|  |  |
| --- | --- |
| Input Contribution | |
| Meeting ID\* | SDS#42 |
| Title:\* | TR-0043-Modbus-interworking-procedures |
| Source:\* | JaeSeung Song, KETI, [jssong@sejong.ac.kr](mailto:jssong@sejong.ac.kr)  Sherzod Elamanov, KETI, [selamanov@gmail.com](mailto:selamanov@gmail.com) |
| Date:\* | 2019-09-16 |
| Input related to\* | TR-0043 Modbus Interworking |
| Intended purpose of  document:\* | Decision  Discussion  Information  Other <specify> |
| Impacted other TS/TR(s) | N/A |
| Decision requested or recommendation:\* | Add Section 7 Modbus interworking procedures |
| Template Version: January 2017 (Do not modify) | |

**oneM2M Notice**

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

# Introduction

This contribution proposes to add a new section for Modbus Interworking procedures.

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

# 7 Possible Solutions for oneM2M and Modbus Interworking

*This clause studies the possible solutions to realize oneM2M interworking with Modbus. Modbus-based devices can interwork with oneM2M system by usage of IPE that deploys on ASN, MN and IN, such as Modbus-based device connects to MN by IPE on MN. Resource mapping based on Modbus data model and operational procedure will be studied. Semantic method will also be consided in the solution.*

## 7.1 Exposure of Modbus Functions to the oneM2M System

### 7.1.1 Summary of Interworking Architecture for exposure of Modbus Functions

A Modbus-IPE exposing Modbus Functions to the oneM2M System is responsible for the creation of oneM2M Resources representing the exposed Modbus Functions on its own Registrar CSE. A single Modbus-IPE may expose Modbus Functions provided by one or more Modbus devices to the oneM2M System. A high-level summary of the relationship of Modbus devices providing Modbus Functions to be exposed to oneM2M and Modbus-IPE representing the exposed Modbus Functions is depicted in figure 7.1.1-1.



**Figure 7.1.1-1 Exposure of Modbus Functions to the oneM2M System via Modbus-IPE**

In the oneM2M System, Modbus devices are designed according to Smart Device Template (SDT) described in TS-0023 as SDT offers generic and flexible modeling structure for non-oneM2M devices including Modbus devices.

## 7.1.2 Registration

#### 7.1.2.1 <*AE*> resource representing a Modbus-IPE

The first step to support the Modbus interworking with the oneM2M System is to register a Modbus-IPE to its Registrar CSE as an <AE> resource. When the Modbus-IPE completes its registration with the Registrar CSE by initiating an <AE> Create request, an <*AE*> resource representing that Modbus-IPE is created as a result of the registration. This resource is a parent of multiple instances of a modbusDeviceSDT component representing the Modbus Slave device. In order to register another Modbus Master that manages one or more Modbus slave devices, a separate *<AE>* resource for that Modbus Masterneeds to be created. Figure 7.1.2.1-1 shows a resource tree structure of the IPE\_AE resource. <IPE\_AE> contains zero or more <modbusDevices> and <subscriptions> resources.

IPE\_AE

modbusDevice

<subscriptions>

0..n

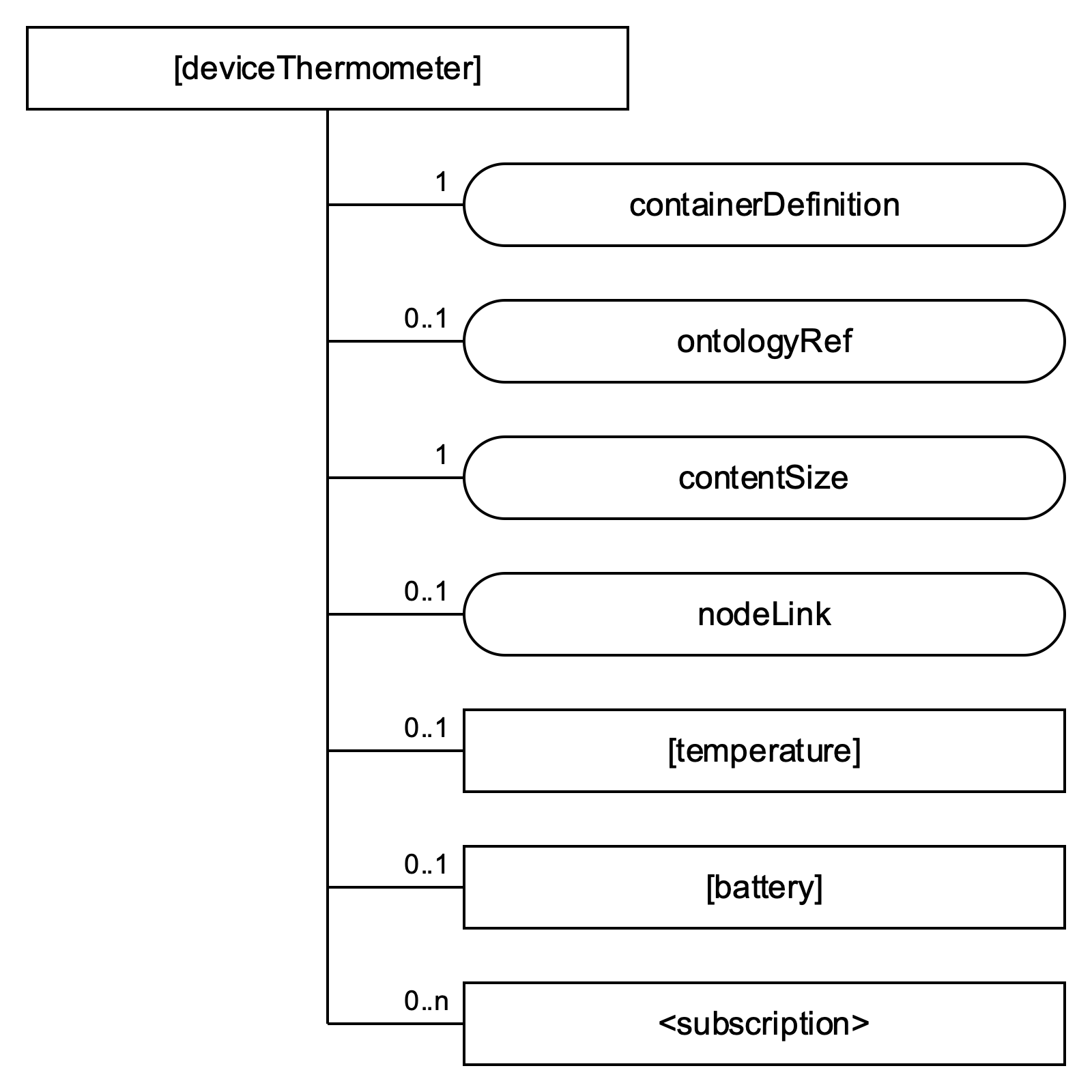
0..n

**Figure 7.1.2.1-1 *<AE>* resource representing a Modbus-IPE**

#### 7.1.2.2 modbusDevice as a <*flexContainer*> resource representing a Modbus Slave

modbusDeviceis an SDT Device which represents a single Modbus Slave device communicating with its Modbus Master. Mapping of the modbusDevice model to oneM2M resources is to be performed according to the general mapping procedure described in clause 6.2.2 of TS-0023. SDT Device component shall be mapped to a specialization of <*flexContainer*> resource with associated 'DeviceClass ID' (e.g. "org.onem2m.home.device.tv") on *containerDefinition* attributes.

Figure 7.1.2.2-1 shows an example of modbusDevice:*[deviceThermometer],* which is modelled as *a <flexContainer>* resource specialization derived from the corresponding SDT Device component. Model of *[deviceThermometer]* follows the schema described in clause 5.5.45 of TS-0023 but with small adjustment to support Modbus interoperability which is described in the next clause.



**Figure 7.1.2.2-1 *[deviceThermometer]* resource representing a Modbus Slave**

#### 7.1.2.3 Defining SDT DataPoints based on Modbus device’s register information

In order to enable interworking, a mapping between Modbus device’s registers and SDT DataPoints should be defined. Every Modbus register has following properties: *slave id*, *register type*, *address*, *length*. The information of these registers are typically provided by a manufacturer in a device’s datasheet. Register type and length are used to define following SDT DataPoint attributes: *DataType*, *writable* and *readable*. The rules to perform mapping are shown on Figure 7.1.2.3-1. A holding register and input register of length 2 can be mapped into either xs:integer or xs:float DataType depending on data context. As an example of mapping, a coil register can be mapped to a DataPoint with DataType (xs:boolean), Redable (True), and Writable (True).

These information can be stored in an optional attribute called nodnProperties of the <flexContainer> during the specialization of SDT to <flexContainer>.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Modbus Register | | Mapping | SDT Data points | | |
| Modbus register type | Length | DataType | Readable | Writable |
| Coil (1 bit, Read-Write) | 1 (1 bit) | 🡪 | xs:boolean | True | True |
| Discrete Input (1 bit, Read-Only) | 1 (1 bit) | xs:boolean | True | False |
| Holding Register (16-bit, Read-Write) | 2 (4 bytes) | xs:integer / xs:float | True | True |
| Input Register (16-bit, Read-Only) | 2 (4 bytes) | xs:integer / xs:float | True | False |
| Holding Register (16-bit, Read-Write) | 1 (2 bytes) | xs:integer | True | True |
| Input Register (16-bit, Read-Only) | 1 (2 bytes) | xs:integer | True | False |
| Holding Register (16-bit, Read-Write) | 4 (8 bytes) | xs:double | True | True |
| Input Register (16-bit, Read-Only) | 4 (8 bytes) | xs:double | True | False |

#### **Figure 7.1.2.3-1 Mapping between Modbus register types and SDT Data points**

#### 7.1.2.4 Temperature as a <*flexContainer*> resource specialization representing an SDT Module for Modbus device

Depending on the functionalities of a target Modbus device, one or more ModuleClasses, which are generic ModuleClasses, defined in TS-0023 can be used to design a ModuleClass for the target Modbus device. The derived ModuleClass shall describe all functional capabilities of the target Mobus device.

For the [*deviceThermometer*] example described in clause 7.1.2.2, two ModuleClasses, Temperature (see clause 5.3.76 in TS-0023) and Battery (see clause 5.3.10 in TS-0023), can be used. DataPoints of a parent ModuleClass (in this example Thermometer) are created according to the mapping rule described in clause 7.1.2.3. The child ModuleClasses (in this case Temperature and Battery) inherit from those generic ModulesClasses.

ModuleClass shall be mapped into <*flexContainer*> resource specialization, for example Temperature, and its data points shall be mapped into [*customAttributes*]of that <*flexContainer*> resource specialization. However, those ModuleClasses do not consider interworking options with a non-oneM2M Device Nodes (noDN) such as Modbus devices. For that reason, a *nodnProperties* shall be added as [*customAttribute*]of a <*flexContainer*> resource which is mapped from associated ModuleClass model.

The *nodnProperties* stores one-to-one mapping in serialized string format (e.g. JSON) between each DataPoint and a Modbus register from which it is created. *nodnProperties* shall contain *slave id*, *register type*, *address*, and *length* attributes for each DataPoint as well as a protocol type (in this case Modbus). An example content of *nodnProperties* is shown on Figure **7.1.2.4-1.**

**Figure 7.1.2.4-1 An example contents of *noDNproperties***

{"currentTemperature": {

"slaveID": 1,

"registerType": "inputRegister", "address": "23", "length": 2

},

"targetTemperature": {

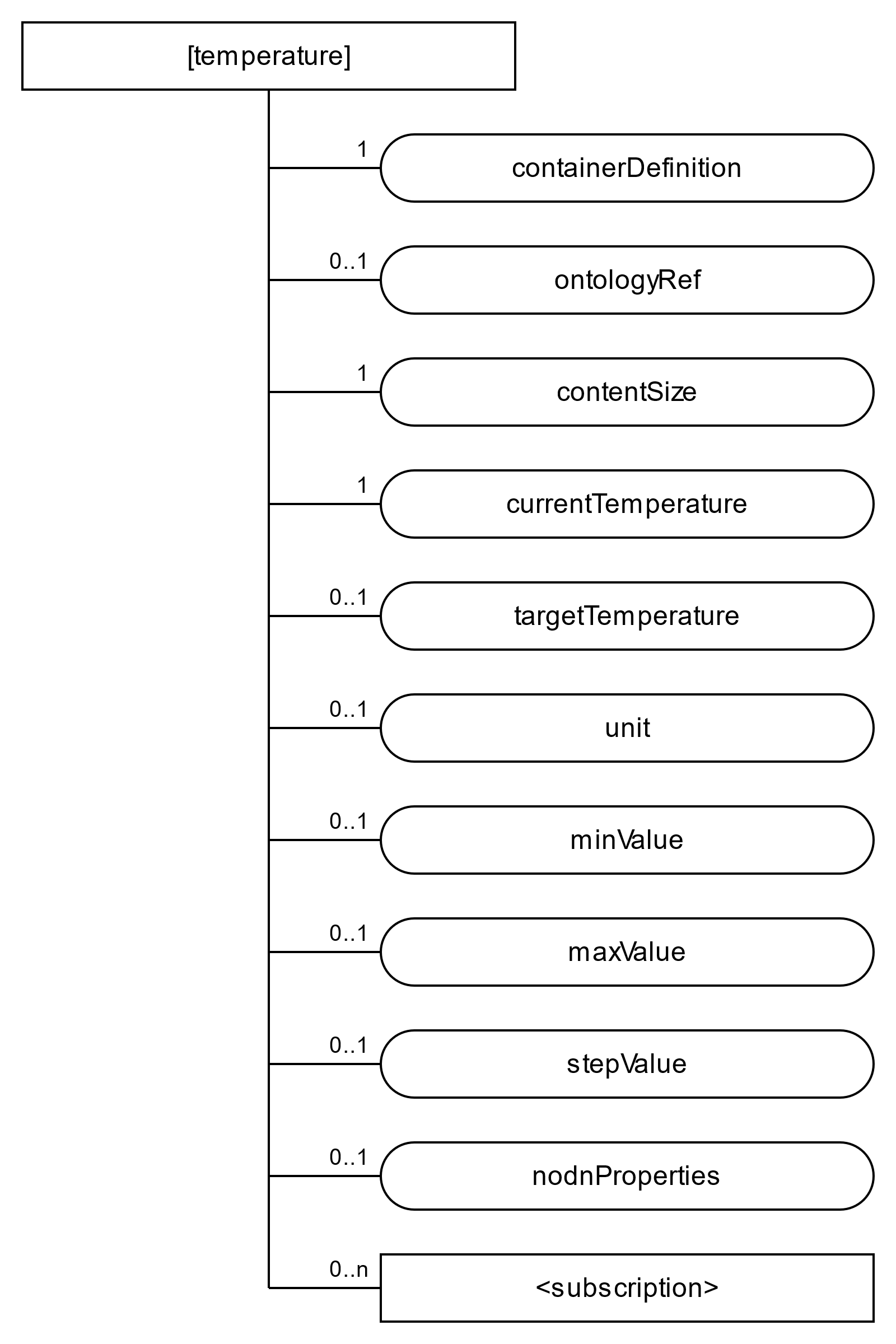
"slaveID": 1

"registerType": "holdingRegister", "address": "25", "length": 2

},  
"unit": {…},

…

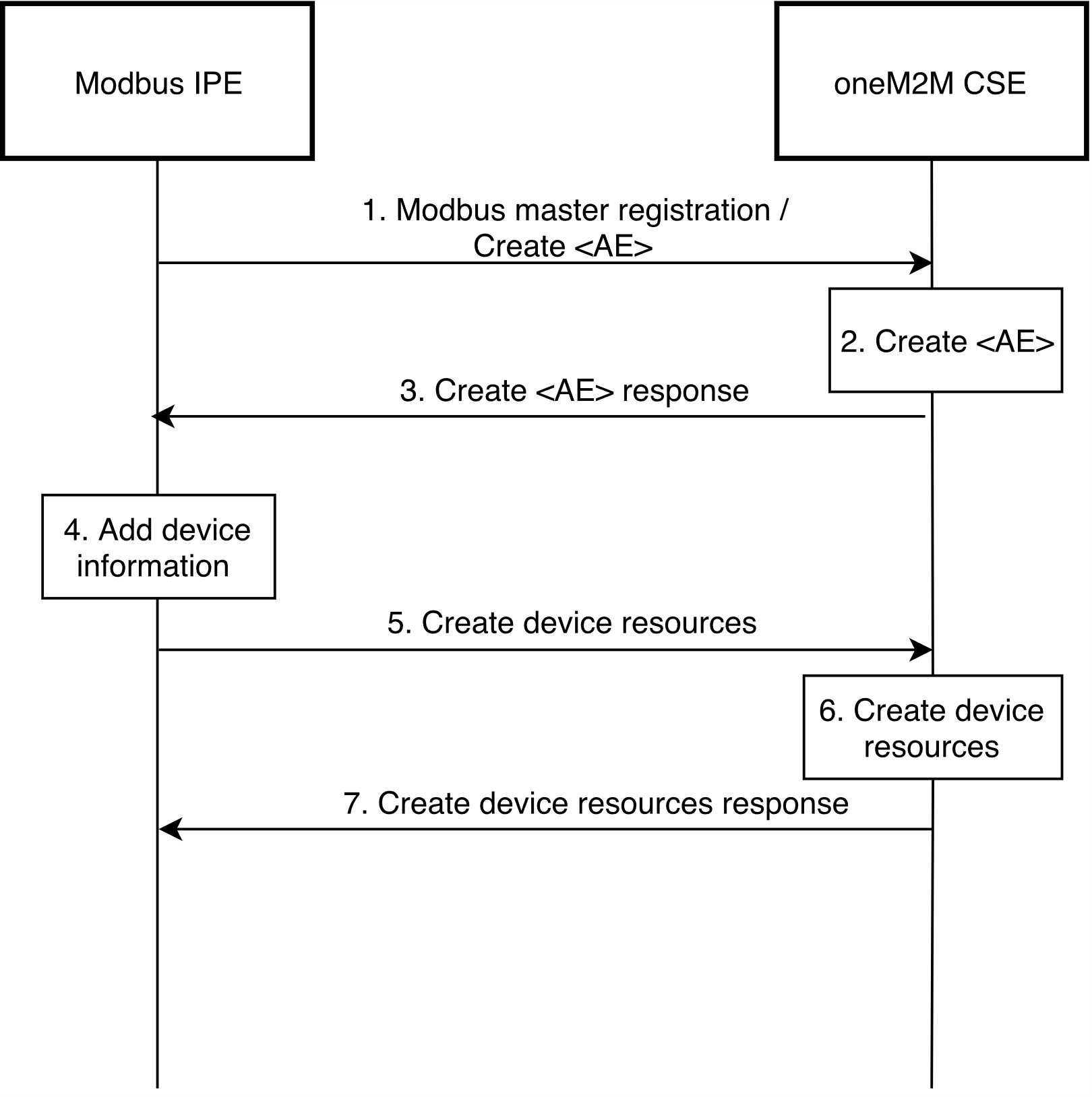
}

** Figure 7.1.2.4-2 shows an example of a Module, the specialization for TemperatureModuleClass*,* which isa *<flexContainer>* resource specialization derived from corresponding SDT Module component. Here it is assumed that DataPoints of *[Temperature]* resource specialization are derived as a result of the mapping procedures described in clause 7.1.2.3 and DataPoints are obtained as in clause 5.3.76 of TS-0023. *nodnProperties* [*customAttribute*] is added to support Modbus interworking.

**Figure 7.1.2.4-2** ***[Temperature]* as a <*flexContainer*> resource specialization representing an SDT Module for Modbus device**

#### 7.1.2.5 Modbus device registration call flow

Figure 7.1.2.5-1 shows the device registration call flow.

1. Modbus IPE sends Create *<AE>* request to a Hosting CSE to register Modbus-IPE (see clause 7.4.5.2.1 in TS-0004). The request must specify *App-ID* and *requestReachabily* attributes of the to be created <*AE*> resource. Other <*AE*> attribtes are optional.
2. After verifying the privileges and the given attributes, the Hosting CSE creates <*AE*> resource.
3. Hosting CSE shall respond with the successful result of *<AE>* resource creation, otherwise shall respond with an error.
4. Modbus devices are registered at Modbus IPE, in particular Modbus interworking information (slave id, registers type, address, length) are defined in accordance with provided device datasheet.
5. Modbus IPE sends corresponding requests to a CSE to create resourses as described in sections 7.1.2.1 - 7.1.2.3. For all <*flexContainer*> resources the *containerDefinition* attribute is mandatory*.* The *contentSize* attribute shall be calculated by Hosting CSE. *CustomAttributes* must be specified if they are mandatory for that <*flexContainer*>. Each resource creation shall be originated by Modbus-IPE in separate requests for each resource.
6. After verifying the privileges and the given attributes, the Hosting CSE creates each requested to be created resource.
7. ****Hosting CSE shall respond with the successful result for each created resource, otherwise shall respond with an error.

**Figure 7.1.2.5-1 Device registration call flow**

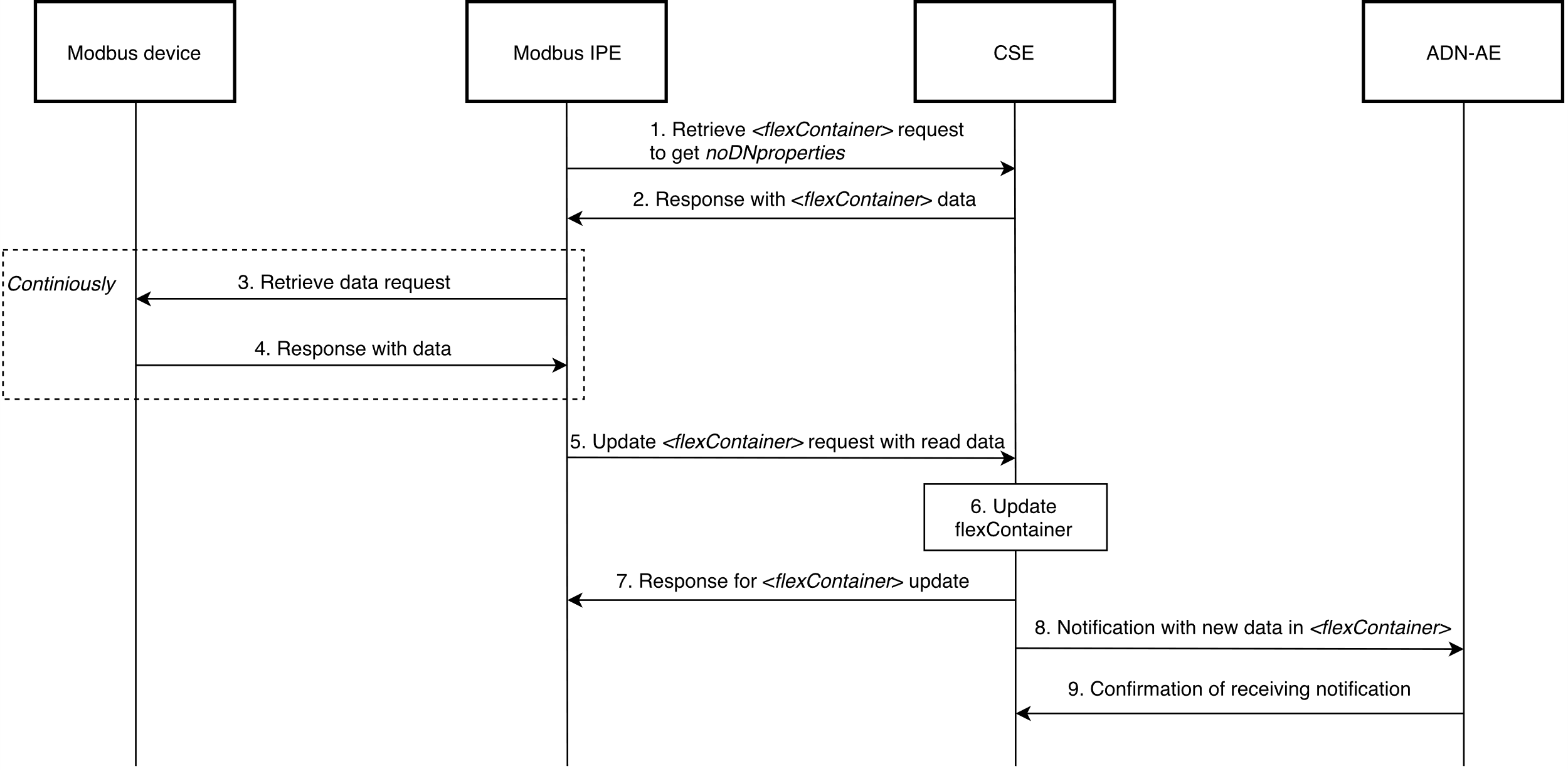
## 7.1.3 Retrieve data from a Modbus device monitoring

In a scenario when ADN-AE registered in a CSE hosting a Modbus-IPE needs to continuously monitor Modbus device the following steps described in the Figure 7.1.3-1 shall be performed:

1. Modbus IPE sends a retrieve <*flexContainer*> request to a hosting CSE. This <*flexContainer*> resource is a specialization of some Modbus module and contains *noDNproperties* attribute.
2. Hosting CSE shall respond for the retrieve request with <*flexContainer*> data.
3. Modbus IPE uses information stored in *noDNproperties* to compose Modbus read request. The function code to be used can be identified from a register type as in the Table 7.1.3-1. Slave id, address and length should written in correspong message fields. After Modbus message is composed Modbus IPE sends this message to Modbus device.

|  |  |
| --- | --- |
| **Register type** | **Function code** |
| Coil | 01 |
| Discrete input | 02 |
| Holding register | 03 |
| Input register | 04 |

**Table 7.1.3-1 Register type to function code mapping for Modbus read request**

1. Modbus device responds with requested data.
2. Modbus IPE sends an update <*flexContainer*> request (see clause 7.4.37.2.3 in TS-0004). The request body shall specify the *customAttributes* to be updated and their new values read from Modbus device.
3. After verifying the privileges and the given attributes, the Hosting CSE updates <*flexContainer*> resource.
4. The Hosting CSE shall respond with updated <*flexContainer*> data after successful update, otherwise shall respond with an error.
5. The Hosting CSE sends a notification with new values to subscribed to resource ADN-AE (see clause 7.5.1.2.2 in TS-0004).
6. ADN-AE send confirmation message about notification receiving to CSE (see clause 7.5.1.2.2 in TS-0004).

**Figure 7.1.3-1 Modus Slave Device monitoring call flow**

## 7.1.4 Write data to a Modbus Slave device

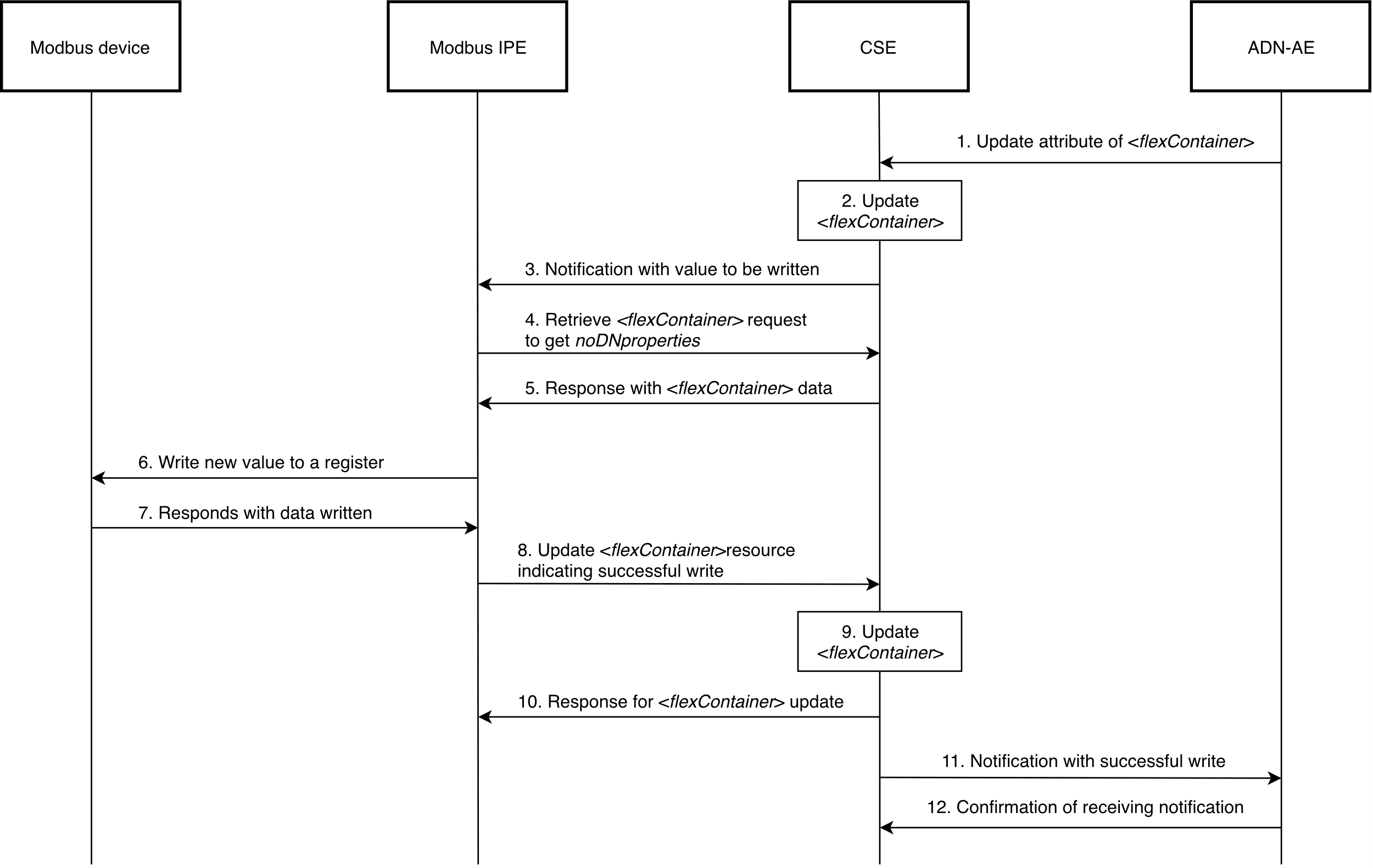
In a scenario when ADN-AE registered in a CSE hosting a Modbus-IPE wants to write data to a Modbus device the following steps described in the Figure 7.1.3-1 shall be performed:

1. In order to write data to Modbus device from ADN-AE, ADN-AE shall send a Update request and specify *customAttributes* of *<flexContainer>* resource to be written in Modbus Device (see clause 7.4.37.2.3 in TS-0004).
2. After verifying the privileges and the given attributes, the Hosting CSE updates <*flexContainer*> resource.
3. The hosting CSE sends a notification about received write request to a subscribed to resource Modbus IPE (see clause 7.5.1.2.2 in TS-0004).
4. Modbus IPE sends a retrieve <*flexContainer*> request to a hosting CSE. This <*flexContainer*> resource is a specialization of some Modbus module and contains *noDNproperties* attribute.
5. Hosting CSE shall respond for the retrieve request with <*flexContainer*> data.
6. Modbus IPE uses information stored in *nodnProperties* to compose Modbus write request. The function code to be used can be identified from a register type and length as in the Table 7.1.4-1. Slave id, adress and length should written in correspong message fields. After Modbus message is composed Modbus IPE sends this message to Modbus device.

|  |  |  |
| --- | --- | --- |
| **Register type** | **Length > 1** | **Function code** |
| Coil | false | 05 |
| Coil | true | 0F |
| Holding register | false | 06 |
| Holding register | true | 10 |

**Table 7.1.4-1 Register type and length to function code mapping for Modbus write request**

1. Modbus device responses with written data to Modbus IPE.
2. Modbus IPE sends a request to the hosting CSE to update the value *customAttribute* of <flexContainer> resource representing a register where data was written, otherwise respond with an error (see clause 7.5.1.2.2 in TS-0004).
3. After verifying the privileges and the given attributes, the Hosting CSE updates <*flexContainer*> resource.
4. The Hosting CSE shall respond with updated <*flexContainer*> data after successful update, otherwise shall respond with an error.
5. The hosting CSE sends a notification with written data to the subscribed to resource ADN-AE (see clause 7.5.1.2.2 in TS-0004).
6. ADN-AE send confirmation message about notification receiving to CSE (see clause 7.5.1.2.2 in TS-0004).

**Figure 7.1.4-1 Writing to a Modbus Slave Device call flow**

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