizes of unsigned and signed integers. The different types of data types here are implemented in SDF using "number" type with *minimum* and *maximum* fields, which allows also reverse conversion if needed
- When a message contains a set of parameters, they are now converted to sdfData, with type "object" and all parameters are listed as properties of the object.

7 Conclusions and discussion topics

From the manual mapping between Mesh and SDF models, we can see that the conversion is possible, if there are machine readable Mesh model files available with sufficient information. We have also created a proof of concept implementation that takes a Mesh model YAML file as input and provides SDF output. The YAML file is our own design as machine readable format for Mesh models is not currently defined by the Bluetooth SIG. The mesh model YAML files can be manually created using the Mesh model specifications.

With the current knowledge, all Bluetooth Mesh elements seem to be possible to be described using SDF. This indicates that there is no need to design any extensions or do modifications to the basic SDF specification. However, not all the Mesh models have a strict structure and may contain exceptions that may affect the formal definitions. This in turn may have effect on the SDF definitions.

7.1 Discussion topics

- Any definition for Mesh Model Subscribe / Publish? How this is handled in Bluetooth? The way the subscriptions are made is not obvious from the Mesh Profile, nor Model documents, not at least in the Mesh models. Shall we have some specific sdfActions for subscribing or sdfEvent for publishing changed state information to support this in the SDF model?
- Mesh model bound states: In this document one solution is described, but is it the way to go? Each bound state requires some operation / recalculation at the other state, but where

should these be described? One way to do it is to include this in the mapping file, or then it left for the developer to figure out. This should follow the Bluetooth thinking.

- Current data type conversion of integer values with specific type in Bluetooth spec (e.g., uint8, uint16) is done by using "integer" SDF type with minimum and maximum values to denote the possible range of such types. This allows also reverse conversion to original data types.
    - Note: there are cases, where the field defines clearly Boolean operation (on/off), but the field is still using uint8. Values 2-255 are described as prohibited.
- If there is a requirement in the BT Mesh model specification, such as in section 5.3, that another model must be implemented on the same node with the specified model, is that something to specify also in SDF model and if yes, how we can define it there? One option is to use the sdfRelation extension to describe the relation to another component with such information.
- Characteristics: currently only simple Bluetooth characteristics are used, i.e., one characteristic file can be converted into single sdfProperty. However, there are other characteristics not used in mesh models, that contain more information. This may indicate that they should be converted to sdfObjects, but it is out of the scope of this evaluation work.
- TID field: the Bluetooth Transaction identifier (TID) is used in some cases as a parameter to identify a specific message exchange. This may be a protocol mapping issue and not actually modeling issue.
- Mesh model defines messages that either do or do not require acknowledgement. Should these be converted to separate messages in sdfActions, or shall the unacknowledged messages be handled only as protocol issues (possibly with mapping file)?
- Is the suggested sdfRef usage correct when extending another model? While all the properties from the original models are included in the model, the sdfRef would provide this for SDF. Example: Appendix D: extending GenerifOnOff and GenericLevel

Thanks to Ericsson team Ari Keränen, Niklas Widell, and Lorenzo Corneo, as well as to Michael Koster, Carsten Bormann, and Szymon Slupik for their input.

8 References

[1] Bluetooth Mesh profile specification, v 1.0.1, 21.1.2019, Mesh Working Group

[2] Bluetooth Mesh model specification, v 1.0.1, 21.1.2019, Mesh Working Group

[3] Bluetooth Mesh Models, technical overview, v 1.0, 27.3.2019, Martin Woolley

[4] Mesh Device Properties, v 2, 15.9.2020, Mesh Working Group

[5] Bluetooth White Paper: "Building a Sensor-Driven Lighting Control System Based on Bluetooth Mesh", v1.0, 25.8.2023, Mesh Working Group

[6] "Extending Relation Information for Semantic Definition Format (SDF)", 12.12.2022, IETF ASDF WG

[7] "Semantic Definition Format (SDF) for Data and Interactions of Things", 12.1.2023, IETF ASDF WG

[8] "Bluetooth Mesh Models: A Technical Overview" , 27.3.2019, Martin Woolley

[9] "A Developer's Guide to Bluetooth Technology", 10.8.2016, Martin Woolley

9 Appendixes

9.1 Appendix A: Property states and characteristics

In this Appendix, we list the device property states and the corresponding characteristics that are needed for the Mesh models.

9.1.1 Property states

These device property states are defined in the Mesh Device Properties specification to be used by some of the SIG Mesh models:

- Light Control Time Occupancy Delay
- Light Control Time Fade On
- Light Control Time Run On
- Light Control Time Fade
- Light Control Time Prolong
- Light Control Time Fade Standby Auto
- Light Control Time Fade Standby Manual
- Light Control Lightness On
- Light Control Lightness Prolong
- Light Control Lightness Standby
- Light Control Ambient LuxLevel On
- Light Control Ambient LuxLevel Prolong
- Light Control Ambient LuxLevel Standby
- Light Control Regulator Kiu
- Light Control Regulator Kid
- Light Control Regulator Kpu
- Light Control Regulator Kpd
- Light Control Regulator Accuracy
- Motion Sensed
- Time Since Motion Sensed
- People Count
- Presence Detected
- Present Ambient Light Level

9.1.2 Characteristics and conversions to SDF

Each device property is mapped to a certain characteristic from the GATT. The following characteristics are used by the property states:

- Time Second 16
- Time Millisecond 24
- Perceived Lightness
- Illuminance

- Coefficient
- Percentage 8
- Count 16
- Boolean

In the Bluetooth – SDF conversion, these are typically converted using the following template:

```
"sdfProperty":
  "<characteric_name_from_BT>": {
    "description": "<description_from_BT>",
    "type": "<type_from_BT_converted>",
    "minimum": <lower_boundary_from_BT>,
    "maximum": <upper_boundary_from_BT>
  }
}
```

For some property states, the raw value is not directly used, but the characteristic contains values that are used to calculate the actual value. In SDF, these values would be listed in the mapping file, accompanying the model file. Bluetooth specification provides a way to construct 'Representative values' for different uses using the following formula.

$R = C * M * 10^d * 2^b$, where

R = Represented value

C = raw value

M = Multiplier

d = decimal exponent

b = binary exponent

In the following all the characteristics that are needed in the Mesh models are listed and the corresponding values for calculating the representative value are shown when applicable:

Time Millisecond 24

This 24-bit field represents a period of time with a resolution of 1 millisecond.

M = 1, d = -3, b = 0

Time Second 16

This 16-bit field represents time value with a unit of 1 second.

Perceived Lightness

This represents the perceived lightness of a light.

M = 1, d = 0, b = 0

Illuminance

This represents a measure of illuminance.

M = 1, d = -2, b = 0

Coefficient

This is a generic coefficient value.

Percentage 8

This is an 8-bit value representing a percentage.

M = 1, d = 0, b = -1

Count 16

General 16-bit count value

M = 1, d = 0, b = 0

Boolean

This defines True or False for an item. In Bluetooth, this is defined to be 0 or 1 and for translation, this must be specified either separately, or then there must be an unambiguous way to determine it from the source file and convert it to SDF boolean.

9.2 Appendix B: Generic OnOff Server Mesh Model YAML description

```
model:
  identifier: org.bluetooth.meshmodel.generic_onoff_server
  name: Generic OnOff Server
  description: |-
    This is a generic onoff server, root model.
  states:
    - name: Generic OnOff
      description: |-
        This defines on-off state, boolean
      type: uint8
      values:
        - value: '0'
          description: Off
        - value: '1'
          description: On
  messages:
    - message: Generic OnOff Get
      description: |-
        Return the current GenericOnOff state value
      fields:
        - field: Response
          direction: out
          outdata: Generic OnOffStatus
    - message: Generic OnOff Set
      description: |-
        Set OnOff status, send acknowledgement
      fields:
        - field: OnOff
          direction: in
          type: uint8
          size: 1
          description: |-
            The target value of the Generic OnOff state: 0 = off, 1 = on
        - field: TID
          direction: in
          type: uint8
          size: 1
          description: |-
            Transaction identifier
        - field: Transition Time
```

```
      direction: in
      type: uint8
      size: 1
      description: |-
        Transition Time, see section 3.1.3 (Mesh Model specification)
    - field: Delay
      direction: in
      type: uint8
      size: 1
      description: |-
        Message execution delay in 5 milliseconds steps)
    - field: Response
      direction: out
      outdata: Generic OnOffStatus

- message: Generic OnOff Set Unacknowledged
  description: |-
    Set OnOff status, do not acknowledge
  fields:
    - field: OnOff
      direction: in
      type: uint8
      size: 1
      description: |-
        The target value of the Generic OnOff state: 0 = off, 1 = on
    - field: TID
      direction: in
      type: uint8
      size: 1
      description: |-
        Transaction identifier
    - field: Transition Time
      direction: in
      type: uint8
      size: 1
      description: |-
        Transition Time, see section 3.1.3 (Mesh Model specification)
    - field: Delay
      direction: in
      type: uint8
      size: 1
      description: |-
        Message execution delay in 5 milliseconds steps)
- message: Generic OnOffStatus
  description: Send out the status of the Generic OnOff Server
  fields:
    - field: Present OnOff
      direction: out
      type: uint8
      size: 1
      description: |-
        The present value of the Generic OnOff state
    - field: Target OnOff
      direction: out
      type: uint8
      size: 1
      description: |-
        The target value of the Generic OnOff state (optional)
    - field: Remaining Time
      direction: out
      type: uint8
```

```
          size: 1
          description: |-
            Format defined in section 3.1.3. of the Mesh Model
specification
```

9.3 Appendix C: Generic OnOff Server converted to SDF

This appendix shows the result of the automatic conversion from the YAML file shown in Section 9.2. to SDF using the tool that we have developed.

```
{
  "info": {
    "title": "Bluetooth Generic OnOff Server"
  },
  "namespace": {
    "ext": "http://example.com/default"
  },
  "defaultNamespace": "ext",
  "sdfObject": {
    "Generic OnOff Server": {
      "description": "This is a generic onoff server, root model.",
      "sdfProperty": {
        "Generic OnOff": {
          "description": "This defines on-off state, boolean",
          "type": "boolean"
        }
      },
      "sdfAction": {
        "Generic OnOff Get": {
          "description": "Return the current GenericOnOff state value",
          "sdfOutputData": "#/sdfData/Generic_OnOffStatus"
        },
        "Generic OnOff Set": {
          "description": "Set OnOff status, send acknowledgement",
          "sdfInputData": "#/sdfData/Generic_OnOff_Set",
          "sdfOutputData": "#/sdfData/Generic_OnOffStatus"
        },
        "Generic OnOff Set Unacknowledged": {
          "description": "Set OnOff status, do not acknowledge",
          "sdfInputData": "#/sdfData/Generic_OnOff_Set_Unacknowledged"
        },
        "Generic OnOffStatus": {
          "description": "Send out the status of the Generic OnOff
                          Server",
          "sdfOutputData": "#/sdfData/Generic_OnOffStatus"
        }
      },
      "sdfData": {
        "Generic_OnOff_Set": {
          "type": "object",
          "properties": {
            "OnOff": {
              "description": "The target value of the Generic OnOff
                              state: 0 = off, 1 = on, boolean",
              "type": "boolean"
            },
            "TID": {
              "description": "Transaction identifier",
              "minimum": 0,
              "maximum": 255
            },
            "Transition Time": {
              "description": "Transition Time, see section 3.1.3 (Mesh
```

```
                    Model specification)",
      "minimum": 0,
      "maximum": 255
    },
    "Delay": {
      "description": "Message execution delay in 5 milliseconds
                    steps)",
      "minimum": 0,
      "maximum": 255
    }
  }
},
"Generic_OnOff_Set_Unacknowledged": {
  "type": "object",
  "properties": {
    "OnOff": {
      "description": "The target value of the Generic OnOff
                    state: 0 = off, 1 = on, boolean",
      "type": "boolean"
    },
    "TID": {
      "description": "Transaction identifier",
      "minimum": 0,
      "maximum": 255
    },
    "Transition Time": {
      "description": "Transition Time, see section 3.1.3 (Mesh
                    Model specification)",
      "minimum": 0,
      "maximum": 255
    },
    "Delay": {
      "description": "Message execution delay in 5 milliseconds
                    steps)",
      "minimum": 0,
      "maximum": 255
    }
  }
},
"Generic_OnOffStatus": {
  "type": "object",
  "properties": {
    "Present OnOff": {
      "description": "The present value of the Generic OnOff
                    state: 0 = off, 1 = on, boolean",
      "type": "boolean"
    },
    "Target OnOff": {
      "description": "The target value of the Generic OnOff state
                   (optional), 0 = off, 1 = on, boolean",
      "type": "boolean"
    },
    "Remaining Time": {
      "description": "Format defined in section 3.1.3. of the
                    Mesh Model specifiation",
      "type": "integer",
      "minimum": 0,
      "maximum": 255
    }
  }
}
```

```
      }
    }
  }
}
```

9.4 Appendix D: Light Lightness Server SDF representation

This is a Light Lightness Server Mesh Model converted to SDF manually from the Mesh Model, and Mesh Profile specifications. Note that there may be errors and the mappings on each field may change in the future.

```
{
  "info": {
    "title": "Light Lightness Server (0x1300)",
  },
  "namespace": {
    "bt": "https://example.com/BTmeshmodels"
  },
  "defaultNamespace": "bt",
  "sdfObject": {
    "lightLightnessServer": {
      "label": "lightLignessServer",
      "description": "The Light Lightness Server model extends the
Generic Power OnOff Server model and the Generic Level Server model",
      "sdfProperty": {
        "LightLightnessActual": {
          "label": "Light Lightness Actual",
          "description": "The Light Lightness Actual state represents the
lightness of a light on a perceptually uniform lightness scale. This is
bound to Generic Level and Generic OnOff (BT specification)",
          "type": "integer",
          "minimum": 0,
          "maximum": 65535
        },
        "LightLightnessLinear": {
          "label": "Light Lightness Linear",
          "description": "The Light Lightness Linear state represents the
lightness of a light on a linear scale. The state is bound to the Light
Lightness Actual state.",
          "type": "integer",
          "minimum": 0,
          "maximum": 65535
        },
        "LightLightnessLast": {
          "label": "Light Lightness Last",
          "description": "The purpose of the Light Lightness Last state
is to store the last known non-zero value of the Light Lightness Actual
state, which is a result of a completed transactional change of the
state. ",
          "type": "integer",
          "minimum": 1,
          "maximum": 65535
        },
        "LightLightnessDefault": {
```

```
          "label": "Light Lightness Default",
          "description": "The purpose of the Light Lightness Default
state is to determine the lightness level of an element when powered up
and when the Generic OnPowerUp state bound to the Light Lightness
composite state is set to 0x01 (Default).",
          "type": "integer",
          "minimum": 0,
          "maximum": 65535
        },
        "LightLightnessRangeMinimum": {
          "label": "Light Lightness Range",
          "description": "Minimum value for the Lighting value range",
          "type": "integer",
          "minimum": 1,
          "maximum": 65535
        },
        "LightLightnessRangeMaximum": {
          "label": "Light Lightness Range",
          "description": "Maximum value for the Lighting value range",
          "type": "integer",
          "minimum": 1,
          "maximum": 65535
        },
        "PublishAddress": {
          "label": "Publish Address",
          "description": "Model Publish composite state: The Publish
Address state determines the destination address in messages sent by a
model. ",
          "type": "string"
        },
        "PublishPeriod": {
          "label": "Publish period",
          "description": "Model Publish composite state: The Publish
Period state determines the interval at which status messages are
published by a model.",
          "type": "integer",
          "minimum": 0,
          "maximum": 63
        },
        "PublishAppkeyIndex": {
          "label": "Publish Appkey Index",
          "description": "Model Publish composite state: The Publish
AppKey Index state is the global AppKey Index of the application key used
in messages sent by a model.",
          "type": "integer"
        },
        "PublishFriendshipCredentialsFlag": {
          "label": "Publish Friendship Credentials Flag",
          "description": "Model Publish composite state: The Publish
Friendship Credential Flag is a 1-bit state controlling the credentials
used to publish messages from a model. 0: Master security material used,
1: Friendship security material used. ",
          "type": "integer",
          "minimum": 0,
          "maximum": 1
        },
         "PublishTTL": {
          "label": "Publish TTL",
          "description": "Model Publish composite state: The Publish TTL
state determines the TTL value for outgoing messages published by the
mode",
```

```
          "type": "integer",
          "sdfChoice": {
            "The Publish TTL value": {
              "value": {
                 "minimum": 0,
                 "maximum": 127
              },
              "prohibited": {
                "minimum": 128,
                "maximum": 254
              },
              "Use Default TTL": {"const": 255}
            }
          }
        },
        "PublishRetransmitCount": {
          "label": "Publish Retransmit Count",
          "description": "Model Publish composite state: The Publish
Retransmit Count state is a 3-bit value controlling the number of times
that a message published by a model will be retransmitted",
          "type": "integer",
          "minimum": 0,
          "maximum": 7
        },
        "PublishRetransmitIntervalSteps": {
          "label": "Publish Retransmit Interval Steps",
          "description": "Model Publish composite state: The Publish
Retransmit Interval Steps state is a 5-bit value controlling the interval
between retransmissions of a message that is published by a model.",
          "type": "integer",
          "minimum": 0,
          "maximum": 31
        }
      },
      "sdfAction": {
        "LightLightnessGet": {
          "label": "Light Lightness Get",
          "description": "Get the value"
        },
        "LightLightnessSet": {
          "label": "Light Lightness Set",
          "description": "Set the value"
        },
        "LightLightnessSetUnacknowledged": {
          "label": "Light Lightness Set Unacknowledged",
          "description": "Set the value, do not send ack"
        },
        ...

      },
    "GenericOnOff": {
      "label": "Generic OnOff",
      "description": "Generic OnOff Server, extended by Light Lightness
Server",
      "sdfRef": "bt:#/GenericOnOffServer"
    },
    "GenericOnPowerUp": {
      "label": "Generic OnPowerUp",
      "description": "Generic OnPowerUp Server, extended by Light
Lightness Server",
      "sdfRef": "bt:#/GenericOnPowerUpServer"
```

```
    },
    "GenericLevel": {
      "label": "Generic Level",
      "description": "Generic Level Server, extended by Light Lightness
Server",
      "sdfRef": "bt:#/GenericLevel"
    }
  }
}
```

9.5 Appendix E: SDF skeletons for Light LC Server and Light LC Setup Server

```
{
  "info": {
    "title": "Light LC Server (0x130F)"
  },
  "namespace": {
    "bt": "https://example.com/bluetoothmeshmodels"
  },
  "defaultNamespace": "bt",
  "sdfObject": {
    "LightLCServer": {
      "LightLightnessServer": {
        "description": "Extending the Light Lightness Server",
        "sdfRef": "bt:/LightLightnessServer"
      },
      "sdfProperty": {
        "LightLCMode": {},
        "LightLCOccupancyMode": {},
        "LightLCLightOnOff": {},
        "LightLCOccupancy": {},
        "LightLCAmbientLuxLevel": {}
      },
      "sdfAction": {
        "GenericOnOffGet": {},
        "GenericOnOffSet": {},
        "LightLCModeGet": {},
        "LightLCModeSet": {},
        "LightLCOMGet": {},
        "LightLCOMSet": {},
        "LightLCLightOnOffGet": {},
        "LightLCLightOnOffSet": {},
        "SensorStatus": {}
      }
    }
  }
}
```

Figure 13 Light LC Server, SDF skeleton

```
{
  "info": {
    "title": "Light LC Setup Server (0x1310)"
  },
  "namespace": {
    "bt": "https://example.com/bluetoothmeshmodels"
  },
  "defaultNamespace": "bt",
  "sdfObject": {
    "LightLCServer": {
      "description": "Extending LightLCServer",
      "sdfRef": "bt:#/LightLCServer"
    },
    "sdfProperty": {
      "LightLCTimeOccupancyDelay": {},
      "LightLCTimeFadeOn": {},
      "LightLCTimeRunOn": {},
      "LightLCTimeFade": {},
      "LightLCTimeProlong": {},
      "LightLCTimeFadeStandbyAuto": {},
      "LightLCTimeFadeStandbyManual": {},
      "LightLCLightnessOn": {},
      "LightLCLightnessProlong": {},
      "LightLCLightnessStandby": {},
      "LightLCAmbientLuxLevelOn": {},
      "LightLCAmbientLuxLevelProlong": {},
      "LightLCAmbientLuxLevelStandby": {},
      "LightLCRegulatorKiu": {},
      "LightLCRegulatorKid": {},
      "LightLCRegulatorKpu": {},
      "LightLCRegulatorKpd": {},
      "LightLCRegulatorAccuracy": {}
    },
    "sdfAction": {
      "LightLCPropertyGet": {},
      "LightLCPropertySet": {},
      "LightLCPropertySetUnacknowledged": {}
    }
  }
```

Figure 14 Light LC Setup Server, SDF skeleton