ONEM2M
TECHNICAL SPECIFICATION

Document Number	TS-0012-V2.2.2
Document Name:	Base Ontology
Date:	2018-03-12
Abstract:	oneM2M's base ontology constitutes a basis framework for specifying the semantics of data that are handled in oneM2M. Sub-classes of some of its concepts are expected to be defined by other bodies in order to enable semantic interworking. In particular interworking with non-oneM2M systems (e.g. Area Networks and their devices) should be facilitated.

Template Version:23 February 2015 (Dot not modify)

About oneM2M

The purpose and goal of oneM2M is to develop technical specifications which address the need for a common M2M Service Layer that can be readily embedded within various hardware and software, and relied upon to connect the myriad of devices in the field with M2M application servers worldwide.

More information about oneM2M may be found at: http//www.oneM2M.org

Contents

1 Scope 6

2 References 6
2.1 Normative references 6
2.2 Informative references 6

3 Definitions and abbreviations 7
3.1 Definitions 7
3.2 Abbreviations 7

4 Conventions 8

5 General information on the oneM2M Base Ontology (informative) 8
5.1 Motivation and intended use of the ontology 8
5.1.1 Why using ontologies in oneM2M? 8
5.1.1.1 Introduction to ontologies 8
5.1.1.2 The purpose of the oneM2M Base Ontology 9
5.1.1.2.0 Introduction 9
5.1.1.2.1 Syntactic interoperability 9
5.1.1.2.2 Semantic interoperability 10
5.1.2 How are the Base Ontology and external ontologies used? 10
5.1.2.1 Overview 10
5.1.2.2 Introduction to usage of classes, properties and restrictions 10
5.1.2.3 Methods for jointly using the Base Ontology and external ontologies 11
5.2 Insights into the Base Ontology 12
5.2.1 General design principles of the Base Ontology 12
5.2.1.1 General Principle 12
5.2.1.2 Essential Classes and Properties of the Base Ontology 13
5.2.2 Use of ontologies for Generic interworking with Area Networks 16
5.2.2.1 General Principle 16

6 Description of Classes and Properties 18
6.1 Classes 18
6.1.1 Class: Thing 18
6.1.2 Class: ThingProperty 19
6.1.3 Class: Aspect 20
6.1.4 Class: MetaData 21
6.1.5 Class: Device 22
6.1.6 Class: InterworkedDevice 24
6.1.7 Class: AreaNetwork 25
6.1.8 Class: Service 26
6.1.9 Class: Function 28
6.1.9.0 General description 28
6.1.9.1 Class: ControllingFunction 29
6.1.9.2 Class: MeasuringFunction 29
6.1.10 Class: Operation 30
6.1.10.0 General description 30
6.1.10.1 Class: GET_InputDataPoint 32
6.1.10.2 Class: SET_OutputDataPoint 32
6.1.11 Class: Command 33
6.1.12 Class: OperationInput 35
6.1.13 Class: OperationOutput 36
6.1.14 Class: OperationState 37
6.1.15 Class: InputDataPoint 39
6.1.16 Class: OutputDataPoint 40
6.1.17 Class: Variable 41
6.1.18 Class: SimpleTypeVariable 43
6.2 Object Properties 44
6.2.1 Void 44
6.2.2 Void 44

6.2.3 Object Property: consistsOf 44
6.2.4 Object Property: describes 44
6.2.5 Object Property: exposesCommand 45
6.2.6 Object Property: exposesFunction 45
6.2.7 Object Property: hasCommand 45
6.2.8 Object Property: hasFunction 45
6.2.9 Object Property: hasInput 46
6.2.10 Object Property: hasInputDataPoint 46
6.2.11 Object Property: hasMetaData 46
6.2.12 Void 46
6.2.13 Object Property: hasOperation 46
6.2.14 Object Property: hasOperationState 47
6.2.15 Void 47
6.2.16 Object Property: hasOutput 47
6.2.17 Object Property: hasOutputDataPoint 47
6.2.18 Object Property: hasService 47
6.2.19 Object Property: hasSubStructure 48
6.2.20 Object Property: hasThingProperty 48
6.2.21 Object Property: hasThingRelation 48
6.2.22 Void 48
6.2.23 Void 48
6.2.24 Void 48
6.2.25 Object Property: isPartOf 48
6.2.26 Object Property: refersTo 49
6.3 Data Properties 49
6.3.1 Data Property: hasDataType 49
6.3.2 Data Property: hasDataRestriction 51
6.3.2.0 General description 51
6.3.2.1 Data Property: hasDataRestriction_minInclusive 51
6.3.2.2 Data Property: hasDataRestriction_maxInclusive 51
6.3.2.3 Data Property: hasDataRestriction_minExclusive 51
6.3.2.4 Data Property: hasDataRestriction_maxExclusive 51
6.3.2.5 Data Property: hasDataRestriction_length 51
6.3.2.6 Data Property: hasDataRestriction_minLength 51
6.3.2.7 Data Property: hasDataRestriction_maxLength 51
6.3.2.8 Data Property: hasDataRestriction_pattern 51
6.3.2.9 Data Property: hasDataRestriction_langRange 51
6.3.3 Data Property: hasValue 52
6.3.4 Data Property: netTechnologyCommunicationProtocol 52
6.3.5 Data Property: netTechnologyPhysicalStandard 52
6.3.6 Data Property: netTechnologyProfile 52
6.3.7 Data Property: oneM2MTargetURI 53
6.3.8 Data Property: oneM2MAttribute 53
6.3.9 Data Property: oneM2MMethod 53
6.3.10 Data Property: hasThingAnnotation 54
6.4 Annotation Properties 54
6.4.1 Annotation Property: resourceDescriptorLink 54

7 Instantiation of the Base Ontology and external ontologies to the oneM2M System 55
7.1 Instantiation rules for the Base Ontology 55
7.1.1 Instantiation of classes of the oneM2M Base Ontology and derived external ontologies in the oneM2M System: 55
7.1.1.1 General on instantiating classes of the Base Ontology in the oneM2M System 55
7.1.1.2 Instantiation of individual classes of the Base Ontology 56
7.1.2 Instantiation of Object Properties 61
7.1.3 Instantiation of Data Properties 62
7.1.4 Instantiation of Annotation Properties 62
7.2 Common mapping principles between the Base Ontology and external ontologies 62

8 Functional specification of communication with the Generic interworking IPE 63
8.1 Usage of oneM2M resources for IPE communication 63
8.1.1 General design principles (informative) 63

8.1.2 Parent-child and linking resource relationships 64
8.2 Specification of the IPE for Generic interworking 65
8.2.1 General functionality of a Generic interworking IPE 65
8.2.2 Interworked Device discovery 66
8.2.3 Handling of DataPoints by the IPE 66
8.2.4 Handling of Operations by the IPE 67
8.2.5 Removing Devices 68
8.3 Specification of the behaviour of a communicating entity in message flows between IPE and the communicating entity 68
8.3.1 Preconditions on the communicating entity 68
8.3.2 Flow from the communicating entity to the IPE using InputDataPoints of a Service 69
8.3.2.1 Flow from the communicating entity to the IPE using a <*container*> type InputDataPoint 69
8.3.2.2 Flow from the communicating entity to the IPE using a <*flexContainer*> type InputDataPoint 70
8.3.3 Flow from the IPE to the communicating entity using OutputDataPoints of a Service 70
8.3.4 Flow from the communicating entity to the IPE using Operations of a Service 70
8.3.5 Flow from the IPE to the communicating entity using Operations of a Service 71

9 FlexContainer specializations for Generic interworking 71
9.1 Introduction 71
9.2 Resource Type *genericInterworkingService* 71
9.3 Resource Type *genericInterworkingOperationInstance* 74

Annex A (normative): OWL representation of Base Ontology 79

Annex B (informative): Mappings of selected external ontologies to the Base Ontology 80

B.1 Mapping of SAREF 80
B.1.1 Introduction to SAREF 80
B.1.2 Class mapping relationship between SAREF and the Base Ontology 81
B.1.3 Mapping SAREF to oneM2M resource structure 84
B.1.3.1 Introduction 84
B.1.3.2 Mapping rules 84
B.1.3.3 Example showing the uses of the semanticDescriptor resource and instantiation in the oneM2M resource structure 84

History 90

1 Scope

The present document contains the specification of the oneM2M base ontology. A formal OWL representation of the base ontology can be found at http://www.onem2m.org/ontology/Base_Ontology.

The present document also specifies an instantiation of the base ontology in oneM2M resources which is required for generic interworking.

In addition the present document contains the functional specification for an Interworking Proxy Application Entity (IPE), the oneM2M resources and their usage for generic interworking.

Finally an example is given how external ontologies can be mapped to the base ontology. The example uses the Smart Appliances REFerence (SAREF) ontology (http://ontology.tno.nl/saref).

2 References

2.1 Normative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

The following referenced documents are necessary for the application of the present document.

[1] oneM2M TS-0011: "Common Terminology".

[2] oneM2M TS-0001: "Functional Architecture".

[3] W3C Recommendation: "RDF 1.1 Concepts and Abstract Syntax".

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] oneM2M Drafting Rules.

NOTE: Available at http://www.onem2m.org/images/files/oneM2M-Drafting-Rules.pdf.

[i.2] The Smart Appliances REFerence (SAREF) ontology.

NOTE: Available at http://ontology.tno.nl/saref/.

[i.3] Open-source ontology editor PROTÉGÉ.

NOTE: Available at http://protege.stanford.edu/.

[i.4] W3C Recommendation: "OWL 2 Web Ontology Language Document Overview".

NOTE: Available at http://www.w3.org/TR/owl2-overview/.

3 Definitions and abbreviations

3.1 Definitions

For the purposes of the present document, the terms and definitions given in oneM2M TS-0011 [1] and the following apply:

annotation property: property that can be used to add information (metadata/data about data) to classes, individuals and Object/Data Properties

class: OWL standard ontology language from the World Wide Web Consortium (W3C) (see [i.4]), Concepts are called "Classes"

concept: entity of an Ontology that has an agreed, well defined, meaning within the domain of interest of that ontology

NOTE: A Concept is conceptually grouping a set of Individuals.

data property: property that relates an individual of a Class to data of a specified type and range

interworked device: non-oneM2M device (NoDN) for which communication with oneM2M entities can be achieved via an Interworking Proxy Application Entity (IPE)

generic interworking: generic interworking allows interworking with many types of non- oneM2M Area Networks and Devices that are described in the form of a oneM2M compliant ontology which is derived from the oneM2M Base Ontology

NOTE: Generic interworking supports the interworking variant "full mapping of the semantic of the non-oneM2M data model to Mca" as indicated in clause F.2 of oneM2M TS-0001 [2].

object property: property that relates an individual of a domain Class to an individual of a range Class

ontology: formal specification of a conceptualization, that is defining Concepts as objects with their properties and relationships versus other Concepts

property: in OWL standard ontology language Properties represent relations among individuals

NOTE: Properties can be sub-categorized as Object Properties, Data Properties and Annotation Properties.

proxied device: virtual Device (i.e. a set of oneM2M resources together with an IPE) that represents the Interworked Device in the oneM2M System

relation: (also called "interrelation" or "property") stating a relationship among individuals

restriction: describes a class of individuals based on the relationships that members of the class participate in

NOTE: Restrictions can be sub-categorized as: existential Restrictions, universal Restrictions, Cardinality restrictions and has Value Restrictions.

3.2 Abbreviations

For the purposes of the present document, the abbreviations given in oneM2M TS-0011 [1] and the following apply:

AE	Application Entity
OWL	Web Ontology Language
SAREF	Smart Appliances REFerence ontology
SPARQL	SPARQL Protocol and RDF Query Language

4 Conventions

The key words "Shall", "Shall not", "May", "Need not", "Should", "Should not" in the present document are to be interpreted as described in the oneM2M Drafting Rules [i.1].

5 General information on the oneM2M Base Ontology (informative)

5.1 Motivation and intended use of the ontology

5.1.1 Why using ontologies in oneM2M?

5.1.1.1 Introduction to ontologies

In a nutshell an ontology is a vocabulary with a structure. The vocabulary applies to a certain domain of interest (e.g. metering, appliances, medicine, etc.) and it contains concepts that are used within that domain of interest, similar to the "defined terms" in clause 3, "Definitions".

An ontology should:

- Capture a shared understanding of a domain of interest.
- Provide a formal and machine interpretable model of the domain.

The ontology lists and denominates these concepts which have agreed, well defined, meanings within the domain of interest (e.g. the concept of "Device" has an agreed, well defined, meaning within the scope of the Smart Appliances REFerence (SAREF) ontology see [i.2]).

Concepts do not identify individuals but they identify classes of individuals. Therefore, in the OWL standard ontology language from the World Wide Web Consortium (W3C) (see [3]), concepts are called "Classes".

The structure part of the ontology is introduced through agreed, well defined, relationships between its concepts. Such a relationship - in OWL called "Object Property" - links a *subject* concept to an *object* concept.

subject concept ➔ relationship ➔ *object* concept

in OWL:

domain Class ➔ Object Property ➔ *range* Class

EXAMPLE 1: In SAREF an Object Property "accomplishes" relates the "Device" class to the "Task" class:

Device ➔ accomplishes ➔ Task

Also the relationships/Object Properties of an ontology have agreed, well defined, meanings within the domain of interest. In the example above the "accomplishes" part of the relationship is well documented as part of SAREF (see [i.2]).

A second type of Properties in OWL is called "Data Properties". A Data Property is linking a subject Class to a data. These data may be typed or untyped.

EXAMPLE 2: in SAREF the Data Property "hasManufacturer" links the class "Device with data of datatype "Literal":

Device ➔ hasManufacturer ➔ Literal

Again, the Data Properties of an ontology have agreed, well defined, meanings within the domain of interest.

In the example 2, the Data Property "hasManufacturer" indicates that the Literal, that is linked via this Data Property will indicate the manufacturer of the Device.

Data Properties can be considered similar to attributes in oneM2M.

A third type of Properties in OWL is called "AnnotationProperties". An Annotation Property is used to provide additional information about ontology elements like classes and instances, which typically are external to the ontology and would not be used for reasoning. Example usages for such additional information are for providing a creator, a version or a comment. The object of an annotation property is either a data literal, a URI reference, or an individual.

In general, an individual of a certain Class may or may not have a particular relation (Object Property, Data Property or Annotation Property) that is defined by the ontology. However, if such a relation exists for the individual then that relation should be used with the meaning specified by the ontology.

One additional, crucial aspect differentiates an ontology from a vocabulary with a structure. An ontology enables specified, allowed constructs (based on predicate logic) and can be represented in a formal, machine interpretable form e.g. by the OWL standard ontology language. This allows the creation of queries (e.g. through the SPARQL query language) that search for individuals of specified classes, having specified relationships, etc.

The OWL flavour OWL-DL (where DL stands for "Description Logic"), that is used in the present document and that is supported by the ontology-editing tool "Protégé" (see [i.3]), has the additional advantage that it is underpinned by a description logic. For ontologies that fall into the scope of OWL-DL a reasoner can be used to automatically check the consistency of classes, take what has explicitly stated in the ontology and use it to infer new information. OWL-DL ensures that queries are decidable.

Additionally, OWL-DL allows the creation of Intersection, Union and Complement classes, restrictions (e.g. on the required/allowed number of relationships for any individual of the Class along this property) an other useful constructs.

5.1.1.2 The purpose of the oneM2M Base Ontology

5.1.1.2.0 Introduction

Ontologies and their OWL representations are used in oneM2M to provide syntactic and semantic interoperability of the oneM2M System with external systems. These external systems are expected to be described by ontologies.

The only ontology that is specified by oneM2M is the oneM2M Base Ontology, as described in the present document. However, external organizations and companies are expected to contribute their own ontologies that can be mapped (e.g. by sub-classing, equivalence..) to the oneM2M Base Ontology.

Such external ontologies might describe specific types of devices (as e.g. in the SAREF ontology) or, more generally, they might describe real-world "Things" (like buildings, rooms, cars, cities.) that should be represented in a oneM2M implementation. The value for external organizations and companies to provide their ontologies to oneM2M consists in supplementing oneM2M data with information on the meaning/purpose of these data. The OWL representation of that ontology provides a common format across oneM2M.

The oneM2M Base Ontology is the minimal ontology (i.e. mandating the least number of conventions) that is required such that other ontologies can be mapped into oneM2M.

5.1.1.2.1 Syntactic interoperability

Syntactic interoperability is mainly used for interworking with non-oneM2M devices in Area Networks. In this case an ontology - represented as an OWL file - that contains the Area Network specific types of communication parameters (names of operations, input/output parameter names, their types and structures, etc.) is used to configure an Interworking Proxy Entity (IPE).

With the help of this OWL file the IPE is able to allocate oneM2M resources (AEs, containers) that are structured along the Area Network specific parameters and procedures. This enables oneM2M entities to read/write from/into these resources such that the IPE can serialize the data and send/receive them from/to the devices in the Area Network.

The semantic meaning of these resources is implicitly given by the interworked Area Network technology.

Each ontology that describes a specific type of interworked Area Network needs to be derived from the oneM2M Base Ontology. In particular the device types of an ontology of an interworked Area Network need to be mapped (e.g. by sub-typing) into the concept "Interworked Device" of the oneM2M Base Ontology.

5.1.1.2.2 Semantic interoperability

Semantic interoperability is mainly used to describe functions provided by oneM2M compliant devices (M2M Devices).

EXAMPLE: Different, oneM2M compliant types of washing machines may all perform a Function like "washing-function" ,"drying-function", "select wash temperature", etc., however the oneM2M resources (containers), through which these functions can be accessed, can have different resourceNames, child-structures and type of content.

In this case an ontology - represented as an OWL file -contains the specific types of the M2M Application Service and/or Common Service of the M2M Device (e.g. CRUD operation, resourceNames, child-structures and type of content, etc.) together with the Function of that service (e.g. "washing-function").

Each ontology that describe a specific type of M2M Device needs to be derived from the oneM2M Base Ontology. In particular the device type needs to be mapped (e.g. by sub-typing) into the concept "Device" of the oneM2M Base Ontology.

5.1.2 How are the Base Ontology and external ontologies used?

5.1.2.1 Overview

This clause describes how an external ontology that is compatible with the Base Ontology can be used in a joint fashion.

NOTE: Further use of external ontologies is left to subsequent releases.

5.1.2.2 Introduction to usage of classes, properties and restrictions

An ontology consists of Properties and Classes.

Properties represent relationships, and link individuals from the specified domain (a class) to individuals from the specified range (another class). There are two main types of properties in the Base Ontology, object properties and data properties. An object property describes a relationship between two object individuals. A data properties describes a relationship between an object individuals and a concrete data value that may be typed or untyped.

Classes are interpreted as sets of individuals, and sometimes classes are also seen as a concrete representation of concepts. In the Base Ontology, a Class can be directly defined by the class name and class hierarchy or defined by the properties characteristics of the individuals in the class. The latter method is known as restriction. The classes defined by restriction can be anonymous, which contains all of the individuals that satisfy the restriction.

In the Base Ontology, the restrictions can be divided as existential restrictions, universal restrictions and cardinality restrictions:

- Existential restrictions describe classes of individuals that participate in at least one (some) relationship along a given property to individuals that are members of the class, e.g. since a Device (Class: Device) has at least one function (Object Property: hasFunction) (Class: Function) that this device accomplishes, then (Class: Device) is a subclass of the anonymous class of (Object Property: hasFunction) *some* (Class: Function).

- Universal restrictions describe classes of individuals that for a given property only have relationships along this property to individuals that are members of the class. For example, since a subclass "Watervalve" of (Class: Device) only has a Function (Object Property: hasFunction) subclass "Open_or_Close_Valve" of (Class: Function), then (Class:Watervale) is a superclass of the anonymous class of (Object Property: hasFunction) *only* (Class: Open_or_Close_Valve).

- Cardinality restrictions describe classes of individuals that, for a given property, only have a specified number of relationships along this property to individuals that are members of the class.

5.1.2.3 Methods for jointly using the Base Ontology and external ontologies

If the Base Ontology is available and the external ontologies are compatible with the Base Ontology, the Base Ontology and the external ontologies can be jointly used in the following ways:

1) Classes and properties mapping:

- The names of the class and properties in different ontologies may be totally different, but the meanings of these class and properties can be relevant. Classes and properties mapping is used to link the relevant classes and properties in different ontologies.

- The descriptions for the classes and properties mapping relationship of the Base Ontology and external ontologies can be given in an ontology or a semantic rule depending on the frequency of the usage. For the frequent cases, it is better to give the mapping description in an ontology, even in the Base Ontology.

- The classes and properties mapping can be based on the properties defined in OWL and RDFs, e.g. rdfs:subClassOf, owl:equivalentClass, for classifying the hierarchy of the classes and properties in Base Ontology and external ontologies. The inheritance from upper properties and classes will be implied according to the mapped hierarchy. For example, when a class A in an external ontology is mapped as a subclass of the class B in the Base Ontology, it implies that the properties of class B in the Base Ontology will be inherited by the class A in the external ontology.

Table 1 gives a simple example for classes and properties mapping between two ontologies.

Table 1: An example for classes and properties mapping between two ontologies

Properties mapping			**Classes mapping**		
property I	mapping relationship	property II	class I	mapping relationship	Class II
OntologyB: hasSwitch	rdfs:subPropertyOf	OntologyA: hasOperation	OntologyB: appliance	rdfs:subClassOf	OntologyA:device
OntologyA: hasPower	owl:equivalentProperty	OntologyB: hasPower	OntologyB:lamp	owl:equivalentClass	OntologyA:light
OntologyA: hasVendor	owl:equivalentProperty	OntologyB: hasManufacturer	OntologyB:Switch	rdfs:subClassOf	OntologyA:Operation

2) Individual annotation across multiple ontologies:

- Though the names of the class and properties in different ontologies may be totally different, the semantic annotation for individuals can be done based on these different ontologies respectively and independently. In this way, the knowledge from different ontologies are used together to describe the individuals.

Table 2 gives a simple example for individual annotation across two ontologies.

Table 2: An example for individual annotation across two ontologies

Individuals	**Semantic annotation based on Ontology A**		**Semantic annotation based on Ontology B**	
	Properties	**classes**	**properties**	**Classes**
Light A	rdf:type	Ontology A: Light	rdf:type	Ontology B:ledLight
	OntologyA: hasOperation	Ontology A:Open	OntologyB: hasColor	rdf:datatype="&xsd;string">'red'<
	OntologyA: hasStatus	rdf:datatype="&xsd;boolean">true<	OntologyB: hasSwitch	OntologyB:Switch
NOTE: The two methods can be used jointly or independently.				

The compatibility of two ontologies depends on their class hierarchies. When the class hierarchy of one ontology can be mapped as a part or an external part of the class hierarchy of the other ontology, they are compatible. When multiple ontologies are pairwise compatible, they are compatible.

5.2 Insights into the Base Ontology

5.2.1 General design principles of the Base Ontology

5.2.1.1 General Principle

The Base Ontology has been designed with the intent to provide a minimal number of concepts, relations and restrictions that are necessary for semantic discovery of entities in the oneM2M System. To make such entities discoverable in the oneM2M System they need to be semantically described as classes (concepts) in a - technology/vendor/other-standard specific - ontology and these classes (concepts) need to be related to some classes of the Base Ontology as sub-classes.

Additionally, the Base Ontology enables non-oneM2M technologies to build derived ontologies that describe the data model of the non-oneM2M technology for the purpose of interworking with the oneM2M System.

The Base Ontology only contains Classes and Properties but not instances because the Base Ontology and derived ontologies are used in oneM2M to only provide a semantic description of the entities they contain.

Instantiation (i.e. data of individual entities represented in the oneM2M System - e.g. devices, things, etc.) is done via oneM2M resources.

The Base Ontology is available at the web page:

- http://www.onem2m.org/ontology/Base_Ontology;

which contains the latest version of the ontology and individual versions of the ontology (see Annex A)

5.2.1.2 Essential Classes and Properties of the Base Ontology

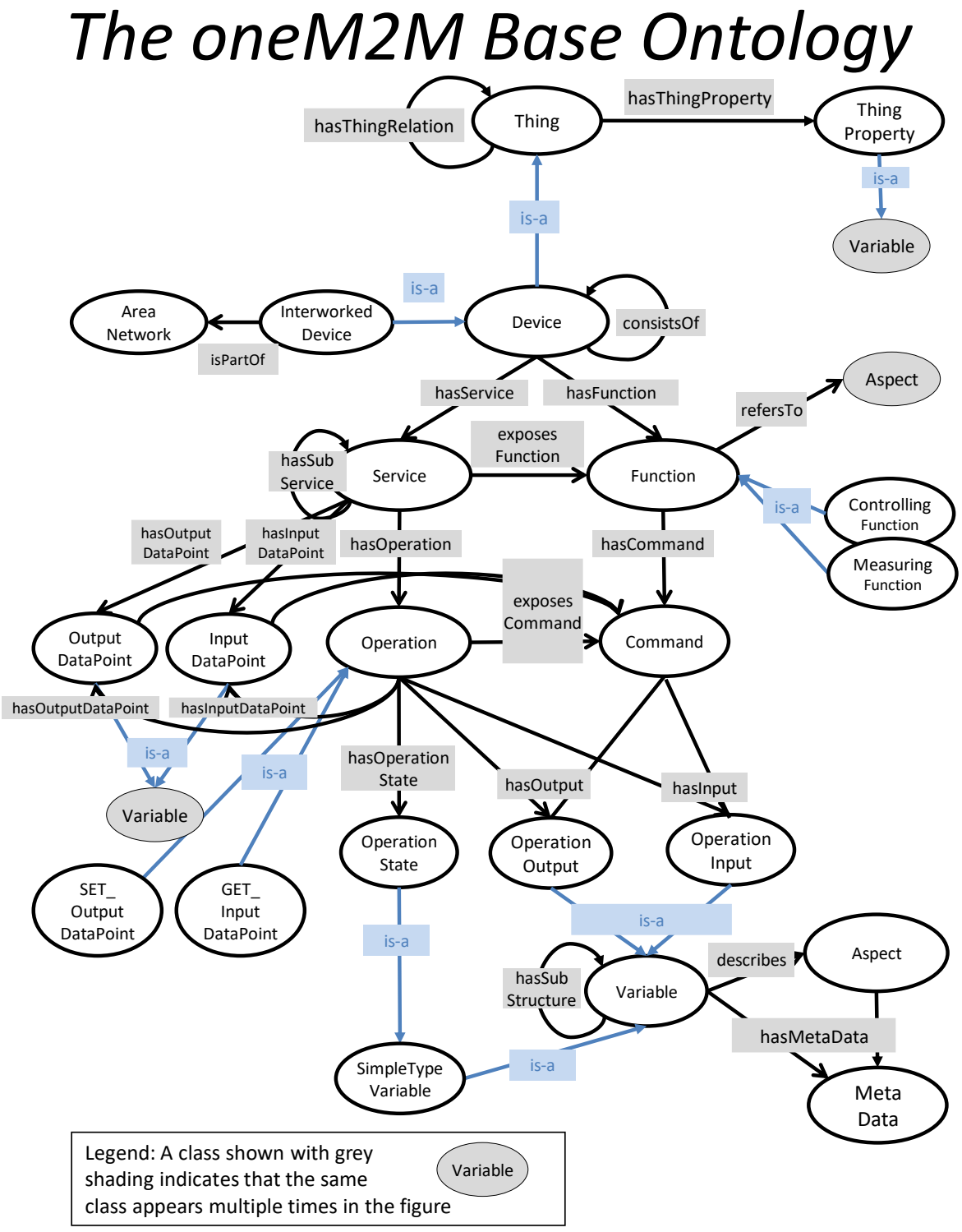

Figure 1: The oneM2M Base Ontology

Figure 1 shows the essential Classes and Properties of the Base Ontology. The nodes (bubbles) denote Classes whereas edges (arrows) denote Object Properties.

The graph in figure 1 can be read as follows:

- A **Thing** in oneM2M (Class: Thing) is an entity that can be identified in the oneM2M System.
  A Thing may have properties (Object Property: hasThingProperty).
  A Thing can have relations to other things (Object Property: hasThingRelation).
  E.g. A room that is modelled in oneM2M would be a Thing that could have a room-temperature as a ThingProperty (via hasThingProperty) and could have a hasThingRelation "isAdjacentTo" to another room.
  In general it is assumed that a Thing is not able to communicate electronically with its environment. However, the sub-class of Thing that ***is*** able to interact electronically is called a "Device".

- A **ThingProperty** (Class: ThingProperty) denotes a property of a Thing. A Thing can be described with (the values of) ThingProperties, but in general the Thing cannot influence that value or being influenced by it. A human or a computer or a device could set the value of a Thing's ThingProperty and possibly read it. A ThingProperty be can be retrieved or updated by an entity of the oneM2M System.
  E.g. the indoor temperature of the room could be a Value of a Thing "room", or the manufacturer could be a ThingProperty of a Thing "car".
  A ThingProperty of a thing can describe a certain Aspect, e.g. the indoor temperature describes the Aspect "Temperature" that could be measured by a temperature sensor.
  A ThingProperty of a Thing can have meta data

- **Variable** (Class: Variable) constitutes a super class to the following classes: ThingProperty, OperationInput, OperationOutput, OperationState, InputDataPoint, OutputDataPoint. Its members are entities that have some data (e.g. integers, text, etc., or structured data) that can change over time. These data of the Variable usually describe some real-world Aspects (e.g. a temperature) and can have MetaData (e.g. units, precision).
  A Variable can be structured, i.e. it can consist of (sub-) Variables.

- One sub-class is defined in the base ontology:

  - **SimpleTypeVariable** (Class: SimpleTypeVariable) is a sub-class of Variable that only consists of Variables of simple xml types like xsd:integer, xsd:string…, potentially including restrictions.

  - **MetaData** (Class: MetaData) contain data (like units, precision-ranges, etc.) about the Values of a Thing or about an Aspect.
    E.g. the indoor temperature could have meta data: "Degrees Celsius".

  - A **Device** (Class: Device) is a Thing (a sub-class of class:Thing) that is designed to accomplish a particular task via the Functions the Device performs.
    A Device can be able to interact electronically with its environment via a network.
    A Device contains some logic and is producer and/or consumer of data that are exchanged via its Services with other entities (Devices, Things) in the network. A Device interacts through the DataPoints and/or Operations of its Services.
    In the context of oneM2M a Device is always assumed to be capable of communicating electronically via a network (oneM2M or interworked non-oneM2M network):

    - In order to accomplish its task, the device performs one or more Functions (Object Property: hasFunction) (Class: Function).
      These Functions are exposed in the network as Services of the Device.

    - A Device can be composed of several (sub-) Devices (Object Property: consistsOf) (Class: Device).
      => consistsOf only Device.

    - Each Device (including sub-Devices) needs to be individually addressable in the network.

    E.g. a "lightswitch" would be a device, a combined fridge/freezer would be a device that consists of a sub-device fridge and a sub-device freezer.

  - A **Function** (Class: Function) represents the Function necessary to accomplish the task for which a Device is designed. A device can be designed to perform more than one Function.
    The Class: Function exhibits the - human understandable - meaning what the device "*does*":

  - A Function refers to (e.g. observes or influences) a certain Aspect.

    E.g. considering a "light switch" then a related Function could be "Controlling_ON_OFF".
    These Functions would refer to an Aspect "lighting", that is influenced by the device "light switch".

- Two sub-classes of class Function are defined in the base ontology:

  - **ControllingFunction** (Class: ControllingFunction) is a sub-class of Function that only controls/influences real world Aspects that the Function relates to.

  - **MeasuringFunction** (Class: MeasuringFunction) is a sub-class of Function that only measures/senses real world Aspects that the Function relates to.

- An **Aspect** (Class: Aspect) describes the real-world aspect that a Function relates to. Aspect is also used to describe a quality or kind of OperationInput- or OperationOutput variables. The Aspect could be a (physical or non-physical) entity or it could be a quality.

- A **Command** (Class: Command) represents an action that can be performed to support the Function. An Operation exposes a Command to the network. OperationInput and OperationOutput of the related Operation can parameterize the command.
  e.g. the Function "Dimming-Function" could have a Command "setPercentage", with a parameter that has values 0 - 100.

- A **Service** (Class: Service) is a representation of a Function to a network that makes the Function discoverable, registerable, remotely controllable in the network. A Service can represent one or more Functions. A Service is offered by a device that wants (a certain set of) its Functions to be discoverable, registerable, remotely controllable by other devices in the network:

  - While a Function describes the meaning of the device's Function the Service (Class: Service) is used to describe how such Function is represented in a communication network and is therefore dependent on the technology of the network.

  E.g. the Function: "turn_light_On_or_Off" could be exposed in the network by a Service "Binary Value Actuator".

  - A Service may be composed of smaller, independent (sub)Services, e.g. re-usable servicemodules.

- An **OutputDataPoint** (class: OutputDataPoint) is a Variable of a Service that is set by a RESTful Device in its environment and that provides state information about the Service. The Device updates the OutputDataPoint autonomously (e.g. at periodic times). To enable a third party to retrieve the current value of a OutputDataPoint (out of schedule) devices often also offer a SET_OutputDataPoint Operation to trigger the device to update the data of the OutputDataPoint.

- An **InputDataPoint** (class: InputDataPoint) is a Variable of a Service that is set by a RESTful Device in its environment and that the Device readsout autonomously (e.g. at periodic times). To enable a third party to instruct the device to retrieve (out of schedule) the current value of a InputputDataPoint devices often also offer a GET_InputDataPoint Operation to trigger the device to retrieve the data from the InputDataPoint.

NOTE 1: Input- and Output DataPoints are usually used by Devices (AEs) that communicate in a RESTful way, while Operations are the procedures that are used for remote procedure based communication. Operations are, however, also needed in RESTful systems to correlate output, that is produced by a device, to the input that triggered the production of that output.

- An **Operation** (Class: Operation) is the means of a Service to communicate in a procedure-type manner over the network (i.e. transmit data to/from other devices).
  An Operation is a representation of a Command to a network:

  - An Operation can have OperationInput (data consumed by the Device) and OperationOutput (Data produced by the Device), as well as a Method that describes how the Operation is invoked over the network.

  - An Operation shall have a Data Property "OperationState" that indicates how the operation has progressed in the device.

  - An Operation is transient. I.e. an Operation can be invoked, possibly produces output and is finished.

  - An Operation correlates the output data of the Operation to the input data that were used at Operation invocation.

- Two sub-classes of class Operation are defined in the base ontology:

  - **GET_InputDataPoint** (Class: GET_InputDataPoint) is a sub-class of Operation that may be offered by a Device to trigger the device to retrieve the data of an InputDataPoint.
    (e.g. outside of the schedule when the device normally retrieves that DataPoint).

  - **SET_OutputDataPoint** (Class: SET_OutputDataPoint) is a sub-class of Operation that may be offered by a Device to trigger the device to update the data of an OutputDataPoint.
    (e.g. outside of the schedule when the device normally updates that DataPoint).

- **OperationInput** (Class: OperationInput) describes the type of input of an Operation to a service of the device. The OperationInput class represents all possible values for that input (data types and -ranges or a list of enumerated individuals). An Operation can have multiple OperationInputs and/or OperationOutputs. If an instance of an Operation is executed then the input value to that Operation is an instance of its OperationInput classes (e.g. enumerated instances like "ON" or "OFF" for an OperationInput class that sets the state of a switch or a real number within a certain range for a "Temperature" OperationInput class for a thermostat).

- **OperationOutput** (Class: OperationOutput) describes the type of output of an Operation from a service of the device. The OperationOutput class represents all possible values for that OperationOutput (data types and ranges or a list of enumerated individuals). An Operation can have multiple OperationInputs and/or OperationOutputs. If an instance of an Operation is executed then the output values of that Operation are instances of its OperationOutput classes.

- **OperationState** (Class: OperationState) describes the current state during the lifetime of an Operation. The OperationState class represents all possible values for that state (enumerated individuals). The OperationState is set during the progress of the operation by the entity invoking the operation, the entity that is the target of the operation, e.g. a device (or for interworked devices by the IPE) and the CSE. It takes values like "data_received_by_application", "operation_ended", "operation_failed", "data_transmitted_to_interworked_device".

- **Area Network** (Class: AreaNetwork):

  - An Area Network (see [1]) is characterized by its technology:

    - physical properties (e.g. IEEE_802_15_4_2003_2_4GHz); its

    - communication protocol (e.g. ZigBee_1_0); and

    - potentially a profile (e.g. ZigBee_HA).

- **Interworked Device** (Class: InterworkedDevice):

  - Is part of an AreaNetwork.

NOTE 2: An Interworked Device is not a oneM2M Device and can be only accessed from the oneM2M System by communicating with a "proxied" (virtual) device that has been created by an Interworking Proxy Entity (IPE).
The InterworkedDevice class describes the "proxied" (virtual) device that is represented in the oneM2M System as an individual <AE> resource or a child resource of the <AE> of its IPE.

5.2.2 Use of ontologies for Generic interworking with Area Networks

5.2.2.1 General Principle

Interworking with Area Networks is accomplished in oneM2M through Function provided by Interworking Proxy Entities (IPE).

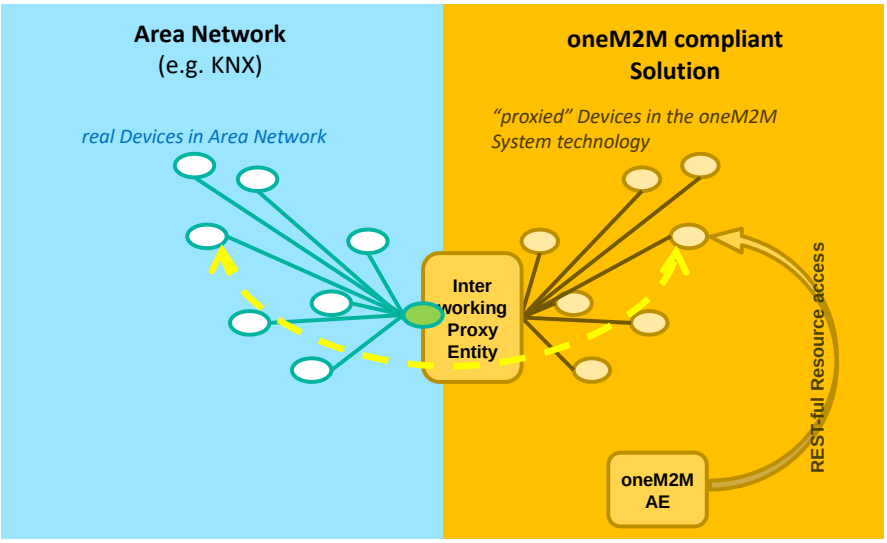

Figure 2: Interworking

The IPE creates "proxied" devices as oneM2M Resources (e.g. AEs) in the oneM2M Solution that can be accessed by oneM2M Applications in the usual way.

To accomplish the creation of "proxied" devices the IPE uses an ontology that describes the type of interworked Area Network and its entities (device types, their operations, etc.).
For example, in figure 2, an ontology that describes a KNX Area Network and its entities would be needed.

To achieve the flexibility for the IPE to create "proxied" Devices for many different types of Area Networks each ontology that describes a specific type of interworked Area Network needs to be derived from the Base Ontology that is specified in the present document.
E.g. the OWL representation of an ontology that describes the entities of an Area Network of type "KNX" needs to:

a) contain an 'include' statement which includes Base Ontology;

b) the Class of "KNX Nodes" needs to be a subclass of the "Device" Class of oneM2M's Base Ontology;

c) the Class of "KNX Communication Objects" needs to be a subclass of the "Service" Class of the Base Ontology;

d) etc.

NOTE: For the purpose of Generic interworking with Area Networks the Base Ontology is only used to describe type information and not for describing instances of these types. E.g. the Base Ontology describes the type "Device", but does not contain information about a specific Device.
The Base Ontology therefore only contains Classes and Properties but not instances.

6 Description of Classes and Properties

6.1 Classes

6.1.1 Class: Thing

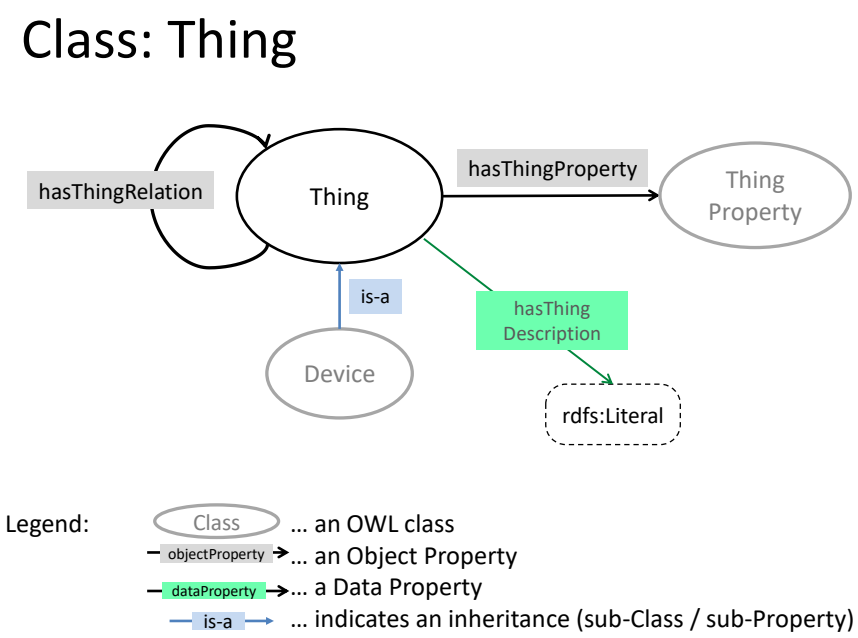

Figure 3: Thing

Description

- A **Thing** in oneM2M (Class: Thing) is an entity that can be identified in the oneM2M System.
  A Thing that is not a Device is not able to communicate electronically with its environment. However, the sub-class of Thing that ***is*** able to interact electronically is called a "Device".
  A Thing may have ThingProperties (Object Property: hasThingProperty). A Thing can have relations to other things (Object Property: hasThingRelation).
  Since a Thing that is not a Device is not able to communicate electronically it cannot influence the value of its ThingProperties or being influenced by it. Similarly a Thing cannot document its - real-world - relationships (via hasThingRelation) to other Things.

- E.g. A room that is modelled in oneM2M would be a Thing that could have a room-temperature as a ThingProperty and could have a relationship "isAdjacentTo" to another room.

Object Properties

This Class is the domain Class of Object Property:

- hasThingProperty (range Class: ThingProperty)
- hasThingRelation (range Class: Thing)

This Class is the range Class of Object Property:

- hasThingRelation (domain Class: Thing)

Data Properties

- hasThingAnnotation (range datatype: rdfs:Literal)

Superclass-subclass Relationships

This Class is subclass of:

- none

This Class is superclass of:

- device

Restrictions

This Class is anonymous sub-class of:

- hasThingRelation only Thing
  (Universal restriction: a Thing can *only* have a relationship "hasThingRelation" to other Things)
- hasThingProperty only ThingProperty

6.1.2 Class: ThingProperty

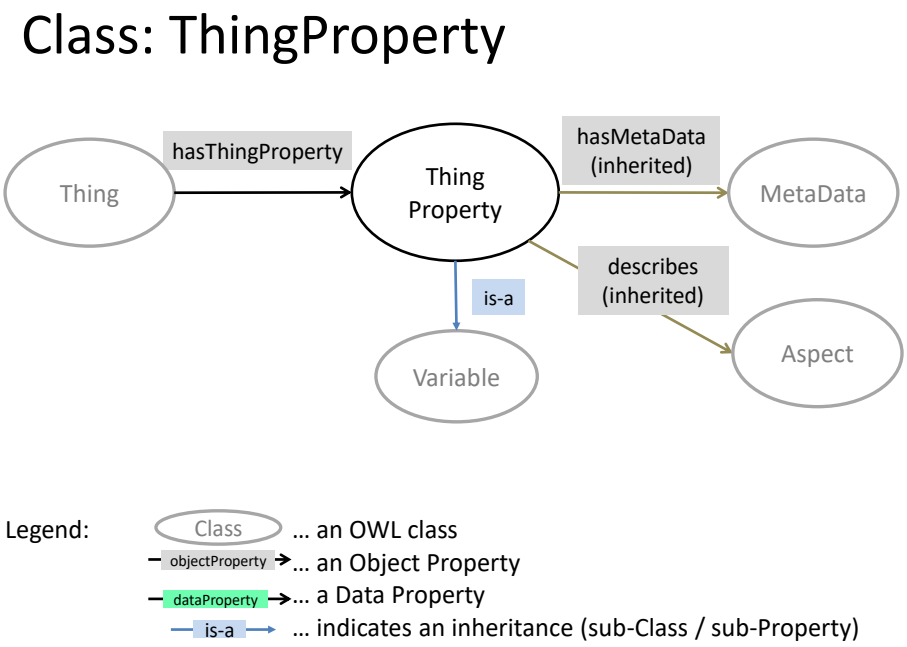

Figure 4: ThingProperty

Description

- A **ThingProperty** (Class: ThingProperty) denotes a property of a Thing. A ThingProperty can e.g. be observed or influenced by devices, or it constitutes static data about a Thing.
  E.g. the indoor temperature of the room could be a ThingProperty of a Thing "room".
  A ThingProperty of a thing can describe a certain Aspect, e.g. the indoor temperature describes the Aspect "Temperature" that could be measured by a temperature sensor.
  A ThingProperty of a Thing can have meta data.
- The class ThingProperty is a sub-class of the Variable class.

Object Properties

This Class is the domain Class of Object Property:

- describes (range Class: Aspect)
  (inherited from class: Variable)

- hasMetaData (range Class: MetaData)
  (inherited from class: Variable)

This Class is the range Class of Object Property:

- hasThingProperty (domain Class: Thing)

Data Properties

This Class is part of the domain Class of Data Property:

- Inherited from class: Variable and possibly Class SimpleTypeVariable
  (see clauses 6.1.17 and 6.1.18)

Superclass-subclass Relationships

This Class is sub-class of:

- Variable

This Class is super-class of:

- None

Restrictions

- None

6.1.3 Class: Aspect

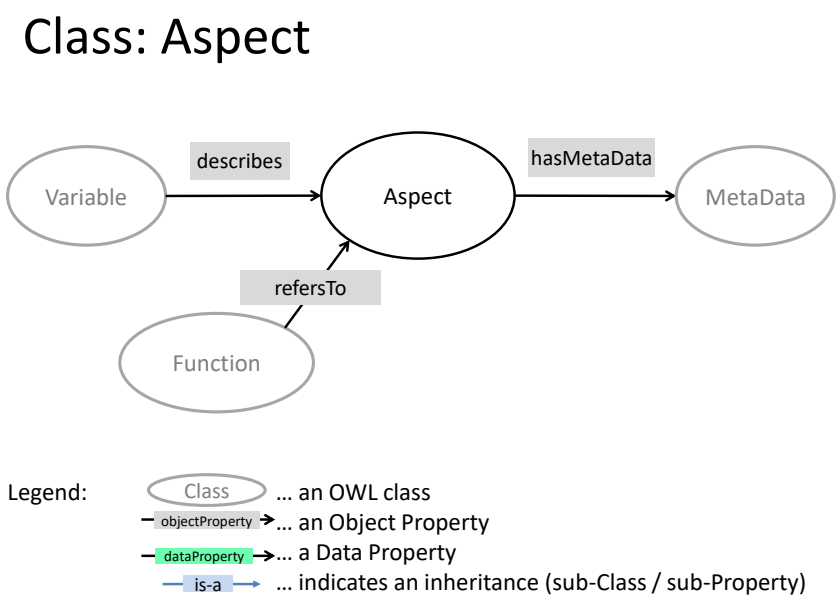

Figure 5: Aspect

Description

- An **Aspect** (Class: Aspect) describes the real-world aspect that a Function relates to. Aspect is also used to describe the quality or kind of a Variable.
  The Aspect could be a (physical or non-physical) entity or it could be a quality.

Object Properties

This Class is the domain Class of Object Property:

- hasMetaData (range Class: MetaData)

This Class is the range Class of Object Property:

- refersTo (domain Class: Function)
- describes (domain Class: Variable)

Data Properties

- none

Superclass-subclass Relationships

This Class is sub-class of:

- none

This Class is super-class of:

- none

Restrictions

- none

6.1.4 Class: MetaData

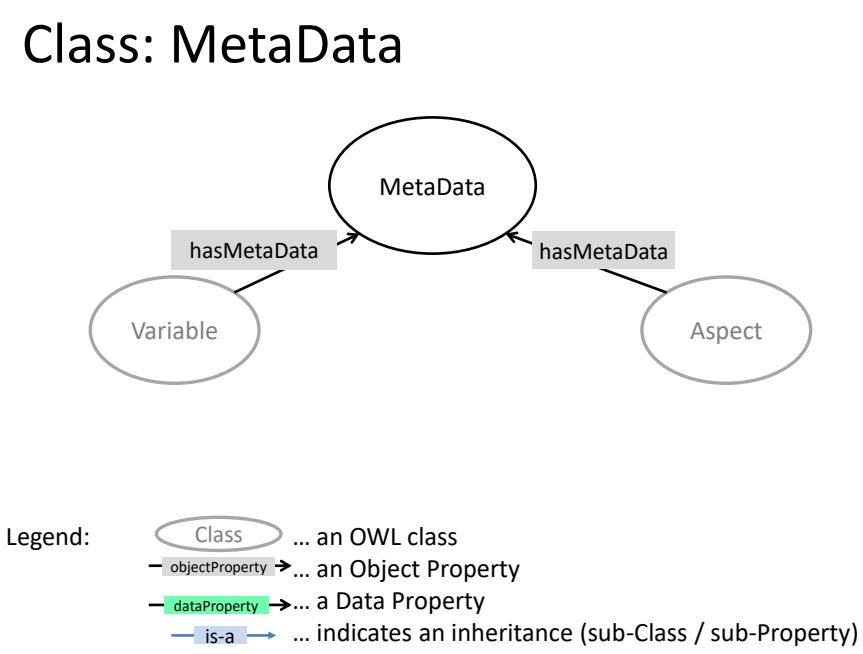

Figure 6: MetaData

Description

- **MetaData** (Class: MetaData) contain data (like units, precision-ranges …) about a Variable or about an Aspect.
  E.g. the indoor temperature could have as meta data an individual "Celsius_Scale" that specifies that the temperature needs to be understood as degrees Celsius.

Object Properties

This Class is the domain Class of Object Property:

- none

This Class is the range Class of Object Property:

- hasMetaData (domain Class: Variable)
- hasMetaData (domain Class: Aspect)

Data Properties

- none

Superclass-subclass Relationships

This Class is sub-class of:

- none

This Class is super-class of:

- none

Restrictions

- none

6.1.5 Class: Device

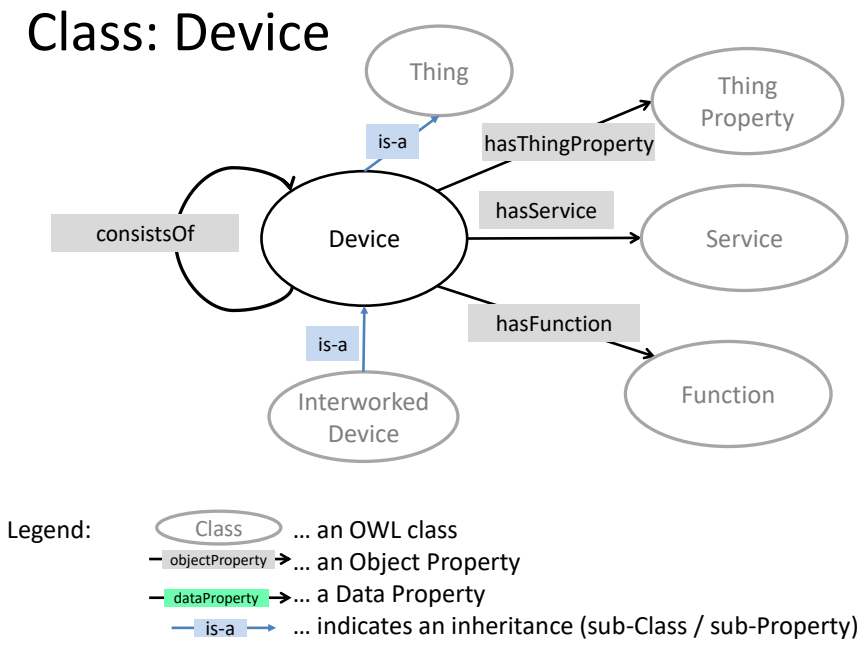

Figure 7: Device

Description

- A **Device** (Class: Device) is a Thing (a sub-class of class:Thing) that is designed to accomplish a particular task via the Functions the Device performs.
  A Device can be able to interact electronically with its environment via a network. A Device contains some logic and is producer and/or consumer of data that are exchanged via its Services with other oneM2M entities (Devices, Things) in the network. A Device may be a physical or non-physical entity.
  A Device interacts through the DataPoints and/or Operations of its Services:
  - In order to accomplish its task, the device performs one or more Functions.
  - These Functions are exposed in the network as Services of the Device.
  - A Device can be composed of several (sub-) Devices.

- Each Device (including sub-Devices) needs to be individually addressable in the network.

Object Properties

This Class is the domain Class of Object Property:

- consistsOf (range Class: Device)
- hasService (range Class: Service)
- hasFunction (range Class: Function)
- hasThingProperty (range Class: ThingProperty)
  (inherited from Class:Thing)

This Class is the range Class of Object Property:

- consistsOf (domain Class: Device)

Data Properties

- none

Superclass-subclass Relationships

This Class is sub-class of:

- Thing

This Class is super-class of:

- InterworkedDevice

Restrictions

This Class is anonymous sub-class of:

- consistsOf only Device
  (Universal restriction: a Device can have a relationship "consistsOf" *only* to other Devices)
- hasFunction min 1 Function
  (Universal restriction: a Device needs to have a relationship " hasFunction " to at least 1 Function)
- hasService only Service
  (Universal restriction: a Device can have a relationship " hasService " *only* to Services)

6.1.6 Class: InterworkedDevice

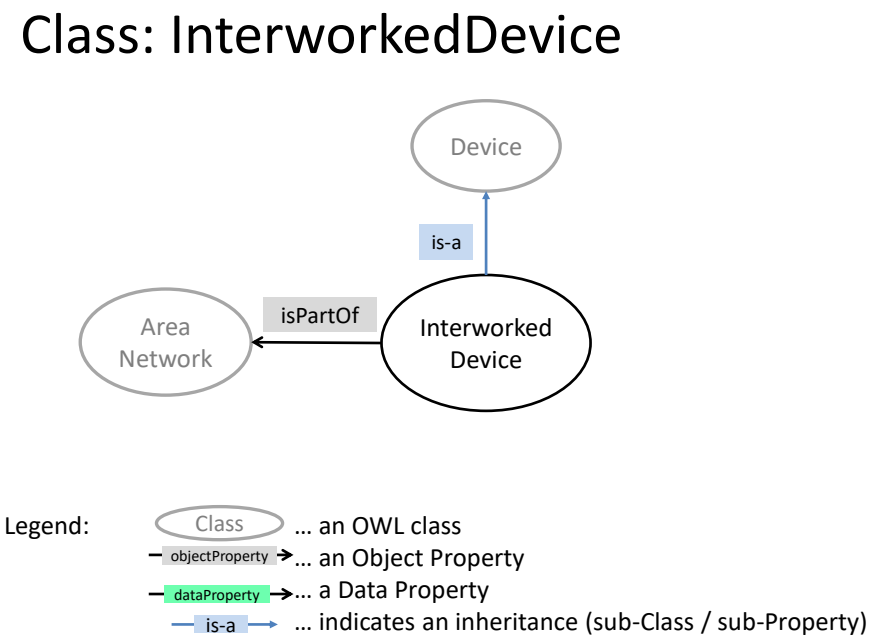

Figure 8: InterworkedDevice

Description

- An **InterworkedDevice** (Class: InterworkedDevice) is a Device - e.g. in an Area Network - that does not support oneM2M interfaces and can only be accessed from the oneM2M System by communicating with a "proxied" (virtual) device that has been created by an Interworking Proxy Entity.

Object Properties

This Class is the domain Class of Object Property:

- isPartOf (range Class: AreaNetwork)

This Class is the range Class of Object Property:

- none

Data Properties

- none

Superclass-subclass Relationships

This Class is sub-class of:

- Device

This Class is super-class of:

- none

Restrictions

- none

6.1.7 Class: AreaNetwork

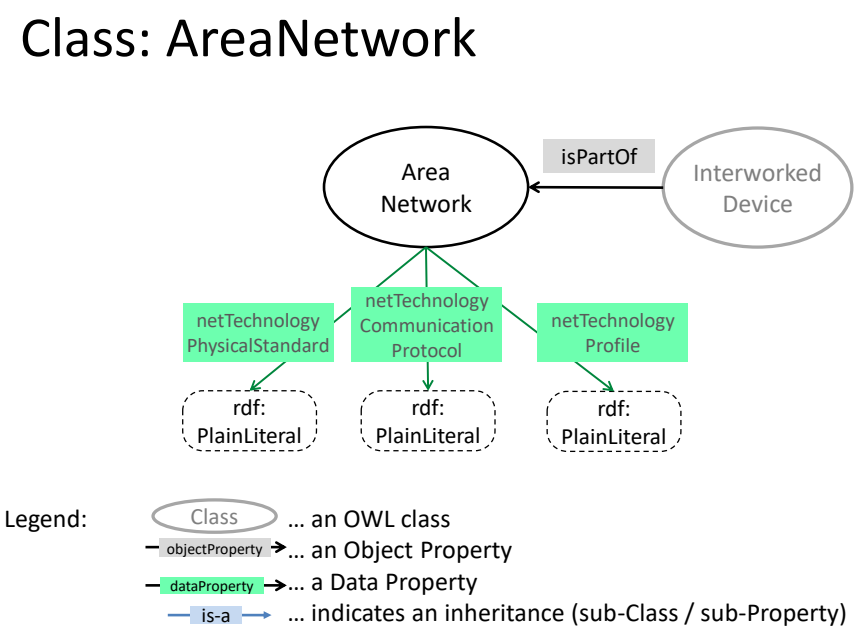

Figure 9: AreaNetwork

Description

- An **AreaNetwork** (Class: AreaNetwork) is a Network that provides data transport services between an Interworked Device and the oneM2M System. Different area Networks can use heterogeneous network technologies that may or may not support IP access.

Object Properties

This Class is the domain Class of Object Property:

- none

This Class is the range Class of Object Property:

- isPartOf (domain Class: InterworkedDevice)

Data Properties

- netTechnologyPhysicalStandard (range datatype: rdf:PlainLiteral) which serves for Identification of the physical properties of an Area Network technology (e.g. IEEE_802_15_4_2003_2_4GHz)
- netTechnologyCommunicationProtocol (range datatype: rdf:PlainLiteral) which serves for Identification of a communication protocol (e.g. ZigBee_1_0)
- netTechnologyProfile (range datatype: rdf:PlainLiteral) which serves for Identification of a profile (e.g. ZigBee_HA) of an Area Network technology

Superclass-subclass Relationships

This Class is sub-class of:

- none

This Class is super-class of:

- none

Restrictions

- none

6.1.8 Class: Service

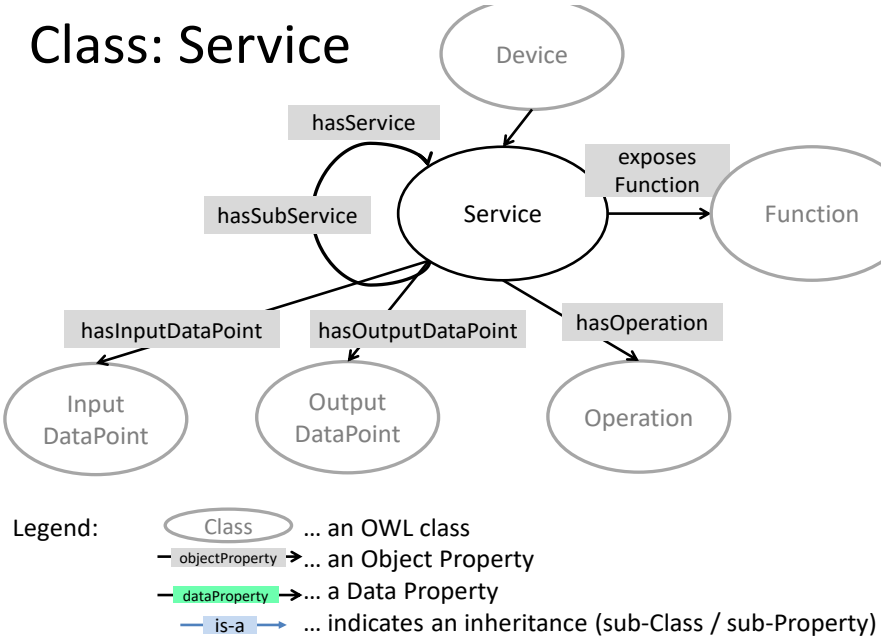

Figure 10: Service

Description

- A **Service** (Class: Service) is an electronic representation of a Function in a network. The Service exposes the Function to the network and makes it discoverable, registerable and remotely controllable in the network.
  A Service is offered by a device that wants (a certain set of) its Functions to be discoverable, registerable, remotely controllable by other devices in the network.
  A Service can expose one or more Functions and a Function can be exposed by one or more Services.

- The Input- and Output DataPoints and Operations of a Service may have the same names as for a different Service, however the Service to which they belong differentiates how they are addressed in the Device (e.g. via a port specific to the Service).

NOTE: While a Function describes the - human understandable - meaning of a Service of the device the Service is used to describe how such Function is represented in a communication network and can be accessed by electronic means. The Service and its Operations is therefore dependent on the technology of the network, hard- and software of the device.

- E.g. the Function: "turn_light_On_or_Off" could be exposed in the network by a Service "UPDATE Binary Value".

  - Object Property "hasSubService" is expresses the fact that Services can be composed of independent (sub)Services.
    E.g. a Service could thus be composed out of multiple (reusable) service modules. A Dimmer could contain a module "binaryActuator" to turn on/off and additionally "setInteger0-255Actuator" to set the dimming level.

Object Properties

This Class is the domain Class of Object Property:

- exposesFunction (range Class: Service)

- hasOperation (range Class: Operation)

- hasInputDataPoint (range Class: InputDataPoint)
- hasOutputDataPoint (range Class: OutputDataPoint)
- hasSubService (range Class: Service)

If for a Service an Operation, Input-, OutputDataPoint or sub-Service is mandatory then the related Object Property (hasOperation, hasInputDataPoint, hasOutputDataPoint, hasSubService) shall have a Property Restriction with cardinality "min 1".

This Class is the range Class of Object Property:

- hasService (domain Class: Device)
- hasSubService (domain Class: Service)

Data Properties

- none

Superclass-subclass Relationships

This Class is sub-class of:

- none

This Class is super -class of:

- none

Restrictions

- hasSubService only Service
  (Universal restriction: a Service can have a relationship "hasSubService" *only* to other Service)
- hasOperation only Operation
- exposesFunction some Function
  (Universal restriction: at least one of the relationship " exposesFunction "of a Service needs to point of a Function)
- hasInputDataPoint only InputDataPoint
- hasOutputDataPoint only OutputDataPoint

6.1.9 Class: Function

6.1.9.0 General description

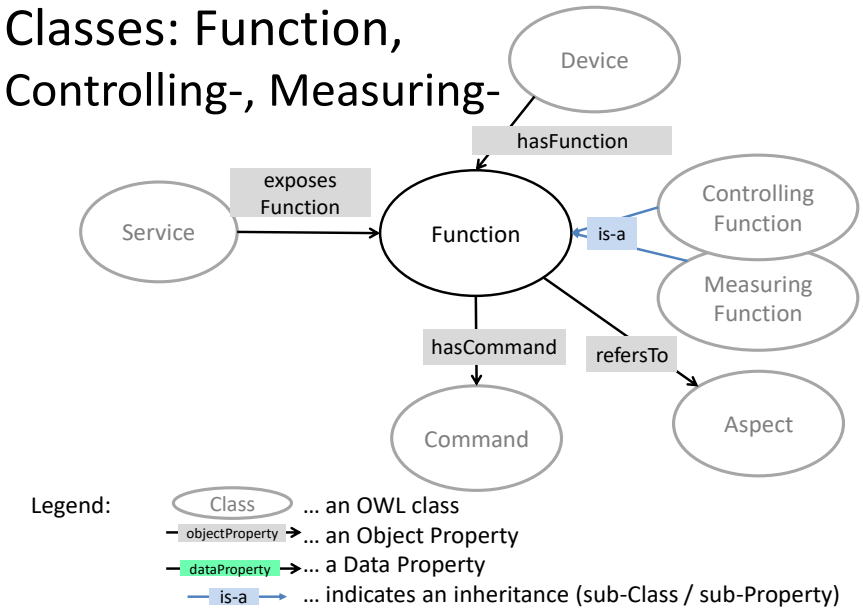

Figure 11: Function

Description

- A **Function** (Class: Function) represents a particular function necessary to accomplish the task for which a Device is designed. A device can be designed to perform more than one Function.
  The Function exhibits the - human understandable - meaning what the device "*does*".
- A Function refers to (e.g. observes or influences) some real-world aspect(s), that can be modelled as a Class: Aspect.
- E.g. considering a "light switch" then a related Function could be "Controlling_ON_OFF" or "Controlling Brightness". These Functions would refer to an Aspect "light-control".
- A Function of a Device can be influenced/observed by a human user through the Commands that this Function has and that are offered to the user.

Object Properties

This Class is the domain Class of Object Property:

- hasCommand (range Class: Command)
- refersTo (range Class: Aspect)

This Class is the range Class of Object Property:

- exposesFunction (domain Class: Service)
- hasFunction (domain Class: Device)

Data Properties

- none

Superclass-subclass Relationships

This Class is sub-class of:

- none

This Class is super-class of:

- ControllingFunction
- MeasuringFunction

Restrictions

- none

6.1.9.1 Class: ControllingFunction

Description

- A **ControllingFunction** (Class: ControllingFunction) represents a Function that has impacts on the real world, but does not gather data. In general a ControllingFunction has Commands (and/or Operations of its related Services) that receive input data.
- E.g. a thermostat would have "temperature-adjustment" as a ControllingFunction.

Object Properties

This Class is the domain Class of Object Property:

- none

This Class is the range Class of Object Property:

- none

Data Properties

- none

Superclass-subclass Relationships

This Class is sub-class of:

- Function

This Class is super-class of:

- none

Restrictions

- none

6.1.9.2 Class: MeasuringFunction

Description

- A **MeasuringFunction** (Class: MeasuringFunction) represents a Function that has no impacts on the real world, but only gathers data. In general a MeasuringFunction has Commands (and/or Operations of its related Services) that generate output data.
- E.g. a temperature sensor would have "temperature-sensing" as a MeasuringFunction.

Object Properties

This Class is the domain Class of Object Property:

- none

This Class is the range Class of Object Property:

- none

Data Properties

- none

Superclass-subclass Relationships

This Class is sub-class of:

- Function

This Class is super-class of:

- none

Restrictions

- none

6.1.10 Class: Operation

6.1.10.0 General description

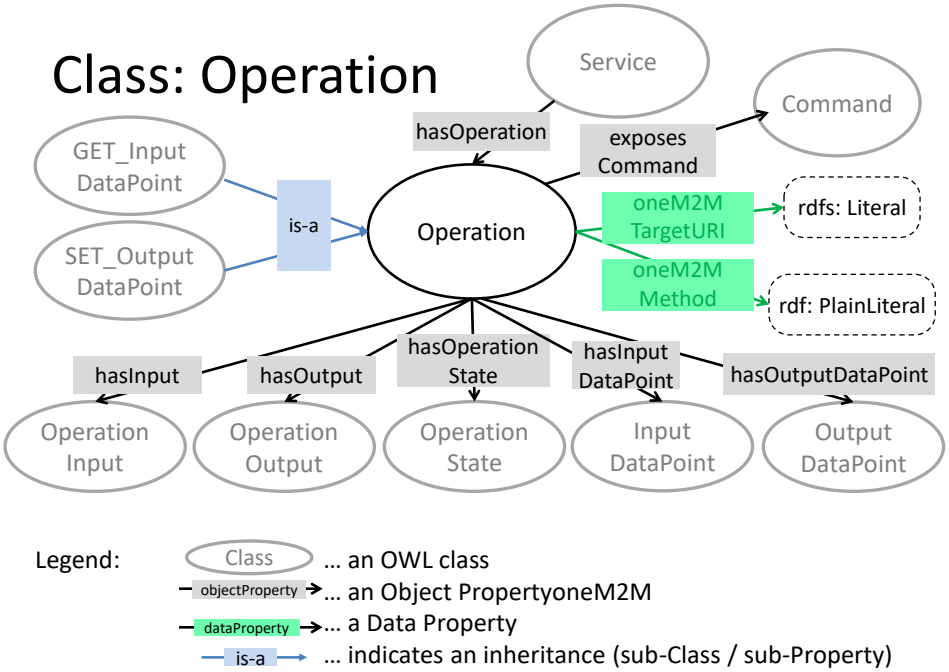

Figure 12: Operation

Description

- An **Operation** (Class: Operation) is the means of a Service to communicate in a procedure-type manner over the network (i.e. transmit data to/from other devices). It is the -machine interpretable- exposure of a -human

understandable- Command to a network.
An Operation is transient. I.e. an Operation can be invoked, possibly produces output and is finished:

- A non-oneM2M Device or a oneM2M entity (e.g. an AE) can invoke an Operation of the Device (oneM2M Device or InterworkedDevice) and that invocation can trigger some action in the Device. If an Operation has input data it may receive input data from:
  - InputDataPoints (persistent entities); and/or
  - OperationInput (transient entities, that are deleted when the Operation finishes);

  and potentially produce output data into:
  - OutputDataPoints (persistent entities) and/or
  - OperationOutput (transient entities, that are deleted when the Operation finishes)
- An Operation correlates the output data of the Operation to the input data that were used at Operation invocation.
- An Operation has an OperationState that allows a oneM2M entity to get informed on the progress of that operation.

NOTE: The OperationState - which provides information of an ongoing or finished operation - should not be confused with state information about the Device or Service, which potentially could be obtained as output data of some operation.

Object Properties

This Class is the domain Class of Object Property:

- exposesCommand (range Class: Command)
- hasInput (range Class: OperationInput)
- hasInputDataPoint (range Class: InputDataPoint)
- hasOutput (range Class: OperationOutput)
- hasOutputDataPoint (range Class: OutputDataPoint)
- hasOperationState (range Class: OperationState)

If for an Operation an OperationInput, OperationOutput, Input-, or OutputDataPoint is mandatory then the related Object Property (hasInput, hasOutput, hasInputDataPoint, hasOutputDataPoint) shall have a Property Restriction with cardinality "min 1".

If for an Operation the OperationState is mandatory then the related Object Property (hasOperationState) shall have a Property Restriction with cardinality "exactly 1".

This Class is the range Class of Object Property:

- hasOperation (range Class: Service)

Data Properties

- oneM2MMethod (range data type: rdf:PlainLiteral)
- oneM2MTargetURI (range data type: rdfs: Literal)

Superclass-subclass Relationships

This Class is sub-class of:

- none

This Class is super-class of:

- GET_InputDataPoint
- SET_OutputDataPoint

Restrictions

- exposesCommand only Command
- hasInput only OperationInput
- hasOperationState only OperationState
- hasOutput only OperationOutput

6.1.10.1 Class: GET_InputDataPoint

Description

- **GET_InputDataPoint** (Class: GET_InputDataPoint) is an Operation that may be offered by a Device to trigger the device to retrieve the data of an InputDataPoint
  (e.g. outside of the schedule when the device normally retrieves data from that DataPoint)

Object Properties

This Class is the domain Class of Object Property:

- hasInputDataPoint (range Class: InputDataPoint)

This Class is the range Class of Object Property:

- none

Data Properties

- none

Superclass-subclass Relationships

This Class is sub-class of:

- Operation

This Class is super-class of:

- none

Restrictions

- none

6.1.10.2 Class: SET_OutputDataPoint

Description

- **SET_OutputDataPoint** (Class: SET_OutputDataPoint) is an Operation that may be offered by a Device to trigger the device to update the data of an OutputDataPoint
  (e.g. outside of the schedule when the device normally updates that DataPoint)

Object Properties

This Class is the domain Class of Object Property:

- hasOutputDataPoint (range Class: OutputDataPoint)

This Class is the range Class of Object Property:

- none

Data Properties

- none

Superclass-subclass Relationships

This Class is sub-class of:

- Operation

This Class is super-class of:

- none

Restrictions

- none

6.1.11 Class: Command

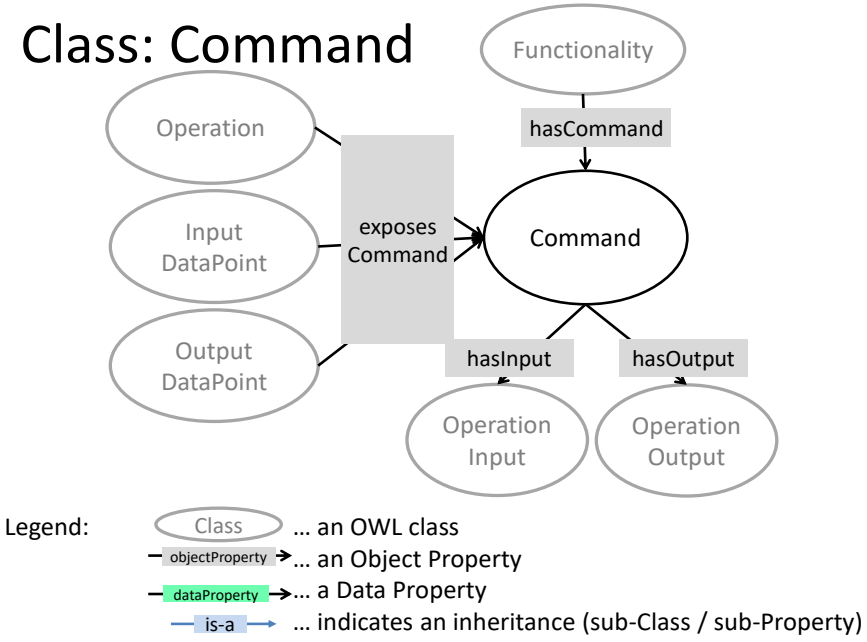

Figure 13: Command

Description

- A **Command** (Class: Command) represents an action that can be performed to support the Function. A Command is the -human understandable - name of that action that is invoked in a device or is reported by the device. An Operation exposes a Command to the network. OperationInput and OperationOutput of the related Operation can parameterize the command.
  e.g. the Function "dimming-Function" of a light switch that remotely controls a light could have a Command "setLightIntensity", with a parameter that has values 0 - 100 %.

  Also InputDataPoints and OutputDataPoints expose Commands to the network. When a Device communicates in a RESTful way then changing (UPDATEing) an InputDataPoint triggers an action in the Device once the Device has read out the data from the InputDataPoint.
  Similarly, when a Device sets the data of an OutputDataPoint then it provides state information about the Device.

NOTE: In RESTful systems the names of Input- and OutputDataPoints are usually chosen in such a way that they express the Command, i.e. the human-understandable meaning (e.g. a binary InputDataPoint of a lightswitch could have a name "Set_Light_Status"). Updating a DataPoint can be interpreted as executing a Command.

Object Properties

This Class is the domain Class of Object Property:

- isExposedByOperation (range Class: Operation)
- hasInput (range Class: OperationInput)
- hasOutput (range Class: OperationOutput)

This Class is the range Class of Object Property:

- hasCommand (domain Class: Function)
- exposesCommand (domain Class: Operation OR InputDataPoint OR OutputDataPoint)

Data Properties

- none

Superclass-subclass Relationships

This Class is sub-class of:

- none

This Class is super-class of:

- none

Restrictions

- hasInput only OperationInput
  (Universal restriction: a Command can have a relationship " hasInput " *only* to OperationInput)
- hasOutput only OperationOutput
- hasInputDataPoint only InputDataPoint
- hasOutputDataPoint only OutputDataPoint

6.1.12 Class: OperationInput

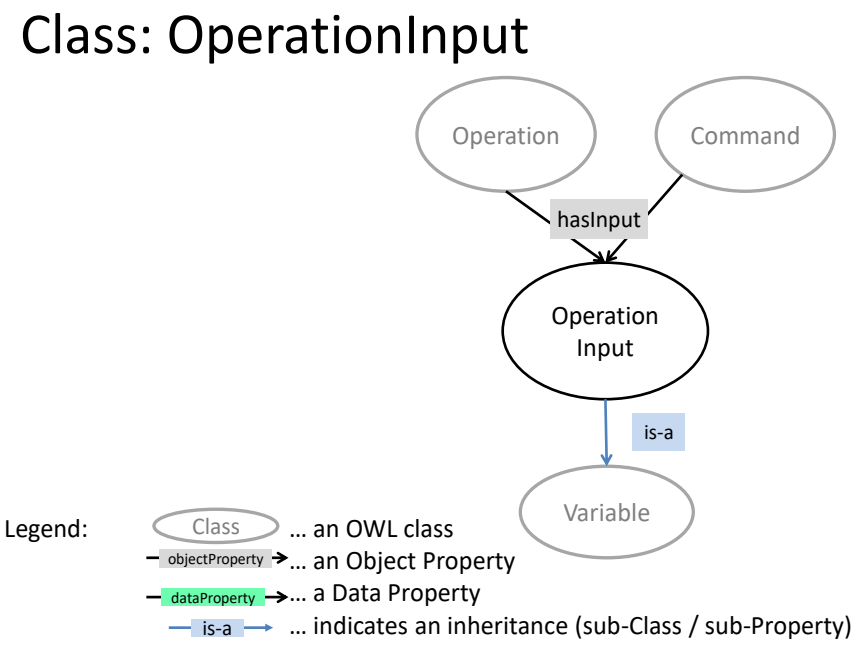

Figure 14: OperationInput

Description

- **OperationInput** (Class: OperationInput) describes an input of an Operation of a Service. OperationInput also describes the input of a Command:
  - OperationInput is transient. An instance of OperationInput is deleted when the instance of its Operation is deleted.
  - An Operation/Command may have multiple OperationInputs and/or OperationOutputs. If an instance of an Operation is invoked then the input value to that Operation shall be an instance of its OperationInput class.

Object Properties

This Class is the domain Class of Object Property:

- hasMetaData (range Class: MetaData)
  (inherited from class:Variable)
- describes (range Class: Aspect)
  (inherited from class:Variable)

This Class is the range Class of Object Property:

- hasInput (domain Class: Operation)
- hasInput (domain Class: Command)

Data Properties

This Class is part of the domain Class of Data Property:

- Inherited from class: Variable and possibly Class SimpleTypeVariable
  (see clauses 6.1.17 and 6.1.18)

Superclass-subclass Relationships

This Class is sub-class of:

- Variable

NOTE: Since class:SimpleTypeVariable is a sub-class of class:Variable a specific instance of OperationInput may also be a SimpleTypeVariable.

This Class is super-class of:

- none

Restrictions

- none

6.1.13 Class: OperationOutput

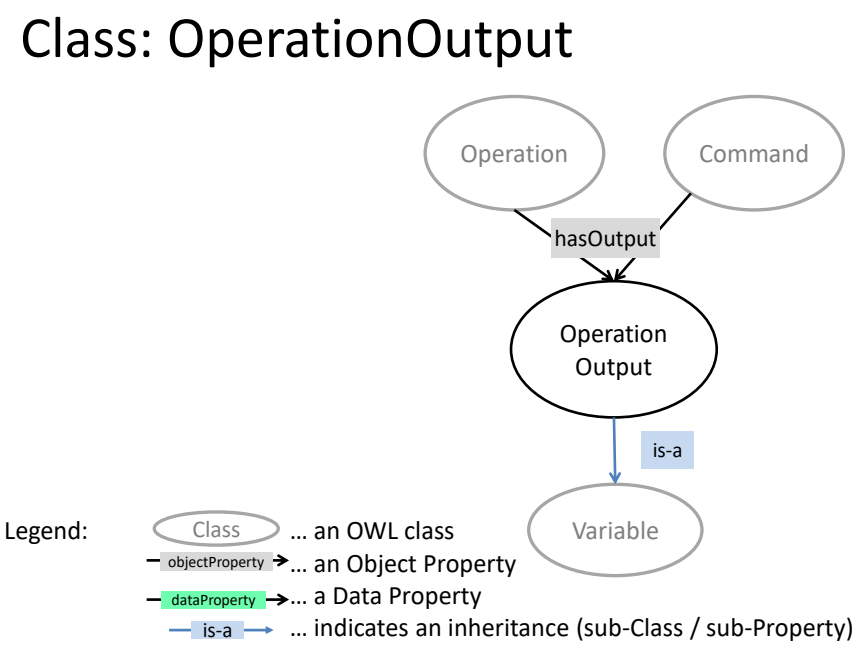

Figure 15: OperationOutput

Description

- **OperationOutput** (Class: OperationOutput) describes an output of an Operation. OperationOutput also describes the output of a Command:

  - OperationOutput is transient. An instance of OperationOutput is deleted when the instance of its Operation is deleted

  - An Operation/Command may have multiple OperationInputs and/or OperationOutputs

Object Properties

This Class is the domain Class of Object Property:

- hasMetaData (range Class: MetaData)
  (inherited from class:Variable)

- describes (range Class: Aspect)
  (inherited from class:Variable)

This Class is the range Class of Object Property:

- hasOutput (domain Class: Operation)
- hasOutput (domain Class: Command)

Data Properties

This Class is part of the domain Class of Data Property:

- Inherited from class: Variable and possibly Class SimpleTypeVariable
  (see clauses 6.1.17 and 6.1.18)

Superclass-subclass Relationships

This Class is sub-class of:

- Variable

NOTE: Since class:SimpleTypeVariable is a sub-class of class:Variable a specific instance of OperationOutput may also be a SimpleTypeVariable

This Class is super-class of:

- none

Restrictions

- none

6.1.14 Class: OperationState

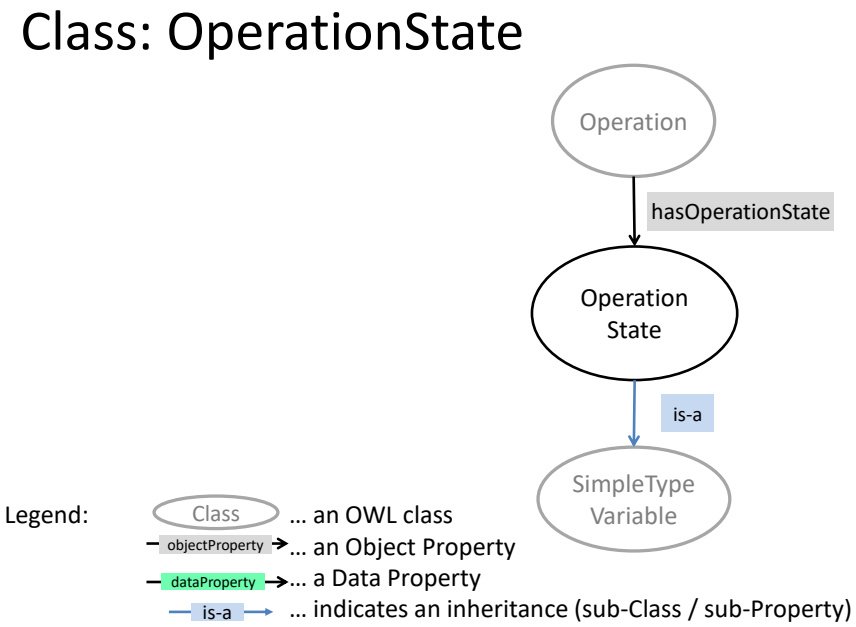

Figure 16: OperationState

Description

- **OperationState** (Class: OperationState) describes the current state of an Operation. The OperationState class represents all possible values for that state (enumerated individuals). The OperationState is set during the progress of the operation by the CSE and, optionally, the entity that is the target of the operation, e.g. a device (or for interworked devices by the IPE).

  This class contains a text string that is provided by the AE (e.g. an IPE). Values for that text that are specified in oneM2M are:

  - "data_received_by_application"
  - "operation_ended"
  - "operation_failed"
  - "data_transmitted_to_interworked_device"

  Additional values for the text string of the *operationState* attribute are permissible.

Object Properties

This Class is the range Class of Object Property:

- hasOperationState (domain Class: Operation)

Data Properties

This Class is part of the domain Class of Data Property:

- Inherited from class: Variable
  (see clause 6.1.17)
- hasDataType (range data type: xsd:string)
  (inherited from class: SimpleTypeVariable, see clause 6.1.18)
- hasDataRestriction_Pattern (range data type: xsd:string{"data received by application", "operation ended", "operation failed", "data transmitted to interworked device"})
  (inherited from class: SimpleTypeVariable, see clause 6.1.18)

Superclass-subclass Relationships

This Class is sub-class of:

- SimpleTypeVariable

This Class is super-class of:

- none

Restrictions

- hasDataRestriction_pattern exactly 1 xsd:string
- (hasDataRestriction_pattern value "data_received_by_application"^^xsd:string) or (hasDataRestriction_pattern value "data_transmitted_to_interworked_device"^^xsd:string) or (hasDataRestriction_pattern value "operation_ended"^^xsd:string) or (hasDataRestriction_pattern value "operation_failed"^^xsd:string)

NOTE: A OperationState has a Data Property "hasDataRestriction" with a pattern of exactly 1 xsd:string. The value of that xsd:string can be: "data_received_by_application", "data_transmitted_to_interworked_device", "operation_ended" or "operation_failed".

6.1.15 Class: InputDataPoint

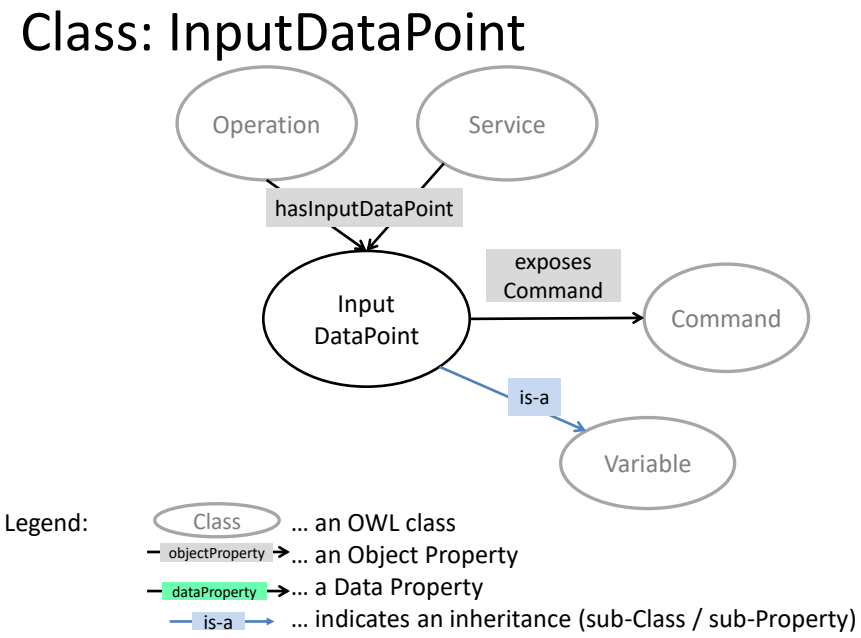

Figure 17: InputDataPoint

Description

- **InputDataPoint** (Class: InputDataPoint) is a Variable of a Service that is accessed by a RESTful Device in its environment and that the Device reads out autonomously (e.g. at periodic times). To enable a third party to instruct the device to retrieve (out of schedule) the current value of a InputputDataPoint devices often also offer a GET_InputDataPoint Operation to trigger the device to retrieve the data from the InputDataPoint:

  - An InputDataPoint is a persistent entity.

NOTE 1: Input- and Output DataPoints are usually used by Devices (AEs) that communicate in a RESTful way, while Operations are the procedures that are used for remote procedure based communication. Operations are, however, also needed in RESTful systems to correlate output, that is produced by a device, to the input that triggered the production of that output.

Object Properties

This Class is the domain Class of Object Property:

- hasMetaData (range Class: MetaData)
  (inherited from class:Variable)

- describes (range Class: Aspect)
  (inherited from class:Variable)

This Class is the range Class of Object Property:

- hasInputDataPoint (domain Class: Operation)

- hasInputDataPoint (domain Class: Command)

Data Properties

This Class is part of the domain Class of Data Property:

- Inherited from class: Variable and possibly Class SimpleTypeVariable
  (see clauses 6.1.17 and 6.1.18)

Superclass-subclass Relationships

This Class is sub-class of:

- Variable

NOTE 2: Since class:SimpleTypeVariable is a sub-class of class:Variable a specific instance of InputDataPoint may also be a SimpleTypeVariable.

This Class is super-class of:

- none

Restrictions

- none

6.1.16 Class: OutputDataPoint

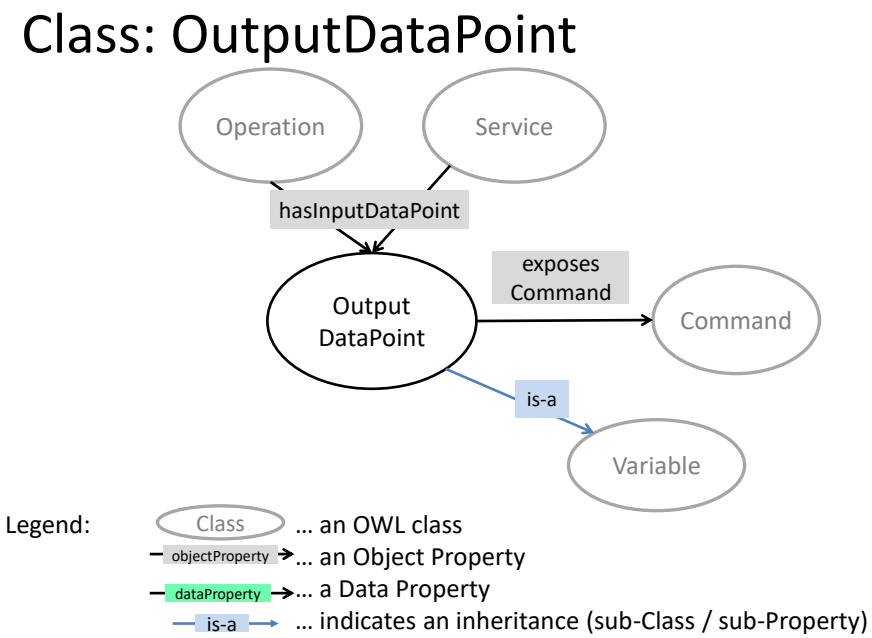

Figure 18: OutputDataPoint

Description

- **OutputDataPoint** (Class: OutputDataPoint) is a Variable of a Service that is set by a RESTful Device in its environment and that the Device updates autonomously (e.g. at periodic times). To enable a third party to instruct the device to update (out of schedule) the current value of a OutputputDataPoint devices often also offer a SET_OutputDataPoint Operation to trigger the device to update the data of the OutputDataPoint:

  - An OutputDataPoint is a persistent entity.

NOTE 1: Input- and Output DataPoints are usually used by Devices (AEs) that communicate in a RESTful way, while Operations are the procedures that are used for remote procedure based communication. Operations are, however, also needed in RESTful systems to correlate output, that is produced by a device, to the input that triggered the production of that output.

Object Properties

This Class is the domain Class of Object Property:

- hasMetaData (range Class: MetaData)
  (inherited from class:Variable)

- describes (range Class: Aspect)
  (inherited from class:Variable)

This Class is the range Class of Object Property:

- hasOutputDataPoint (domain Class: Operation)
- hasOutputDataPoint (domain Class: Command)

Data Properties

This Class is part of the domain Class of Data Property:

- Inherited from class: Variable and possibly Class SimpleTypeVariable
  (see clauses 6.1.17 and 6.1.18)

Superclass-subclass Relationships

This Class is sub-class of:

- Variable

NOTE 2: Since class:SimpleTypeVariable is a sub-class of class:Variable a specific instance of OutputDataPoint may also be a SimpleTypeVariable.

This Class is super-class of:

- none

Restrictions

- none

6.1.17 Class: Variable

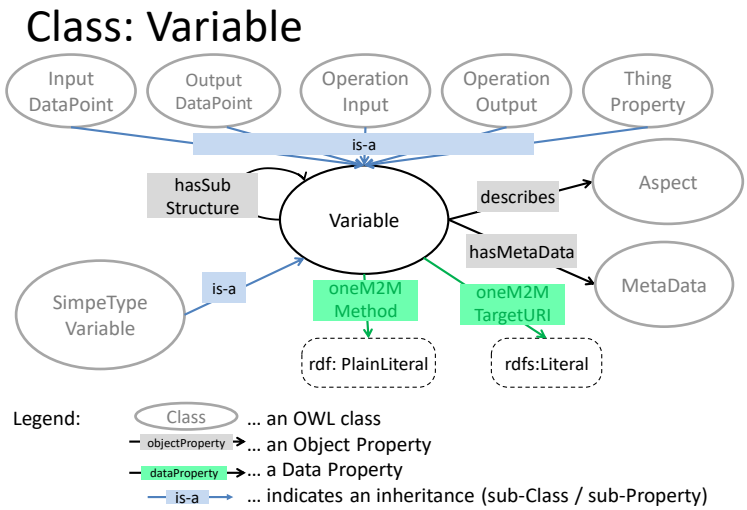

Figure 19: Variable

Description

- A **Variable** (Class: Variable) constitutes a super class to the following classes: ThingProperty, OperationInput, OperationOutput, OperationState, InputDataPoint, OutputDataPoint, SimpleTypeVariable. Its members are entities that store some data (e.g. integers, text, etc., or structured data) that can change over time.
  These data of the Variable usually describe some real-world Aspects (e.g. a temperature) and can have MetaData (e.g. units, precision, etc.)

Object Properties

This Class is the domain Class of Object Property:

- hasMetaData (range Class: MetaData)
- describes (range Class: Aspect)
- hasSubStructure (range Class: Variable)

If a Variable has a sub structure for which certain parts (i.e. Variables of the sub-structure) are mandatory then the related Object Property (hasSubStructure) shall have a Property Restriction with cardinality "min 1".

This Class is the range Class of Object Property:

- hasSubStructure (domain Class: Variable)

Data Properties

- oneM2MMethod (range datatype: rdf:PlainLiteral)
  This data property contains a oneM2M Method through which the oneM2M instantiation of the value of the Variable can be manipulated by the communicating entity:
  - It contains the string "RETRIEVE" for retrieving the variable when the oneM2M resource is of type <container> or <flexContainer>. This applies to sub-classes: OperationOutput, OutputDatapoint, ThingProperty and OperationState.
  - It contains the string "CREATE" for updating the variable when the oneM2M resource is of type <container>. This applies to sub-classes: OperationInput, InputDatapoint, ThingProperty.
  - It contains the string "UPDATE" for updating the variable when the oneM2M resource is of type <flexContainer>. This applies to sub-classes: OperationInput, InputDatapoint, ThingProperty.
- oneM2MTargetURI (range data type: rdfs: Literal)
  This data property contains the URI of a oneM2M resource (<container> or <flexContainer>) through which the oneM2M instantiation of the value of the Variable can be manipulated by the communicating entity. It can contain an absolute address or an address relative to the <semanticDescriptor> resource that holds the RDF description of the Variable.
  That address could be e.g. the value of the parentID for the <container> or <flexContainer> of a Input- or OutputDataPoint which has child-resource of type <semanticDescriptor> that holds the RDF description of the DataPoint.

Superclass-subclass Relationships

This Class is sub-class of:

- none

This Class is super-class of:

- ThingProperty
- OperationInput, OperationOutput
- OperationState
- InputDataPoint
- OutputDataPoint
- SimpleTypeVariable

Restrictions

- hasSubStructure only Variable

6.1.18 Class: SimpleTypeVariable

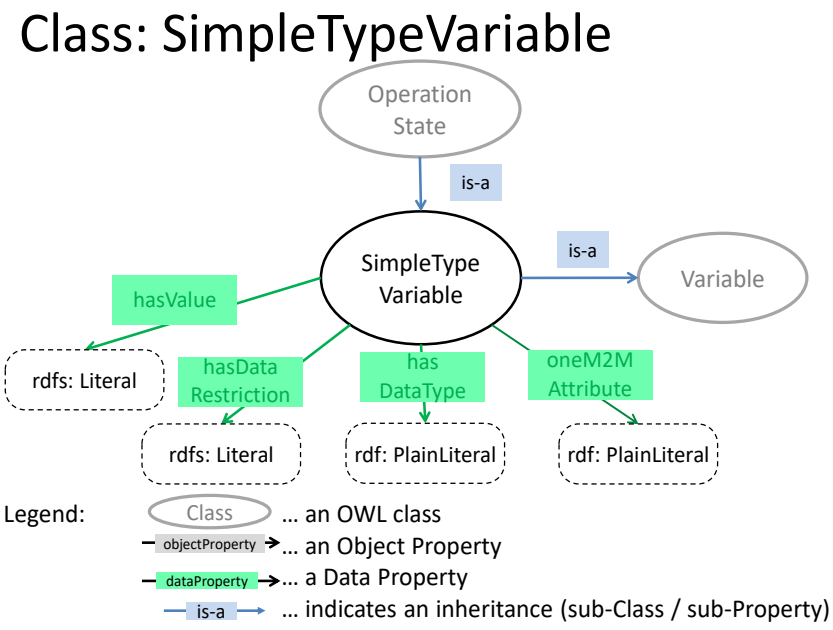

Figure 20: SimpleTypeVariable

Description

- **SimpleTypeVariable** (Class: SimpleTypeVariable) is a sub-class of class:Variable that only consists of Variables of simple xml types like xsd:integer, xsd:string, etc., potentially including restrictions

  The simple datatypes and -restrictions contained in "OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition)" [i.4] are supported.

Object Properties

This Class is the domain Class of Object Property:

- none

This Class is the range Class of Object Property:

- none

Data Properties

This Class is part of the domain Class of Data Property:

- Inherited from class: Variable.
  (see clause 6.1.17)

- hasValue (range data type: rdfs: Literal)
  This data property contains the value of the Variable if that value is part of the semantic description and is not contained in a different resource (identified by the oneM2MTargetURI data property). Storing the value of a Variable in a semantic description (i.e. as part of the RDF description in the semanticDescriptor resource) is useful for values that are relatively static (e.g. the name of the manufacturer):

  - Data properties "hasValue" and "oneM2MTargetURI" are mutually exclusive. Only one of the two shall be instantiated.

- oneM2MAttribute (range data: rdf:PlainLiteral)
  This Data Property contains the name of the attribute of the oneM2M resource (of type <container> or <flexContainer>) that is referenced with the oneM2MTargetURI and that stores the value of the SimpleTypeVariable

- hasDataType (range datatype: rdf:PlainLiteral)
  This Data Property contains the datatype of the SimpleTypeVariable as text string
- hasDataRestriction (range datatype: rdf:PlainLiteral)
  This Data Property contains a restriction of value of the SimpleTypeVariable

Superclass-subclass Relationships

This Class is sub-class of:

- Variable

This Class is super-class of:

- OperationState

Restrictions

- hasDataType exactly 1 rdf:PlainLiteral
- hasDataRestriction only rdf:PlainLiteral
- hasSubStructure exactly 0 Variable

NOTE: A Simple Type Variable has a data type but no other variables as sub-structure.

6.2 Object Properties

6.2.1 Void

6.2.2 Void

6.2.3 Object Property: consistsOf

Description

- A Device can consist of (i.e. be composed) of several (sub-) Devices

Domain Class

- Device

Range Class

- Device

6.2.4 Object Property: describes

Description

- A Variable describes an Aspect (a quality or kind)

Domain Class

- Variable

Range Class

- Aspect

6.2.5 Object Property: exposesCommand

Description

- A -machine interpretable- Operation or an Input/OutputDataPoint of a Service exposes a -human understandable- Command to a network.

Domain Class

- Operation

Range Class

- Command

6.2.6 Object Property: exposesFunction

Description

- A Service exposes a Function to the network and makes it discoverable, registerable and remotely controllable in the network.

Domain Class

- Service

Range Class

- Function

6.2.7 Object Property: hasCommand

Description

- A Function of a Device can be influenced/observed by a human user through the Commands that this Function has and that are offered to the user

Domain Class

- Function

Range Class

- Command

6.2.8 Object Property: hasFunction

Description

- In order to accomplish its task, a Device performs one or more Functions

Domain Class

- Device

Range Class

- Function

6.2.9 Object Property: hasInput

Description

- An Operation of a Service of the Device or a Command of a Function of the Device can have OperationInput data.

Domain Class

- Operation
- Command

Range Class

- OperationInput

6.2.10 Object Property: hasInputDataPoint

Description

- A Service or an Operation of a Service of the Device can have InputDataPoints. Communicating entities write data into InputDataPoints and the Device retrieves the data at times according to an internal schedule.
  An InputDataPoint can also contain the data that are used as input to an Operation.

Domain Class

- Operation
- Service

Range Class

- InputDataPoint

6.2.11 Object Property: hasMetaData

Description

- A Variable can have MetaData (like units, precision-ranges, etc.)

Domain Class

- Variable

Range Class

- MetaData

6.2.12 Void

6.2.13 Object Property: hasOperation

Description

- A Service communicates by means of Operations over the network to transmit data to/from other devices

Domain Class

- Service

Range Class

- Operation

6.2.14 Object Property: hasOperationState

Description

- An Operation may have an OperationState that is exposed

Domain Class

- Operation

Range Class

- OperationState

6.2.15 Void

6.2.16 Object Property: hasOutput

- An Operation of a Service of the Device or a Command of a Function of the Device can have OperationOutput data

Domain Class

- Operation
- Command

Range Class

- OperationOutput

6.2.17 Object Property: hasOutputDataPoint

Description

- A Service or an Operation of a Service of the Device can have OutputDataPoints. The Device writes data into OutputDataPoints at times according to an internal schedule and the communicating entities retrieves the data. An OutputDataPoint can also contain the data that are created as output of an Operation.

Domain Class

- Operation
- Service

Range Class

- OutputDataPoint

6.2.18 Object Property: hasService

Description

- The Functions of a Device are exposed in the network as Services of the Device

Domain Class

- Device

Range Class

- Service

6.2.19 Object Property: hasSubStructure

Description

- A structured Variable can be composed of (sub-)Variables

Domain Class

- Variable

Range Class

- Variable

6.2.20 Object Property: hasThingProperty

Description

- A Thing may have properties that can be described by Values

Domain Class

- Thing

Range Class

- Value

6.2.21 Object Property: hasThingRelation

Description

- A Thing may have relations to itself or to other Things

Domain Class

- Thing

Range Class

- Thing

6.2.22 Void

6.2.23 Void

6.2.24 Void

6.2.25 Object Property: isPartOf

Description

- An InterworkedDevice consist a part of an AreaNetwork

Domain Class

- InterworkedDevice

Range Class

- AreaNetwork

6.2.26 Object Property: refersTo

Description

- A Function of a Device can refer to a certain Aspect (a quality or kind) that is measured or controlled by that Function.
  e.g. a temperature sensor would refer to the Aspect "Temperature" that it measures

Domain Class

- Function

Range Class

- Aspect

6.3 Data Properties

6.3.1 Data Property: hasDataType

Note that in the present document the name space identifier for:

- 'http://www.w3.org/2001/XMLSchema' shall be referred to using the prefix: xsd
- 'http://www.w3.org/2002/07/owl' shall be referred to using the prefix: owl
- 'http://www.w3.org/1999/02/22-rdf-syntax-ns' shall be referred to using the prefix: rdf

Description

- This Data Property specifies the data type of the SimpleTypeVariable as URI

Domain Class

- SimpleTypeVariable

Range Datatype

- *xsd:anyURI*

Permissible URIs are:

- for Numbers types
  (possible restrictions: xsd:minInclusive, xsd:maxInclusive, xsd:minExclusive, xsd:maxExclusive):
  - *owl:real*
  - *owl:rational*
  - *xsd:decimal*
  - *xsd:integer*
  - *xsd:nonNegativeInteger*
  - *xsd:nonPositiveInteger*
  - *xsd:positiveInteger*
  - *xsd:negativeInteger*

- *xsd:long*
- *xsd:int*
- *xsd:short*
- *xsd:byte*
- *xsd:unsignedLong*
- *xsd:unsignedInt*
- *xsd:unsignedShort*
- *xsd:unsignedByte*

- for PlainLiteral - contains all String types
  (possible restrictions: xsd:length, xsd:minLength, xsd:maxLength, xsd:pattern, rdf:langRange):
  - *rdf:PlainLiteral*
- for String types
  (possible restrictions: xsd:length, xsd:minLength, xsd:maxLength, xsd:pattern):
  - *xsd:string*
  - *xsd:normalizedString*
  - *xsd:token*
  - *xsd:language*
  - *xsd:Name*
  - *xsd:NCName*
  - *xsd:NMTOKEN*
- for Boolean Values (no restrictions):
  - *xsd:boolean*
- for Binary Data types
  (possible restrictions: xsd:minLength, xsd:maxLength, xsd:length):
  - *xsd:hexBinary*
  - *xsd:base64Binary*
- for IRIs
  (possible restrictions: xsd:minLength, xsd:maxLength, xsd:length, xsd:pattern):
  - *xsd:anyURI*
- for Time Instants
  (possible restrictions: sd:minInclusive, xsd:maxInclusive, xsd:minExclusive, xsd:maxExclusive):
  - *xsd:dateTime*
  - *xsd:dateTimeStamp*
- for Literals:
  - *rdf:XMLLiteral*

6.3.2 Data Property: hasDataRestriction

6.3.2.0 General description

Description

- This Data Property specifies the restrictions on the data type of the SimpleTypeVariable.

Domain Class

- SimpleTypeVariable

Range Datatype

- rdf:PlainLiteral

The Data Property "hasDataRestriction" shall always be sub-classed as one of the following specializations.

The range of a sub-classed Data Property "hasDataRestriction" shall be instantiated as the value restricting the data. E.g. a value 100 in the range of Data Property: hasDataRestriction_minInclusive specifies that the SimpleTypeVariable can only take values greater or equal to 100.

6.3.2.1 Data Property: hasDataRestriction_minInclusive

For applicability of this sub-class of data property: hasDataRestriction see clause 6.3.1 Data Property: hasDataType.

6.3.2.2 Data Property: hasDataRestriction_maxInclusive

For applicability of this sub-class of data property: hasDataRestriction see clause 6.3.1 Data Property: hasDataType.

6.3.2.3 Data Property: hasDataRestriction_minExclusive

For applicability of this sub-class of data property: hasDataRestriction see clause 6.3.1 Data Property: hasDataType.

6.3.2.4 Data Property: hasDataRestriction_maxExclusive

For applicability of this sub-class of data property: hasDataRestriction see clause 6.3.1 Data Property: hasDataType.

6.3.2.5 Data Property: hasDataRestriction_length

For applicability of this sub-class of data property: hasDataRestriction see clause 6.3.1 Data Property: hasDataType.

6.3.2.6 Data Property: hasDataRestriction_minLength

For applicability of this sub-class of data property: hasDataRestriction see clause 6.3.1 Data Property: hasDataType.

6.3.2.7 Data Property: hasDataRestriction_maxLength

For applicability of this sub-class see of data property: hasDataRestriction clause 6.3.1 Data Property: hasDataType.

6.3.2.8 Data Property: hasDataRestriction_pattern

For applicability of this sub-class of data property: hasDataRestriction see clause 6.3.1 Data Property: hasDataType.

6.3.2.9 Data Property: hasDataRestriction_langRange

For applicability of this sub-class of data property: hasDataRestriction see clause 6.3.1 Data Property: hasDataType.

6.3.3 Data Property: hasValue

Description

- This data property contains the value of the Variable if that value is part of the semantic description and is not contained in a different resource (identified by the oneM2MTargetURI data property). Storing the value of a Variable in a semantic description (i.e. as part of the RDF description in the semanticDescriptor resource) is useful for values that are relatively static (e.g. the name of the manufacturer):
  - Data properties "hasValue" and "oneM2MTargetURI" are mutually exclusive. Only one of the two shall be instantiated.

Domain Class

- SimpleTypeVariable

Range Datatype

- rdfs: Literal

6.3.4 Data Property: netTechnologyCommunicationProtocol

Description

- Identifies a communication protocol (e.g. ZigBee_1_0)

Domain Class

- AreaNetwork

Range Datatype

- netTechnologyCommunicationProtocol rdf:PlainLiteral

6.3.5 Data Property: netTechnologyPhysicalStandard

Description

- netTechnologyPhysicalStandardIdentification of the physical properties of an Area Network technology (e.g. IEEE_802_15_4_2003_2_4GHz).

Domain Class

- AreaNetwork

Range Datatype

- rdf:PlainLiteral

6.3.6 Data Property: netTechnologyProfile

Description

- netTechnologyProfileIdentification of a profile (e.g. ZigBee_HA) of an Area Network technology

Domain Class

- AreaNetwork

Range Datatype

- rdf:PlainLiteral

6.3.7 Data Property: oneM2MTargetURI

Description

- oneM2MTargetURI (range data type: rdfs: Literal)
  This data property contains the URI of a oneM2M resource (<container> or <flexContainer>) through which the oneM2M instantiation of the value of the Variable can be manipulated by the communicating entity. It can contain an absolute address or an address relative to the <semanticDescriptor> resource that holds the RDF description of the Variable.
  That address could be e.g.:

  - The value of the parentID for the <container> or <flexContainer> of a Input- or OutputDataPoint which has child-resource of type <semanticDescriptor> that holds the RDF description of the DataPoint.

Domain Class

- Operation
- Variable

Range Datatype

- rdfs:Literal

6.3.8 Data Property: oneM2MAttribute

Description

- This Data Property contains the name of the attribute of the oneM2M resource of type <flexContainer> or the child resource of the oneM2M resource of type <container> that is referenced with the oneM2MTargetURI and that stores the value of the SimpleTypeVariable:

  - if the resource-type of the oneM2M resource that is referenced with the oneM2MTargetURI is <container> then this Data Property shall contain the text string "#latest".

Domain Class

- SimpleTypeVariable

Range Datatype

- rdf:PlainLiteral

6.3.9 Data Property: oneM2MMethod

Description

- This data property contains a oneM2M CRUD Method through which the oneM2M instantiation of the value of the Variable can be manipulated by the communicating entity:

  - It contains the string "RETRIEVE" for retrieving the variable when the oneM2M resource is of type <container> or <flexContainer>. This applies to sub-classes: OperationOutput, OutputDatapoint, ThingProperty and OperationState.

  - It contains the string "CREATE" for updating the variable when the oneM2M resource is of type <container>. This applies to sub-classes: OperationInput, InputDatapoint, ThingProperty.

  - It contains the string "UPDATE" for updating the variable when the oneM2M resource is of type <flexContainer>. This applies to sub-classes: OperationInput, InputDatapoint, ThingProperty.

Domain Class

- Variable

- Operation

Range Datatype

- rdf:PlainLiteral

6.3.10 Data Property: hasThingAnnotation

Description

- This data property contains a description of a Thing.

NOTE: Data Property: hasThingAnnotation should be used in cases where the annotation data do not change very often and/or where they are usually not the subject of semantic operations (e.g. semantic discovery). Otherwise Object Property: hasThingPropery should be used.

- E.g. a certain type of Device could have the model (as a numeric description) or the name of the manufacturer (as a textual annotation).

Domain Class

- Thing

Range Datatype

- rdfs: Literal

6.4 Annotation Properties

6.4.1 Annotation Property: resourceDescriptorLink

Description

- The *resourceDescriptorLink* annotation property is used to refer to a *semanticDescriptor* resource that contains more information about its subject. Its subject may be any individual and the range shall be the data literal or URI reference that represents the address of the *semanticDescriptor*:

  - For a oneM2M instantiation of the Base Ontology the *resourceDescriptorLink* annotation property is used to annotate instances that appear in the range of object properties. The URI points to the *semanticDescriptor* that contains more information about the instance of that class.

NOTE: In OWL DL it is not allowed to define property axioms on annotation properties, i.e. it is not possible to define a domain and a range within the ontology itself.

Domain Class

- owl:Thing

Range Datatype

- *xsd:anyURI*

7 Instantiation of the Base Ontology and external ontologies to the oneM2M System

7.1 Instantiation rules for the Base Ontology

7.1.1 Instantiation of classes of the oneM2M Base Ontology and derived external ontologies in the oneM2M System:

7.1.1.1 General on instantiating classes of the Base Ontology in the oneM2M System

Clause 7.1.1 describes how the Base Ontology shall be instantiated in the oneM2M System.

NOTE 1: Apart from semantically describing oneM2M Solutions a standardized oneM2M instantiation of the Base Ontology is also needed for the purpose of Interworking with full semantic mapping when the non-oneM2M data model is described by a oneM2M compliant ontology as described in clause F.5 of oneM2M TS-0001 [2].

NOTE 2: Other instantiations of the Base Ontology are permissible if interworking according to clause F.5 of oneM2M TS-0001 [2] is not required.

Every instantiation of a class of the Base Ontology (or a sub-class thereof) shall be instantiated in a *descriptor* attribute of a oneM2M resource of type <*semanticDescriptor*>. A <*semanticDescriptor*> resource may instantiate multiple classes.

That <*semanticDescriptor*> resource shall:

a) Contain instantiations of classes in the RDF data [3] of its *descriptor* attribute:

   - Every instance of a class shall be globally identified within the oneM2M Solution using the *rdf:about* attribute that contains a URI (e.g. based on a MAC address of a Device) that is unique within the oneM2M Solution.

NOTE 3: The choice of a suitable unique URI is out of scope of oneM2M.

b) Contain an Ontology-Ref attribute that identifies the class whose instantiation is described in the *descriptor* attribute. Depending on the instantiation this is a class in the Base Ontology or a class of another ontology that is a sub-class of (includes equals to) a class in the Base Ontology as defined in clause 5.1.2.2.

The <*semanticDescriptor*> shall also contain:

a) The instantiated Object Properties for which the instantiated class is the domain class.

b) The instantiated Data Properties for which the instantiated class is the domain class.

NOTE 4: Instantiations of the domain class and the range class of an object property may be contained in <*semanticDescriptor*>different <*semanticDescriptor*> resources.

If the range class of an object property is instantiated in a <*semanticDescriptor*> resource that is different to the <*semanticDescriptor*> resource in which the domain class is instantiated then the <*semanticDescriptor*> of the instance of the domain class shall contain:

a) an instance of the *resourceDescriptorLink* annotation property that contains the URI of the *semanticDescriptor* of the instance of the range class.

For specific classes that are indicated in clause 7.1.1.2 (in particular class:OperationState and classes that are derived from class:Variable) the values of Data Properties may be stored in the parent resource of that <*semanticDescriptor*> instead of the RDF data [3] of the *descriptor* attribute.

Any class of a derived external ontology shall be either:

- a sub-class of a class of the Base Ontology; or
- the range class of some Object Property whose domain class is a class (e.g. class:Thing) or sub-class of the Base Ontology.

If a class of a derived external ontology is a sub-class of a class of the Base Ontology then it shall be instantiated in the same way as the class of the Base Ontology.

If a class of a derived external ontology is not a sub-class of the Base Ontology but is the range class of some Object Property whose domain class is a class or sub-class of the Base Ontology then it shall be instantiated in the data of the *descriptor* attribute of the <*semanticDescriptor*> child resource of the oneM2M resource that instantiates the sub-class of the Base Ontology.

7.1.1.2 Instantiation of individual classes of the Base Ontology

An overview of the oneM2M resources for instantiating the classes of the oneM2M Base Ontology is shown in the figure 21. Different colours indicate different resource types.

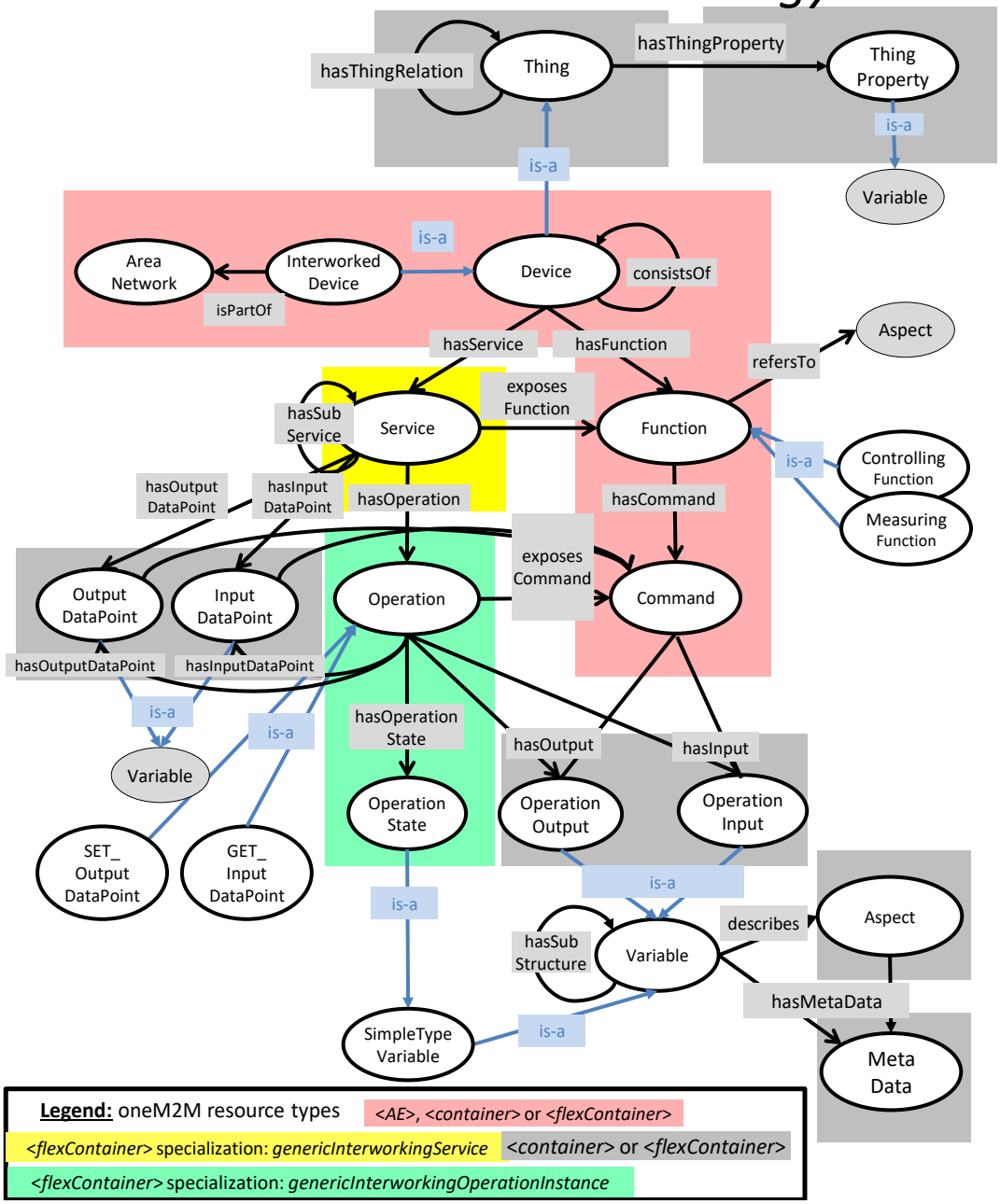

Figure 21: oneM2M instantiation of the Base Ontology

- The **Device** class of the oneM2M Base Ontology (or a sub-class thereof that is not an InterworkedDevice) shall be instantiated in the data of the *descriptor* attribute of a resource of type <*semanticDescriptor*> that is a child resource of an <*AE*>.

  The Device instance is identified using the *rdf:about* attribute that contains a URI (e.g. the MAC address) that is unique within the oneM2M Solution.

  The application logic (identified by its APP-ID of the Device instance is provided by the Application Entity (AE) of that Device.

NOTE 1: The *resourceID* of a <*node*> resource that stores the node specific information where this AE resides is contained in the *nodeLink* attribute of the <AE> of the Device.

- The **InterworkedDevice** class of the oneM2M Base Ontology (or a sub-class) shall be instantiated in the data of the *descriptor* attribute of a resource of type <*semanticDescriptor*> that is a child resource of:

  - an <*AE*> resource of its Interworking Proxy Application Entity (IPE); or, alternatively;

  - a <*container*> or <*flexContainer*> resource that is a child resource of the <*AE*> resource of its Interworking Proxy Application Entity (IPE).

  The InterworkedDevice instance is identified using the *rdf:about* attribute that contains a URI (e.g. the device identifier of the device in the interworked system) that is unique within the oneM2M Solution.

  The APP-ID of the <*AE*> that is the parent (or grand-parent) of the <*semanticDescriptor*> which contains an instance of an InterworkedDevice, shall be the APP-ID of InterworkedDevice's IPE.

NOTE 2: The reason for the different instantiation of Device and InterworkedDevice is the following:

  Case 1 - native oneM2M device:

  - A oneM2M device (ASN, ADN - and potentially MN) always has an AE and an <*AE*> resource. Therefore the Device class for a oneM2M device needs to be instantiated in the *descriptor* attribute of the <*semanticDescriptor*> child resource of that <*AE*>.

  Case 2 - interworked (non-oneM2M) device:

  - In the case of an interworked (non-oneM2M) device the IPE may

    - Instantiate the InterworkedDevice class for the interworked (non-oneM2M) devices in the *descriptor* attribute of the <*semanticDescriptor*> child resource of the IPE's <*AE*>.

    - The IPE can create multiple <*AE*> resources, one for each interworked device.
      In this case each interworked device needs to be instantiated in the descriptor attribute of the <semanticDescriptor> child resource of that newly created <*AE*>.

    - The IPE can create multiple <*container*> or <*flexContainer*> resources as child resources of the <*AE*> resource of its Interworking Proxy Application Entity (IPE) , one for each interworked device.
      In this case each interworked (non-oneM2M) device needs to be instantiated in the descriptor attribute of the <*semanticDescriptor*> child resource of that <*container*> or <*flexContainer*>.

- The **AreaNetwork** class (or a sub-class) shall be instantiated in the data of the *descriptor* attribute of the <*semanticDescriptor*> child resource of the oneM2M resource that instantiates the InterworkedDevice class:

  - The Data Properties "anTechnologyCommunicationProtocol", "anTechnologyPhysicalStandard" and "anTechnologyProfile" are instantiated in the *descriptor* attribute of the <*semanticDescriptor*> child resource of the oneM2M resource that instantiates the InterworkedDevice class.

- The **Service** class (or a sub-class) shall be instantiated in the data of the *descriptor* attribute of the <*semanticDescriptor*> child resource of a *genericInterworkingService* (specialization of <flexContainer>) resource.

  The instance is identified using the *rdf:about* attribute that contains the URI of the Device concatenated with the letter "*" and the class name of the Service (e.g. 00:11:2F:74:2C:8F*MyService).

  The *genericInterworkingService* resource shall be a child resource of the (<*AE*>, <*container*> or <*flexContainer*>) resource that contains the <*semanticDescriptor*> which instantiates the Device class. It contains references to the <container> or <flexContainer> resources that represent Input- and/or OutputDatapoints of the Service.

- The **Function** class (or sub-class) shall be instantiated in the data of the *descriptor* attribute of the <*semanticDescriptor*> child resource of the oneM2M resource that instantiates the Device class.

  The instance is identified using the *rdf:about* attribute that contains the URI of the Device concatenated with the letter "*" and the class name of the Function (e.g. 00:11:2F:74:2C:8F*MyFunction).

- The **Command** class (or sub-class) shall be instantiated in the data of the *descriptor* attribute of the <*semanticDescriptor*> child resource of the oneM2M resource that instantiates the Device class.

  The instance is identified using the rdf:about attribute that contains the URI of the Device concatenated with the letter "*" and the class name of the Command (e.g. 00:11:2F:74:2C:8F*MyCommand).

- The **Operation** class (or sub-classes) shall be instantiated in the *descriptor* attribute of the <*semanticDescriptor*> child resource of a *genericInterworkingOperationInstance* (specialization of <*flexContainer*>) resource.

  The instance is identified using the *rdf:about* attribute that contains the URI of the Device concatenated with the letter "*" and the class name of the Service, concatenated with the letter "*" and a combination of the class name of the Operation with a number that makes the instance unique within its Service instance during the operation's lifetime
  (e.g. at a certain point in time a Service instance might have Operation instances with OperationInstances with IDs:

  - "00:11:2F:74:2C:8F*MyService*MyOperation1", "00:11:2F:74:2C:8F*MyService*MyOperation5";
  - "00:11:2F:74:2C:8F*MyService*MyOtherOperation1";
  - "00:11:2F:74:2C:8F*MyService*MyThirdOperation8").

  The *genericInterworkingOperationInstance* resource shall be a child resource of the *genericInterworkingService* resource that contains the <*semanticDescriptor*> which instantiates the Service class:

  - The range instance of Object Property "hasOperation" that links the instance of the Service to the instance of the Operation shall be annotated with an Annotation Property: resourceDescriptorLink <*semanticDescriptor*>which shall contain a reference to the resource of type <*semanticDescriptor*> that instantiates the Operation.

- The **OperationInput** and **OperationOutput** class (or sub-class) shall be instantiated in the data of the *descriptor* attribute of the <*semanticDescriptor*> child resource of a <*container*> or <*flexContainer*>.

  The instance is identified using the rdf:about attribute that contains the URI of the OperationInstance concatenated with the letter "*" and the class name of the OperationInput and OperationOutput (e.g. 00:11:2F:74:2C:8F*MyService*MyThirdOperation8*MyOperationOutput).

  The <*container*> or <*flexContainer*>, whose the <*semanticDescriptor*> child resource contains the instance of the OperationInput or OperationOutput shall be a child resource of the *genericInterworkingOperationInstance* resource:

  - The range of an instantiation of Object Properties "hasInput" and "hasOutput" that link the instance of the Operation to the instance of the OperationInput and OperationOutput shall be annotated with an Annotation Property: resourceDescriptorLink which shall contain a reference to the resource of type <*semanticDescriptor*> that instantiates the OperationInput and OperationOutput.

- The **InputDataPoint** and **OutputDataPoint** class (or sub-class) shall be instantiated in the data of the *descriptor* attribute of the <*semanticDescriptor*> child resource of a <*container*> or <*flexContainer*>.

  The instance is identified using the *rdf:about* attribute that contains the URI of the Device concatenated with the letter "*" and the class name of the InputDataPoint or OutputDataPoint
  (e.g. 00:11:2F:74:2C:8F*MyInputDataPoint).

  The <*container*> or <*flexContainer*> resource shall be a child resource of the (<*AE*>, <*container*> or <*flexContainer*>) resource that contains the <*semanticDescriptor*> which instantiates the Device class or InterworkedDevice class:

  - The range of an instantiation of Object Properties "hasInputDataPoint" and "hasOutputDataPoint" that link the instance of a Service or Operation to the instance of the InputDataPoint and OutputDataPoint shall be annotated with an Annotation Property: resourceDescriptorLink which shall contain a reference to the resource of type <*semanticDescriptor*> that instantiates the InputDataPoint and OutputDataPoint.

- <*semanticDescriptor*>The **OperationState** class (or sub-class) shall be instantiated in the data of the *descriptor* attribute of the <*semanticDescriptor*> child resource of the *genericInterworkingOperationInstance* resource that is related via the "hasOperationState" Object Property.

  The instance is identified using the rdf:about attribute that contains the URI of the OperationInstance concatenated with the letter "*" and "OperationState" (i.e. the class name of the OperationState) (e.g. 00:11:2F:74:2C:8F*MyService*MyThirdOperation8*OperationState):

  - The data property "oneM2MTargetURI" shall contain the URI of the genericInterworkingOperationInstance resource.

  - The data property "oneM2MAttribute" shall obtain the value "OperationState" (i.e. the name of the Attribute of the OperationState in the *genericInterworkingOperationInstance* resource).

- <*semanticDescriptor*>The **Aspect** class (or sub-classes) may be instantiated in the data of the *descriptor* attribute of the <*semanticDescriptor*> child resource of any resource type that allows a <*semanticDescriptor*> child resource.

  The instance is identified using the using the rdf:about attribute that contains a URI that is unique within the oneM2M Solution.

NOTE 3: The choice of a suitable unique URI is out of scope of oneM2M.

- The **Thing** class (or sub-classes) may be instantiated in the data of the *descriptor* attribute of the <*semanticDescriptor*> child resource of any resource type that allows a <*semanticDescriptor*> child resource:

  - The instance is identified using the using the rdf:about attribute that contains a URI that is unique within the oneM2M Solution.

NOTE 4: The choice of a suitable unique URI is out of scope of oneM2M.

  - The range of an instantiation of Object Property "hasThingRelation" that links the instance of the Thing to the instance of a second Thing shall be annotated with an Annotation Property: resourceDescriptorLink which shall contain a reference to the resource of type <*semanticDescriptor*> that instantiates the second Thing.

NOTE 5: This reference could refer to Thing, or a Device (as a sub-class of Thing).

  - The range of an instantiation of an Object Property "hasThingProperty" that links the instance of the Thing to an instance of a ThingProperty shall be annotated with an Annotation Property: resourceDescriptorLink which shall contain a reference to the resource of type <*semanticDescriptor*> that instantiates the ThingProperty<*semanticDescriptor*>.

- The **TingProperty** class (or sub-class) shall be instantiated in the data of the *descriptor* attribute of the <*semanticDescriptor*> of the Thing or of a separate <*container*> or <*flexContainer*>.

  The instance is identified using the *rdf:about* attribute that contains the URI of the Thing concatenated with the letter "*" and the class name of the ThingProperty (e.g. *[some out of scope Thing URI]**MyThingProperty):

  - If the TingProperty is a SimpleTypeVariable and contains in its data property "hasValue" the value of the ThingProperty then it can be instantiated in the data of the *descriptor* attribute of the <*semanticDescriptor*> of the Thing.

  - Otherwise the TingProperty shall be instantiated in the data of the *descriptor* attribute of a separate <*container*> or <*flexContainer*> which shall be a child resource of the parent resource of <*semanticDescriptor*> which instantiates the Thing class.
    In this case the range of an Object Property "hasThingProperty" that links an instance of a Thing to the instance of the ThingProperty shall be annotated with an Annotation Property: resourceDescriptorLink which shall contain a reference to the resource of type <*semanticDescriptor*> that instantiates the ThingProperty.

- The sub-classes of the **Variable** class shall be instantiated in the data of the *descriptor* attribute of the <*semanticDescriptor*> child resource of resources of a <container> or <*flexContainer*>.
  If the Variable has a structure (if it is composed of (sub-)Variables) - i.e. it is not a SimpleTypeVariable then:

  - The <*semanticDescriptor*> shall have instances of object property "hasSubStructure" and each instance contains in its range an instance of a (sub-) Variable. If that (sub-)Variable is part of a different <*semanticDescriptor*> resource then the range of an object property "hasSubStructure" shall be annotated with an annotation property: "resourceDescriptorLink" which shall contain a reference to the resource of type <*semanticDescriptor*> that instantiates the (sub-)Variable.

  - The<*semanticDescriptor*> shall contain an instance of data property "oneM2MTargetURI" which shall contain the URI of the parent <container> or <*flexContainer*> resource.

  - The<*semanticDescriptor*> shall have instances of the data property "oneM2MMethod" which indicates a oneM2M CRUD Method through which the oneM2M instance of the value of the Variable can be manipulated by the communicating entity:

    - It contains the string "RETRIEVE" for retrieving the variable when the oneM2M resource is of type <container> or <flexContainer>. This applies to sub-classes: OperationOutput, OutputDatapoint, ThingProperty and OperationState.

    - It contains the string "CREATE" for updating the variable when the oneM2M resource is of type <container>. This applies to sub-classes: OperationInput, InputDatapoint, ThingProperty.

    - It contains the string "UPDATE" for updating the variable when the oneM2M resource is of type <flexContainer>. This applies to sub-classes: OperationInput, InputDatapoint, ThingProperty.

- The sub-classes of the **SimpleTypeVariable** class shall be instantiated in the data of the *descriptor* attribute of the <*semanticDescriptor*> child resource of resources of a <container> or <*flexContainer*>.
  The data properties are the same as for instances of the Variable class.
  In addition:

  - The data property "hasValue" contains the value of the Variable if that value is part of the semantic description and is not contained in a different resource (identified by the oneM2MTargetURI data property).

NOTE 6: Storing the value of a Variable in a semantic description (i.e. as part of the RDF description in the <*semanticDescriptor*> resource) is useful for values that are relatively static (e.g. the name of the manufacturer).

- Data properties "hasValue" and "oneM2MTargetURI" are mutually exclusive. Only one of the two shall be instantiated for a SimpleTypeVariable.

- If data property "oneM2MTargetURI" is instantiated then the data property "oneM2MAttribute" shall also be instantiated and contain:

  - In the case of a <*flexContainer*> the name of the Attribute of the SimpleTypeVariable in the <*flexContainer*> resource).

  - In the case of a <*container*> the value "#latest".

- The **MetaData** class (or sub-classes) may be instantiated in the data of the descriptor attribute of the <*semanticDescriptor*> child resource of any resource type that allows a <*semanticDescriptor*> child resource.

  The instance is identified using the using the rdf:about attribute that contains a URI that is unique within the oneM2M Solution.

7.1.2 Instantiation of Object Properties

Object properties relate an instance of domain class to an instance of the range class.
They shall be instantiated in the data of the *descriptor* attribute of the <*semanticDescriptor*> resource that instantiates the domain class of the object property.

If the range class of an Object Property is instantiated in a different resource than the instantiation of the domain class then the range class of an Object Property shall be annotated with the Annotation Property: resourceDescriptorLink which shall contain a reference to that resource.

7.1.3 Instantiation of Data Properties

Data properties shall be instantiated in the data of the *descriptor* attribute of the <*semanticDescriptor*> resource that instantiates the domain class of the data property.

7.1.4 Instantiation of Annotation Properties

Annotation properties may be instantiated in the data of the *descriptor* attribute of any <*semanticDescriptor*> resource.

For a oneM2M instantiation of the Base Ontology the resourceDescriptorLink annotation property is used to annotate the range of object properties with the URI of the <*semanticDescriptor*> that contains the instance (that holds the RDF description of the instance) of the range class of the object property.

The resourceDescriptorLink annotation property is not part of the semantic description but is used to refer to a <*semanticDescriptor*> resource that contains more information about its subject. It is resolved by combining the content of the *descriptor* attributes of the <*semanticDescriptor*> resources before a semantic operation (e.g. SPARQL query) is performed.

7.2 Common mapping principles between the Base Ontology and external ontologies

The base ontology can be mapped to other external ontologies (e.g., SAREF, SSN, etc.). The following principles are applied for the mapping between ontologies:

- Principle 1 - Classes Mapping (owl:equivalentClass):
  - Making the statement X owl:equivalentClass Y essentially means that two named classes are synonymous, i.e. that all instances of class X are instances of class Y and vice versa.
  - Using this principle, two classes specified in different ontologies are declared to be equivalent.
- Principle 2 - Properties Mapping (owl:equivalentProperty):
  - The owl:equivalentProperty construct can be used to state that two properties have the same property extension. Syntactically, owl:equivalentProperty is a built-in OWL property with rdf:Property as both its domain and range.
  - Using this principle, if two properties are declared to be equivalent, two properties have the same semantics or meaning.
- Principle 3 - Class Instances Mapping (owl:sameAs):
  - The property owl:sameAs is used to state that two individuals (i.e. class instances) are the same.
  - Using this principle, two class instances specified in different ontologies are declared to be equivalent.
- Principle 4 - SubClass Mapping (rdfs:subClassOf):
  - The property rdfs:subClassOf is used to state that the class extension of a class description is a subset of the class extension of another class description.

- Making the statement X rdfs:subClassOf Y essentially means that all instances of class X are instances of class Y.

- Principle 5 - SubProperties Mapping (rdfs:subPropertyOf):

  - The property rdfs:subPropertyOf is used to state that one property is the subproperty of another property. Syntactically, the domain and range of rdfs:subPropertyOf are both of type rdf:Property.

  - Making the statement X rdfs:subPropertyOf Y essentially means that all resources related by property X are also related by property Y.

When using the above principles for the mapping between the Base Ontology and an external ontology, the hierarchy and relations of the classes and properties defined in these two ontologies should be carefully considered since there may exist some incompatible points after mapping due to the inheritance of the properties.

8 Functional specification of communication with the Generic interworking IPE

8.1 Usage of oneM2M resources for IPE communication

8.1.1 General design principles (informative)

For Generic interworking the oneM2M resource types <*AE*>, <*container*>, <*flexContainer*>, and specializations of <*flexContainer*>: *genericInterworkingService* and *genericInterworkingOperationInstance* are intended to hold data that can be used for data exchange with the IPE.

For Generic interworking a convention is needed how the IPE uses these resources to communicate with other oneM2M entities. This is described in the subsequent clauses.

Resources for RESTful communication style vs. procedure call (RPC) style:

A Generic interworking IPE needs to be able to communicate with non-oneM2M systems that implement some form of RESTful communication style as well as other systems that communicate in a procedure call (RPC) style.

For RESTful systems the use of Input- or OutputDataPoints may be more appropriate.

On the other hand procedure calls can be better modelled using Operations (and their OperationInputs/-Outputs).

Also a combination of both (where Operations additionally receive input from InputDataPoints and/or write output into OutputDataPoints) is possible.

Persistent resources vs. transient resources:

- Persistent resources are *genericInterworkingService*, <*container*>s and <*flexContainer*>s that contain data of Services, Input- or OutputDataPoints. Services, Input- and OutputDataPoints of an Interworked Device usually exist as long as the IPE enables the communication with the Interworked Device.

- Transient resources are *genericInterworkingOperationInstance*s, <*container*>s and <*flexContainer*>s that contain data of Operations, OperationInput or OperationOutput.
  These resources are created and exist as long as the Interworked Device performs execution of an Operation and receive the output data of the Operation. Once the output data have been delivered to subscribed communicating entities transient resources may be deleted by the IPE.

NOTE: While in general the present document assumes that semantic information can be made available (using the <*semanticDescriptor*> resource) the mechanisms described here for IPE communication **do not rely** on the existence of semanticDescriptors. This allows e.g. very simple devices to exchange their data in "raw" form (e.g. as byte-fields that need to be interpreted by the communicating entity).

8.1.2 Parent-child and linking resource relationships

Figure 22 provides an overview of parent-child resource relationships that are used for communication with AEs (in particular the IPE) in the context of Generic interworking.

It involves the:

- Persistent resource types:
  - <*AE*>, <*container*> or <*flexContainer*> - for a oneM2M Device or an Interworked Device.
  - <*container*> - for an Input- or OutputDataPoint.
  - <*flexContainer*> - for an Input- or OutputDataPoint.
  - *genericInterworkingService* specialization of <flexContainer> - for a Service of a oneM2M Device or an Interworked Device.
- Transient resource types:
  - <*container*> - for OperationInput or OperationOutput data of an Operation.
  - <*flexContainer*> - for OperationInput or OperationOutput data of an Operation.
  - *genericInterworkingOperationInstance* specialization of <*flexContainer*> - for an Operation of a Service.

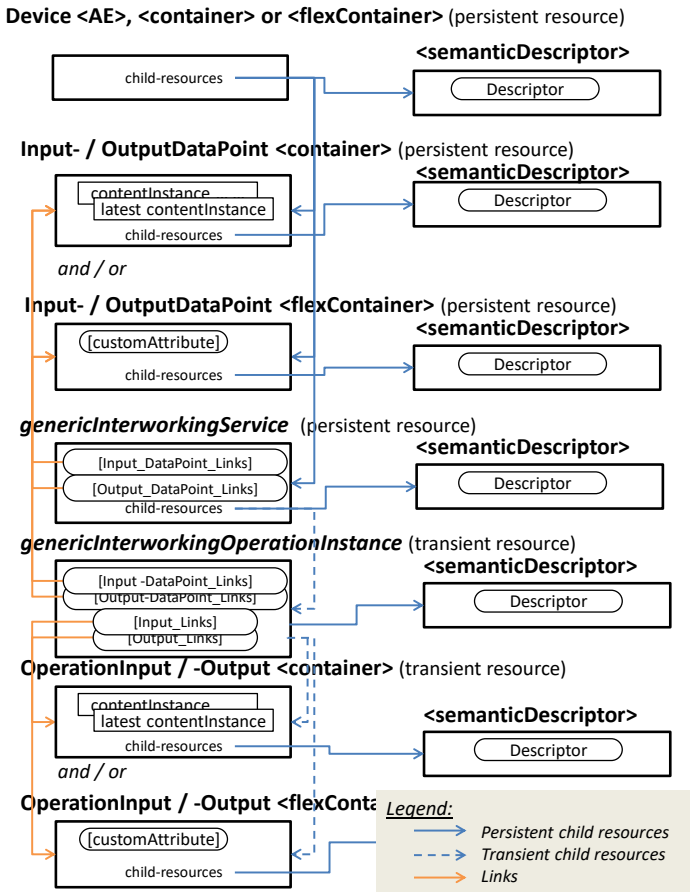

Figure 22: Parent-child and Link relationships in the context of Generic interworking

Parent-child relationships:

- An <*AE*> resource, required for representing a Device, is created by its AE.
  Alternatively, in the case of an Interworked Device, the AE that is the generic interworking IPE may create resources of type <container> or <flexContainer>, that represents the Interworked Device.

- Input- and Output DataPoints (<containers> and/or <flexContainers>) are created by the AE as child resources of its (<AE>, <containers>, <flexContainers>) resource that represents the Device.

- Services (resources of specialization type *genericInterworkingService* of a <flexContainer>) are created by the AE as child resources of its resource that represents the Device.

- OperationInstances (resources of specialization type *genericInterworkingOperationInstance* of a <flexContainer>) are created by the AE or by the communicating entity as child resources of the *genericInterworkingService* of the Service.

- OperationInput (<containers> and/or <flexContainers>) are created by the communicating entity as child resources of the *genericInterworkingOperationInstance* of the Operation instance.

- OperationOutput (<containers> and/or <flexContainers>) are created by the AE as child resources of the *genericInterworkingOperationInstance* of the Operation instance.

- All of the above can contain a <*semanticDescriptor*> as child resource.

Link relationships:

- Services can contain links to:
  - InputDataPoints (contained in the *InputDataPointsLinks* attribute).
  - OutputDataPoints (contained in the *ouputDataPointsLinks* attribute).
- OperationInstances can contain links to:
  - InputDataPoints (contained in the *InputDataPointsLinks* attribute).
  - OutputDataPoints (contained in the *ouputDataPointsLinks* attribute).
  - OperationInputs (contained in the *inputLinks* attribute).
  - OperationOutputs (contained in the *outputLinks* attribute).

8.2 Specification of the IPE for Generic interworking

8.2.1 General functionality of a Generic interworking IPE

Generic interworking supports the interworking variant with full mapping of the semantic of the non-oneM2M data model to Mca as indicated in clause F.2 of oneM2M TS-0001 [2].

The non-oneM2M data model is described in the form of a oneM2M compliant ontology which is derived (as sub-classes and sub-properties) from the oneM2M Base Ontology and may be available in a formal description language (e.g. OWL).

A oneM2M compliant ontology can describe an external technology (e.g. ZigBee) for which a standardized interworking with oneM2M is required or it could describe a model of consensus that is shared by large industry sector (like SAREF, referenced in [i.2]) that facilitates the matching of existing assets (standards/protocols/datamodels/etc.). An IPE that provides Generic interworking with a M2M Area Network shall instantiate the classes, object- and data properties of the ontology describing the non-oneM2M data model of the M2M Area Network as oneM2M resources, according to the instantiation rules of clause 7.1:

- Depending on the capabilities of the IPE and when the ontology describing the non-oneM2M data model is made available as a formal description the IPE may access and parse the OWL file of the ontology to support creation of the required oneM2M resources.

8.2.2 Interworked Device discovery

The IPE shall discover the devices in the non-oneM2M solution or, alternatively, they may be manually configured in the IPE:

1) For each discovered Interworked Device in the non-oneM2M solution the IPE shall:

   - either create a <*container*> or <*flexContainer*> child resource of the IPE's <*AE*> resource for a Proxied Device that represents the non-oneM2M Interworked Device in the oneM2M System; or

   - in the case the IPE provides interworking with a single Interworked Device, the IPE may use its own <*AE*> resource for the Proxied Device that represents the non-oneM2M Interworked Device in the oneM2M System.

2) For each discovered device in the non-oneM2M solution the IPE shall create the Input- and OutputDataPoints (resource types <*container*> and/or <*flexContainer*>) and Services (resource type <*flexContainer*> specialization: <*genericInterworkingService*>) as child resources of the resource of the Proxied Device.

3) The IPE shall create <*semanticDescriptor*>s as child resources of the Input- and OutputDataPoints and Services.

4) The IPE shall subscribe to all created resources.

NOTE: Whether <*AE*>, <*container*> or <*flexContainer*> resource types are used to represent InterworkedDevices and whether <*container*> or <*flexContainer*> resource types are used for input- and OutputDataPoints and operationInputs/-Outputs is not specified and depends on configuration.

8.2.3 Handling of DataPoints by the IPE

- When the IPE receives a request by the interworked non-oneM2M device via the non-oneM2M reference point to write an OutputDataPoint belonging to a Service of the device the IPE shall:

  - de-serialize the received data

  and, depending on the resource type of the OutputDataPoint (<*flexContainer*> or <*container*>)

  - UPDATE/(CREATE contentInstance) the OutputDataPoint resources of the related *genericInterworkingService* with the output data.

- When the IPE receives a request by the interworked non-oneM2M device via the non-oneM2M reference point to read an InputDataPoint belonging to a Service of the device the IPE shall:

  - RETRIEVE data from the InputDataPoint resource of the related *genericInterworkingService*;

  - serialize the data; and

  - return them to the non-oneM2M device.

- When the IPE is notified by the CSE that a <*flexContainer*> or <*container*> child-resource of the Proxied Device has been changed the IPE shall:

  - check to which Service the <*flexContainer*> or <*container*> resource belongs by checking if one of the inputDataPointLinks references the resource as InputDataPoint;

  - read the data of the changed resource; and

  - invoke the Service, parameterized with data of the InputDataPoint, via the non-oneM2M reference point in the interworked non-oneM2M device.

8.2.4 Handling of Operations by the IPE

When the IPE receives notification from the CSE about creation of an OperationInstance resource (resource type *genericInterworkingOperationInstance*) as child resource of a *genericInterworkingService* resource the IPE shall perform the following actions:

1) The IPE shall RETRIEVE the input data of the operation (contained in the resources to which the attributes inputLinks and InputDataPoint Links of *genericInterworkingOperationInstance* provide links).

2) the IPE shall UPDATE the *operationState* attribute of the OperationInstance with the value "data received by application".

3) the IPE shall invoke the related operation together with their input data in the non-oneM2M device via the non-oneM2M reference point.

4) the IPE shall handle the result of the operation, received from the Interworked Device via the non-oneM2M reference point:

   - If the non-oneM2M device is capable of processing the operation (i.e. no error is reported over the non-oneM2M reference point) then:

     a) The IPE shall UPDATE the *operationState* attribute of the OperationInstance with the value "data transmitted to interworked device".

     b) The IPE shall UPDATE the *expirationTime* attribute to an appropriate value that allows the Interworked Device to execute the operation and allows the subscribed communicating entities to get notified and potentially retrieve the results.

     c) When the IPE receives output data from the operation in the non-oneM2M device via the non-oneM2M reference point the IPE shall de-serialize these data and update, depending on the operation specification, the operationOutput resources and/or the OutputDataPoint resources with the output data:

        - When the received output data from the operation contain a state indication (according to the OperationState class of the ontology) then the IPE may UPDATE the *operationState* attribute with the value received in the state indication.

        - When the received output data from the operation contains no state indication (according to the OperationState class of the ontology) then the IPE shall UPDATE the *operationState* attribute with the value "operation ended".

        - In case the operation contains no output data and the non-oneM2M reference point does not contain a state indication then the IPE shall UPDATE the operationState attribute of the OperationInstance with the value "operation ended".

        When an error occurs during communication over the non-oneM2M reference point then the IPE shall UPDATE the *operationState* attribute with the value "operation failed".

   - If the non-oneM2M device is not capable of processing the operation (i.e. an error is reported over the non-oneM2M reference point) then the IPE shall DELETE the OperationInstance resource.

When the IPE receives unsolicited data through an operation in the non-oneM2M device via the non-oneM2M reference point (e.g. when the device reacts on some external event and publishes related output data) the IPE shall de-serialize these data and perform the following actions:

1) Creation of OperationOutputs and OutputDataPoints of the Operation by the IPE:

   - For all Operation parameters that are (transient) OperationOutputs the IPE shall CREATE <*container*>s and/or <*flexContainer*>s that contain the data for the OperationOutputs of the Operation.

   - For all Operation parameters that are (persistent) OutputDataPoints the IPE shall CREATE <*contentInstance*>s of <*container*>s and/or UPDATE <*flexContainer*>s that contain data for the OutputDataPoints of the Operation. *outputDataPointLinks.*

2) The IPE shall CREATE a *genericInterworkingOperationInstance* resource as child-resource of the *genericInterworkingService* resource that represents the Service of the Operation.
The IPE shall:

   a) make the <*container*>s and/or <*flexContainer*>s that contain the data for the OperationOutput child-resources of the *genericInterworkingOperationInstance* resource;

   b) set the *outputDataPointLinks* attribute (with the OutputDataPoint names, links to <*container*>s and/or <*flexContainer*>s for the OutputDataPoints and, if needed, attributeNames);

   c) set the *outputLinks* attribute (with the OperationOutput names, links to <*container*>s and/or <*flexContainer*>s for the OperationOutput, and if needed attributeNames).

3) The IPE shall CREATE <*semanticDescriptor*> resources to all created resources and fill the *descriptor* attribute with RDF data.

4) The IPE shall set the *expirationTime* attribute of the *genericInterworkingOperationInstance* to an appropriate value that allows communicating entities (that had subscribed to the *genericInterworkingService* resource and were notified about the creation of the *genericInterworkingOperationInstance* resource) to retrieve the *genericInterworkingOperationInstance* and its OperationOutput child-resources.

5) The IPE shall set the *operationState* attribute of the *genericInterworkingOperationInstance* resource:

   - When the received output data from the non-oneM2M device operation contains a state indication (according to the OperationState class of the ontology) then the IPE may UPDATE the *operationState* attribute with the value received in the state indication.

   - When the received output data from the non-oneM2M device operation contain no state indication (according to the OperationState class of the ontology) then the IPE shall UPDATE the *operationState* attribute with the value "operation ended".

At periodic, implementation specific, times the IPE shall check the *expirationTime* attribute of all Operation resources of all Proxied Devices and DELETE expired Operations and their OperationInputs and -Outputs.

8.2.5 Removing Devices

When a Interworked Device in the non-oneM2M solution becomes unavailable the IPE shall delete the resource for its Proxied Device and all its related DataPoint, Service and Operation resources.

8.3 Specification of the behaviour of a communicating entity in message flows between IPE and the communicating entity

8.3.1 Preconditions on the communicating entity

1) Any communicating entity, that wants to communicate with:

   a) An interworked non-oneM2M device via the IPE needs to be subscribed to the <*AE*> resource of the IPE to get notified about resources for Proxied Device that are created by the IPE to represent interworked non-oneM2M devices that were discovered by the IPE.

   b) A specific interworked non-oneM2M device via the IPE needs to be subscribed to the <*container*> or <*flexContainer*> or <*AE*> resource that had been created by the IPE as a related Proxied Device to represent the interworked non-oneM2M device.

2) The communicating entity needs also be subscribed to:

   a) The *genericInterworkingService* resources that have been created by the IPE as child resources of the resource of the Proxied Device.

   b) <*container*> or <*flexContainer*> resources that have been created by the IPE as child resources of the Proxied Device to represent (persistent) Input- or OutpuDataPoints of the *genericInterworkingService* resources.

8.3.2 Flow from the communicating entity to the IPE using InputDataPoints of a Service

8.3.2.1 Flow from the communicating entity to the IPE using a <*container*> type InputDataPoint

1) When the communicating entity wants to invoke a Service in the interworked non-oneM2M device it shall determine the *genericInterworkingService* resource that is related to the Service by checking the *serviceName* attribute, which contains the class name of the Service in the related compliant ontology.

2) The communicating entity determines the <*container*> or <*flexContainer*> that is related to the InputDataPoint from the *InputDataPoint Links* attribute of the *genericInterworkingService* resource, which contains references to the InputDataPoints of the Service as a list of triples.
The first field of the triple identifies the InputDataPoint in the related compliant ontology, the second field contains the URI of the resource (container or flexContainer) that holds the data of the InputDataPoint.
The third field indicates whether the InputDataPoint contains simple data or the InputDataPoint contains complex data:

   - If the InputDataPoint is of type <*container*> and contains simple data the third field contains the text string "latest".

   - If the InputDataPoint is of type <*container*> and contains complex data the third field is empty.

3) The communicating entity shall update the InputDataPoint:

   - If the InputDataPoint contains simple data then the communicating entity CREATEs a new <contentInstance> of the InputDataPoint.

   - If the InputDataPoint contains complex data, (contained in child-resources: <*container*> or <*flexContainer*>) then the communicating entity UPDATEs the child-<*flexContainer*>s and/or CREATEs new <contentInstance>s of child-<*container*>s as needed.

   - If the InputDataPoint contains complex data the communicating entity may also CREATE or DELETE child-resources of the InputDataPoint <*container*> as needed. In this case the communicating entity shall create <*subscription*>s to all created resources that notify the IPE.
     When a new child resource of the InputDataPoint resource is created then the communicating entity may optionally also create a <*semanticDescriptor*> child resource of the newly created resource:

      a) The *descriptor* attribute of the <*semanticDescriptor*> shall be updated with the RDF description of the created instance of class:Variable.

      b) The *descriptor* attribute of the parent <*semanticDescriptor*> shall be updated with an instance of the "hasSubStructure" object property.

      c) The *descriptor* attribute of the parent <*semanticDescriptor*> shall be updated with an instance of the *resourceDescriptorLink* annotation property with the URI of the new <*semanticDescriptor*> resource.

   - If only child-resources of an InputDataPoint have changed the communicating entity shall issue a null UPDATE (i.e. containing no attributes) on the InputDataPoint on order to make sure the IPE gets notified by the CSE that the InputDataPoint or its child-resources have been changed.

8.3.2.2 Flow from the communicating entity to the IPE using a <*flexContainer*> type InputDataPoint

1) The communicating entity determines the *genericInterworkingService* resource as in clause 9.2.

2) The communicating entity determines the <*container*> or <*flexContainer*> that is related to the InputDataPoint.
In the case of a <flexContainer> type InputDataPoint the third field indicates whether the InputDataPoint contains simple data - in this case the third field contains a text string with the name of the name of the *[customAttribute]* (which is identical to the name of the InputDataPoint) - or the InputDataPoint contains complex data - in this case the third field is empty.

3) The communicating entity updating the InputDataPoint:

   a) If the InputDataPoint contains simple data then the communicating entity UPDATEs the InputDataPoint with a new value for the *[customAttribute]*.

   b) If the InputDataPoint contains complex data then the communicating entity shall behave as in clause 8.3.2.1 step 3).

8.3.3 Flow from the IPE to the communicating entity using OutputDataPoints of a Service

When the communicating entity is notified by the CSE that a child-resource of the Proxied Device has been changed the IPE shall:

a) Identify the Service to which the <*flexContainer*> or <*container*> resource belongs by checking which one of the *genericInterworkingService* resources contains an outputDataPointLinks attribute that references the resource as OutputDataPoint.

e) read the data of the <*flexContainer*> or <*container*> resource (and possibly its child-resources) and use them in the context of the service to which they belong.

8.3.4 Flow from the communicating entity to the IPE using Operations of a Service

1) If the Operation is parameterized by input parameter that are (transient) OperationInputs the communicating entity shall CREATE <*container*>s and/or <*flexContainer*>s that contain the data for the OperationInputs of the Operation.

2) If the Operation is parameterized by input parameter that are (persistent) InputDataPoints the communicating entity may CREATE <*contentInstance*>s of <*container*>s and/or UPDATE <*flexContainer*>s that contain data for the InputDataPoints of the Operation.

3) The communicating entity shall CREATE a *genericInterworkingOperationInstance* resource as child-resource of the *genericInterworkingService* resource that represents the Service of the Operation.
The communicating entity shall:

   - make the <*container*>s and/or <*flexContainer*>s that contain the data for the OperationInput child-resources of the *genericInterworkingOperationInstance* resource;

   - set the *inputDataPointLinks* attribute (with the InputDataPoint names, links to <*container*>s and/or <*flexContainer*>s for the InputDataPoints, and if needed Attributenames);

   - set the *inputLinks* attribute (with the OperationInput names, links to <*container*>s and/or <*flexContainer*>s for the OperationInput, and if needed Attributenames).

4) The communicating entity may CREATE <*semanticDescriptor*> resources to all created resources and fill the *descriptor* attribute with RDF data.

5) The communicating entity shall CREATE a subscription to the *genericInterworkingOperationInstance* resource in order to get notified about changes of the OperationState and potential creation of OperationOutput <*container*> and/or <*flexContainer*> child resources of the *genericInterworkingOperationInstance*.

6) Since the IPE has subscribed to the *genericInterworkingService* resource it gets notified about the creation of a *genericInterworkingOperationInstance* child-resource and retrieves the resource and its OperationInputs and InputDataPoints.

8.3.5 Flow from the IPE to the communicating entity using Operations of a Service

Since the communicating entity is subscribed to the *genericInterworkingService* resources of the Proxied Device it gets notified by the CSE when the IPE creates a *genericInterworkingOperationInstance* as child-resource of the *genericInterworkingService*:

1) The communicating entity needs to retrieve the *genericInterworkingOperationInstance.*

2) As the *genericInterworkingOperationInstance* contains *outputDataPointLinks* and *outputLinks* attributes the communicating entity receives information about output data of the operation and can retrieve the referenced <*container*> and/or <*flexContainer*> resources.

9 FlexContainer specializations for Generic interworking

9.1 Introduction

For Ontology based Interworking two specialization types of <flexContainer> are needed: *genericInterworkingService* and *genericInterworkingOperationInstance.*

9.2 Resource Type *genericInterworkingService*

Resource type *genericInterworkingService* is used for grouping Input- and/or Output Datapoints and/or OperationInstances of a Service. For Ontology based Interworking Input- and/or Output Datapoints and/or OperationInstances can be grouped as a Service with respect to their usage within a single Device.

A resource of type *genericInterworkingService* contains references to the <container> or <flexContainer> resources that represent Input- and/or Output Datapoints of the Service in the inputDataPointLinks and outputDataPointLinks attributes.

A resource of type *genericInterworkingService* is also the parent resource of genericInterworkingOperationInstances for that Service.

A resource of type *genericInterworkingService* can be a child-resource of:

a) *AE, container, flexContainer* since Ontology based Interworking allows these three resource types to represent Devices and InterworkeDevices.

b) *genericInterworkingService* since Ontology based Interworking allows Services to contain (sub-)Services.

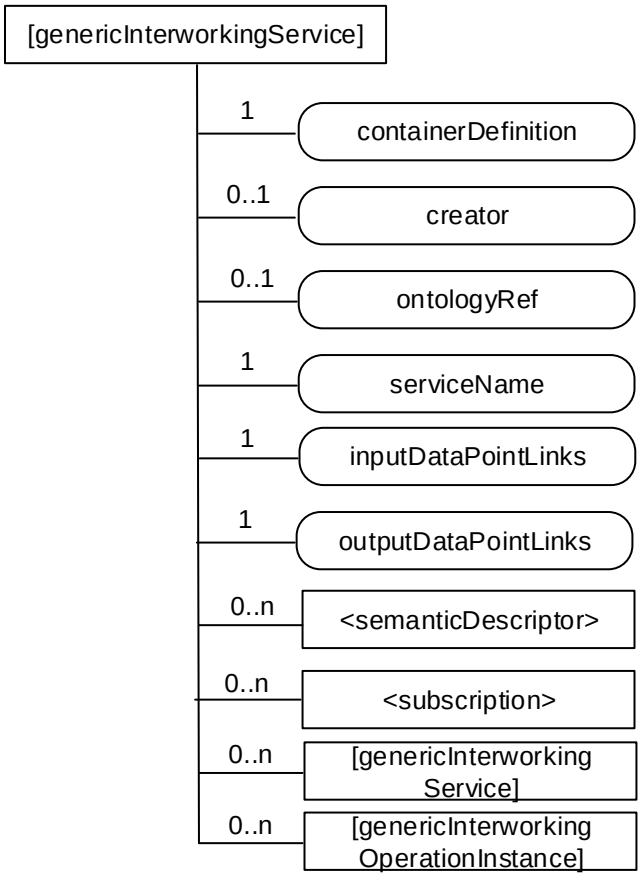

Figure 23: Structure of [genericInterworkingService] resource

The *[genericInterworkingService]* resource shall contain the child resource specified in table 3.

Table 3: Child resources of [genericInterworkingService] resource

Child Resources of *[genericInterworking Service]*	**Child Resource Type**	**Multiplicity**	**Description**	***[genericInterworkingServiceAnnc]* Child Resource Type**
semanticDescriptor	*<semanticDescriptor>*	0..n	See clause 9.6.30 in oneM2M TS-0001 [2]	*<semanticDescriptor>, <semanticDescriptorAnnc>*
[variable]	*<subscription>*	0..n	See clause 9.6.8 in oneM2M TS-0001 [2]	*<subscription>*
[variable]	*<flexContainer> specialization: [genericInterworking Service]*	0..n	A Service may be composed of (sub)-Services that are contained as child-resources	*[genericInterworkingService] [genericInterworkingServiceAnnc]*
[variable]	*<flexContainer> specialization: [genericInterworking OperationInstance]*	0..n	See clause 9.3 For each invocation of an operation of a Service a child-resource of type *[genericInterworkingOperationInstance]* is created. When the operation is finished this child-resource is deleted by the IPE	*[genericInterworkingOperationInstance] [genericInterworkingOperationInstance Annc]*

The *[genericInterworkingService]* resource shall contain the attributes specified in table 4.

Table 4: Attributes of [genericInterworkingService] resource

Attributes of *[genericInterworking Service]*	Multiplicity	RW/ RO/ WO	Description	*[genericInterworkingService Annc]* Attributes
resourceType	1	RO	See clause 9.6.1.3 in oneM2M TS-0001 [2]	NA
resourceID	1	RO	See clause 9.6.1.3 in oneM2M TS-0001 [2]	NA
resourceName	1	WO	See clause 9.6.1.3 in oneM2M TS-0001 [2]	NA
parentID	1	RO	See clause 9.6.1.3 in oneM2M TS-0001 [2]	NA
accessControlPolicyIDs	0..1 (L)	RW	See clause 9.6.1.3 in oneM2M TS-0001 [2]	MA
labels	0..1 (L)	RW	See clause 9.6.1.3 in oneM2M TS-0001 [2]	MA
stateTag	1	RO	See clause 9.6.1.3 in oneM2M TS-0001 [2]	OA
announceTo	0..1 (L)	RW	See clause 9.6.1.3 in oneM2M TS-0001 [2]	NA
announcedAttribute	0..1 (L)	RW	See clause 9.6.1.3 in oneM2M TS-0001 [2]	NA
dynamicAuthorizationConsultationIDs	0..1 (L)	RW	See clause 9.6.1.3 in oneM2M TS-0001 [2]	OA
containerDefinition	1	WO	See clause 9.6.1.2.2 in oneM2M TS-0001 [2] The value shall be "org.onem2m. genericInterworkingService"	MA
creator	0..1	RO	See clause 9.6.35 in oneM2M TS-0001 [2]	NA
ontologyRef	0..1	RW	See clause 9.6.35 in oneM2M TS-0001 [2]	OA
serviceName	1	RW	The attribute contains the name of the Service. The name of the Service is given by the class name of that Service in the used ontology (which needs to be derived from the Base Ontology)	MA
inputDataPointLinks	0..1	RW	This attribute contains a list of triples, each triple containing the following fields: 1. A text string with the name of an inputDatapoint of the Service 2. A URI of the resource (container or flexContainer) that holds the data of the inputDataPoint 3. A field for identifying simple-type data If the inputDataPoint contains simple-type data then: i. If the resource type of the inputDataPoint is <container> then this field shall contain the text string "latest" ii. If the resource type of the inputDataPoint is <flexContainer> then this field shall contain the name of the [customAttribute] (which is identical to the name of the inputDataPoint) If the inputDataPoint contains complex-type data then this field shall remain empty.	MA

Attributes of *[genericInterworking Service]*	Multiplicity	RW/ RO/ WO	Description	*[genericInterworkingService Annc]* Attributes
outputDataPointLinks	0..1	RW	This attribute contains a list of triples, each triple containing the following fields: 1. A text string with the name of an outputDatapoint of the Service 2. A URI of the resource (container or flexContainer) that holds the data of the outputDataPoint 3. A field for identifying simple-type data If the outputData:Point contains simple-type data then i. If the resource type of the outputDataPoint is <container> then this field shall contain the text string "latest" ii. If the resource type of the outputDataPoint is <flexContainer> then this field shall contain the name of the [customAttribute] (which is identical to the name of the outputDataPoint) Otherwise, if the outputDataPoint contains complex-type data then this field shall remain empty.	MA

9.3 Resource Type *genericInterworkingOperationInstance*

In the context of Ontology based Interworking resources of resource type *genericInterworkingOperationInstance* are created as child-resources of a Service by the CSE. The originator of a request can be:

- the AE (for AE initiated communication for notifying communicating entities);
- a communicating entity (to notify the AE about an operation that needs to be performed by the AE and to receive output back from the AE).

After the expirationTime the AE may delete the operationInstance and all linked operationInput and operationOutput resources (contained in the references in attributes: inputLinks and outputLinks).

An OperationInstance resource holds in its attributes *inputDataPointLinks* and *inputLinks* references to resources of type <container> and <flexContainer> from which the AE should retrieve input of the operation. Similarly the attributes *outputDataPointLinks* and *outputLinks* references to resources of type <container> and <flexContainer> to which the AE should write its output of the operation.

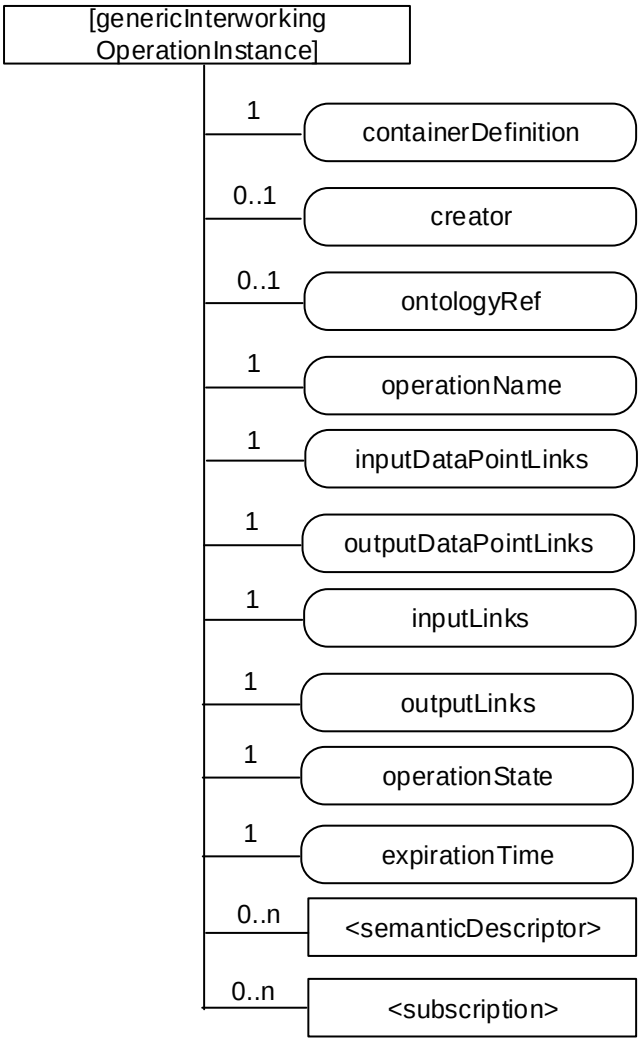

Figure 24: Structure of [genericInterworkingOperationInstance] resource

The *[genericInterworkingOperationInstance]* resource shall contain the child resource specified in table 5.

Table 5: Child resources of [genericInterworkingOperationInstance] resource

Child Resources of *[genericInterworkingOperationInstance]*	**Child Resource Type**	**Multiplicity**	**Description**	***[genericInterworkingOperationInstanceAnnc]* Child Resource Type**
semanticDescriptor	*<semanticDescriptor>*	0..n	See clause 9.6.30 in oneM2M TS-0001 [2]	*<semanticDescriptor>*, *<semanticDescriptorAnnc>*
[variable]	*<subscription>*	0..n	See clause 9.6.8 in oneM2M TS-0001 [2]	*<subscription>*

The *[genericInterworkingOperationInstance]*resource shall contain the attributes specified in table 6.

Table 6: Attributes of [genericInterworkingOperationInstance] resource

Attributes of *[genericInterworking OperationInstance]*	Multiplicity	RW/ RO/ WO	Description	*[genericInterworkingOperation InstanceAnnc]* Attributes
resourceType	1	RO	See clause 9.6.1.3 in oneM2M TS-0001 [2]	NA
resourceID	1	RO	See clause 9.6.1.3 in oneM2M TS-0001 [2]	NA
resourceName	1	WO	See clause 9.6.1.3 in oneM2M TS-0001 [2]	NA
parentID	1	RO	See clause 9.6.1.3 in oneM2M TS-0001 [2]	NA
expirationTime	1	RW	See clause 9.6.1.3 in oneM2M TS-0001 [2] This attribute shall contain the time after which the operationInstance and its operationInput and operationOutput resources may be deleted by the AE. If an AE got notified about creation of the operationInstance and if the AE accepts to process the operation (i.e. does not immediately delete the operationInstance) the expirationTime is set by the AE.	MA
accessControlPolicyIDs	0..1 (L)	RW	See clause 9.6.1.3 in oneM2M TS-0001 [2]	MA
labels	0..1 (L)	RW	See clause 9.6.1.3 in oneM2M TS-0001 [2]	MA
creationTime	1	RO	See clause 9.6.1.3 in oneM2M TS-0001 [2]	NA
lastModifiedTime	1	RO	See clause 9.6.1.3 in oneM2M TS-0001 [2]	NA
stateTag	1	RO	See clause 9.6.1.3 in oneM2M TS-0001 [2]	OA
announceTo	0..1 (L)	RW	See clause 9.6.1.3 in oneM2M TS-0001 [2]	NA
announcedAttribute	0..1 (L)	RW	See clause 9.6.1.3 in oneM2M TS-0001 [2]	NA
dynamicAuthorizationConsultationIDs	0..1 (L)	RW	See clause 9.6.1.3. in oneM2M TS-0001 [2]	OA
containerDefinition	1	WO	See clause 9.6.1.2.2 in oneM2M TS-0001 [2] The value shall be "org.onem2m. genericInterworkingOperationInstance"	MA
creator	0..1	RO	See clause 9.6.35 in oneM2M TS-0001 [2]	NA
ontologyRef	0..1	RW	See clause 9.6.35 in oneM2M TS-0001 [2]	OA
operationName	1	RW	The attribute contains the name of the Operation. The name of the Operation is given by the class name of that Operation in the used ontology (which needs to be derived from the Base Ontology)	MA
operationState	1	RW	This attribute contains a text string that indicates how far the operation has progressed. specified values are: • "data_received_by_application" • "operation_ended" • "operation_failed" • "data_transmitted_to_interworked_ device" Additional, application specific values for the text string of the operationState attribute are permissible.	MA

Attributes of *[genericInterworking OperationInstance]*	Multiplicity	RW/ RO/ WO	Description	*[genericInterworkingOperation InstanceAnnc]* Attributes
inputDataPointLinks	0..1	RW	This attribute contains a list of triples, each triple containing the following fields: 1. A text string with the name of an inputDatapoint of the operationInstance 2. A URI of the resource (container or flexContainer) that holds the data of the inputDataPoint 3. A field for identifying simple-type data If the inputDataPoint contains simple-type data then: i. If the resource type of the inputDataPoint is <container> then this field shall contain the text string "latest" ii. If the resource type of the inputDataPoint is <flexContainer> then this field shall contain the name of the [customAttribute] (which is identical to the name of the inputDataPoint) If the inputDataPoint contains complex-type data then this field shall remain empty.	MA
outputDataPointLinks	0..1	RW	This attribute contains a list of triples, each triple containing the following fields: 1. A text string with the name of an outputDatapoint of the OperationInstance 2. A URI of the resource (container or flexContainer) that holds the data of the outputDataPoint 3. A field for identifying simple-type data If the outputDataPoint contains simple-type data then: i. If the resource type of the outputDataPoint is <container> then this field shall contain the text string "latest" ii. If the resource type of the outputDataPoint is <flexContainer> then this field shall contain the name of the [customAttribute] (which is identical to the name of the outputDataPoint) If the outputDataPoint contains complex-type data then this field shall remain empty.	MA

Attributes of *[genericInterworking OperationInstance]*	Multiplicity	RW/ RO/ WO	Description	*[genericInterworkingOperation InstanceAnnc]* Attributes
inputLinks	0..1	RW	This attribute contains a list of triples, each triple containing the following fields: 1. A text string with the name of an operationInput of the operationInstance 2. A URI of the resource (container or flexContainer) that holds the data of the operationInput 3. A field for identifying simple-type data If the operationInput contains simple-type data then: i. If the resource type of the operationInput is <container> then this field shall contain the text string "latest" ii. If the resource type of the operationInput is <flexContainer> then this field shall contain the name of the [customAttribute] (which is identical to the name of the operationInput) If the Input contains complex-type data then this field shall remain empty.	MA
outputLinks	0..1	RW	This attribute contains a list of triples, each triple containing the following fields: 1. A text string with the name of an operationOutput of the operationInstance 2. A URI of the resource (container or flexContainer) that holds the data of the outputDataPoint 3. A field for identifying simple-type data If the operationOutput contains simple-type data then i. If the resource type of the operationOutput is <container> then this field shall contain the text string "latest" ii. If the resource type of the operationOutput is <flexContainer> then this field shall contain the name of the [customAttribute] (which is identical to the name of the operationOutput) If the operationOutput contains complex-type data then this field shall remain empty.	MA

Annex A (normative):
OWL representation of Base Ontology

The OWL representation of Base Ontology is provided in a separate document.

The Base Ontology is available at the web page:

- http://www.onem2m.org/technical/onem2m-ontologies.

which contains the latest version of the (RDF/XML) OWL representation of the ontology and individual versions of the (RDF/XML) OWL representation of the ontology under the URL:

- http://www.onem2m.org/ontology/Base_Ontology/base_ontology-vx_y_z
  (where x,y,z signify the version numbering for major- minor- and editorial changes of the base ontology).

e.g. http://www.onem2m.org/ontology/Base_Ontology/base_ontology-v2_2_0.

Annex B (informative): Mappings of selected external ontologies to the Base Ontology

B.1 Mapping of SAREF

B.1.1 Introduction to SAREF

The following description of the SAREF ontology is copied from the following site: https://sites.google.com/site/smartappliancesproject/ontologies/reference-ontology.

It provides an introduction to the scope and objectives of SAREF ontology:

"*The Smart Appliances REFerence (SAREF) ontology is a shared model of consensus that facilitates the matching of existing assets (standards/protocols/datamodels/etc.) in the smart appliances domain. The SAREF ontology provides building blocks that allow separation and recombination of different parts of the ontology depending on specific needs.*

The starting point of SAREF is the concept of Device (e.g., a switch). Devices are tangible objects designed to accomplish one or more functions in households, common public buildings or offices. The SAREF ontology offers a lists of basic functions that can be eventually combined in order to have more complex functions in a single device. For example, a switch offers an actuating function of type "switching on/off". Each function has some associated commands, which can also be picked up as building blocks from a list. For example, the "switching on/off" is associated with the commands "switch on", "switch off" and "toggle". Depending on the function(s) it accomplishes, a device can be found in some corresponding states that are also listed as building blocks.

A Device offers a Service, which is a representation of a Function to a network that makes the function discoverable, registerable and remotely controllable by other devices in the network. A Service can represent one or more functions. A Service is offered by a device that wants (a certain set of) its function(s) to be discoverable, registerable, remotely controllable by other devices in the network. A Service must specify the device that is offering the service, the function(s) to be represented, and the (input and output) parameters necessary to operate the service.

A Device in the SAREF ontology is also characterized by an (Energy/Power) Profile that can be used to optimize the energy efficiency in a home or office that are part of a building."

A formal description of the ontology is provided via this link: http://ontology.tno.nl/saref/.

The following table B.1 provides a list of SAREF concepts - source http://www.etsi.org/images/files/Events/2015/201502_SMARTAPP/D-S4 - SMART 2013-0077 - Smart Appliances - Final Study Report_v1.0.pdf.

Table B.1: List of SAREF concepts

Concept	Definition
Building Object	A Building Object is an object in the building that can be controlled by devices, such as a door or a window that can be automatically opened or closed by an actuator.
Building Space	According to FEIMSER, a Building Space in SAREF defines the physical spaces of the building. A building space contains devices or building objects.
Command	A Command is a directive that a device should support to perform a certain function. A command may act upon a state, but does not necessarily act upon a state. For example, the ON command acts upon the ON/OFF state, but the GET command does not act upon any state, since it gives a directive to retrieve a certain value with no consequences on states.
Commodity	A Commodity is a marketable item for which there is demand, but which is supplied without qualitative differentiation across a market. SAREF refers to energy commodities such as electricity, gas, coal and oil.
Device	A Device in the context of the Smart Appliances study is a tangible object designed to accomplish a particular task in households, common public buildings or offices. In order to accomplish this task, the device performs one or more functions. For example, a washing machine is designed to wash (task) and to accomplish this task it performs the start and stop function.
Device Category	A Device Category provides a way to classify devices according to a certain point of view, for example, the point of view of the user of the device vs. the device's manufacturer, or the domain in which the device is used (e.g., smart appliances vs. building domain vs. smart grid domain), etc.
Function	A Function represents the particular use for which a Device is designed. A device can be designed to perform more than one function.
Function Category	A Function Category provides a way to classify functions according to a certain point of view, for example, considering the specific application area for which a function can be used (e.g., light, temperature, motion, heat, power, etc.), or the capability that a function can support (e.g., receive, reply, notify, etc.), and so forth.
Profile	A Profile characterizes a device for the purpose to optimize the energy efficiency in the home or office in which the device is located. The saref:Profile class allows to describe the energy (or power) production and consumption of a certain device using the saref: hasProduction and saref:hasConsumption properties. This production and consumption can be calculated over a time span (the saref:hasTime property) and, eventually, associated to some costs (the saref:hasPrice property).
Property	A Property is anything that can be sensed, measured or controlled in households, common public buildings or offices.
Service	A Service is a representation of a function to a network that makes the function discoverable, registerable, remotely controllable by other devices in the network. A service can represent one or more functions. A Service is offered by a device that wants (a certain set of) its function(s) to be discoverable, registerable, remotely controllable by other devices in the network. A Service should specify the device that is offering the service, the function(s) to be represented, and the (input and output) parameters 144 necessary to operate the service.
State	A State represents the state in which a device can be found, e.g. ON/OFF/STANDBY, or ONLINE/OFFLINE, etc.
Task	A Task represents the goal for which a device is designed (from a user perspective). For example, a washing machine is designed for the task of cleaning.
Unit of Measure	The Unit of Measure is a standard for measurement of a quantity, such as a Property. For example, Power is a property and Watt is a unit of power that represents a definite predetermined power: when 10 Watt is mentioned, it actually means 10 times the definite predetermined power called "watt". Our definition is based on the definition of unit of measure in the Ontology of units of Measure (OM). A list of some units of measure that are relevant for the purpose of the Smart Appliances ontology is proposed here, but this list can be extended.

B.1.2 Class mapping relationship between SAREF and the Base Ontology

This clause provides an example on how the principle specified in clause 7.2 can be applied to the mapping between the Base Ontology and SAREF.

Using two different ontologies (i.e. Base_Ontology.owl and SAREF.owl), a new ontology can be created using the mapping principles, i.e. adding mapping relationships into the union of these two ontologies as a new mapped ontology.

As a simple case, the following illustrates the subClass mapping between oneM2M:Device and saref:Device. The following shows that saref:Device is a subclass of oneM2M:Device.

There are two popular OWL syntax: OWL/XML or RFD/XML.

OWL/XML syntax for sub-class mapping is:

```
    <Import>http://www.onem2m.org/ontology/Base_Ontology</Import>
    <Import>http://ontology.tno.nl/saref</Import>

    <SubClassOf>
        <Class IRI="http://ontology.tno.nl/saref#Device"/>
        <Class IRI="http://www.onem2m.org/ontology/Base_Ontology#Device"/>
    </SubClassOf>

RFD/XML syntax for subclass mapping is:
   <rdf:Description rdf:about="http://ontology.tno.nl/saref#Device">
        <rdfs:subClassOf rdf:resource="http://www.onem2m.org/ontology/Base_Ontology#Device"/>
    </rdf:Description>
```

Then, all instances of saref:Device are instances of oneM2M:Device.

Figure B.1 shows the class hierarchy of the new mapped ontology with mapping saref:Device as subClass of oneM2M:Device.

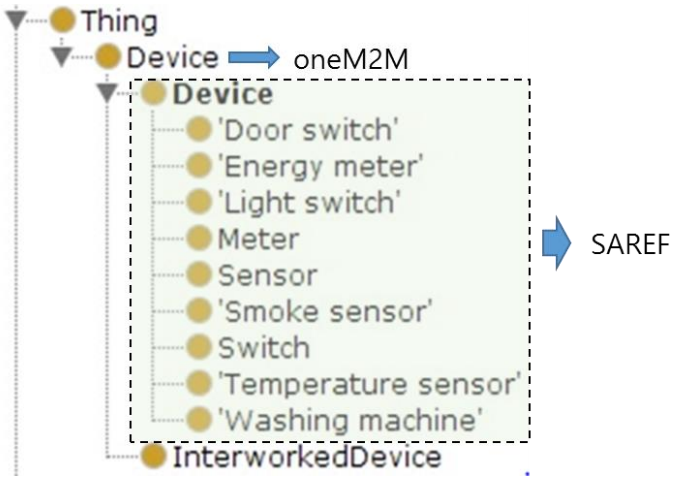

Figure B.1: Class hierarchy of a mapped ontology with mapping saref:Device as subClass of oneM2M:Device

Considering both the definition and the relations of the classes in SAREF and the base ontology, Table B.2 gives the possible subclass mapping between SAREF and the base ontology.

Table B.2: Sub-class mapping between SAREF and the base ontology

Class in SAREF	Mapping relationship	Class in Base Ontology
saref:BuildingObject	rdfs:subClassOf	oneM2M:Thing
saref: BuildingSpace	rdfs:subClassOf	oneM2M:Thing
saref :Commodity	rdfs:subClassOf	oneM2M:Thing
saref :Property	rdfs:subClassOf	oneM2M:InputDataPoint OR oneM2M:OutputDataPoint
saref :UnitOfMeasure	rdfs:subClassOf	oneM2M:MetaData
saref :MeteringFunction	rdfs:subClassOf	oneM2M:MesuringFunction
saref :State	rdfs:subClassOf	oneM2M:InputDataPoint OR oneM2M:OutputDataPoint
saref:Profile,	rdfs:subClassOf	oneM2M:Thing
saref:Task	rdfs:subClassOf	oneM2M:ThingProperty
saref:DeviceCategory	rdfs:subClassOf	oneM2M:ThingProperty
saref:FunctionCategory	rdfs:subClassOf	oneM2M:Aspect
NOTE: Mapping SAREF classes into sub-classes of the Base Ontology is only relevant for the oneM2M System as this sub-class relationship enables a standardized instantiation of these SAREF classes as oneM2M resources.		

For the purpose of semantic discovery also sub-classing some Base Ontology classes to SAREF classes can be done.

Thus oneM2M Devices, Functions, Services … for which instances (oneM2M resources) exist in a oneM2M Solution can be discovered as SAREF instances if these resources are annotated with their semantic description in the appropriate <*semanticDescriptor*> child resources and if the semantic description is made available to SAREF.

Mutual sub-class relationships (rdfs:subClassOf) result in equivalent classes (owl:equivalentClass), i.e. any instance of one class is also an instance of the equivalent class.

Table B.3: Equivalence-class mapping between SAREF and the base ontology

Class in SAREF	Mapping relationship	Class in Base Ontology
saref:Device	owl:equivalentClass	oneM2M:Device
saref:Command	owl:equivalentClass	oneM2M:Command
saref:Function	owl:equivalentClass	oneM2M:Function
saref:Service	owl:equivalentClass	oneM2M:Service
saref:ActuatingFunction	owl:equivalentClass	oneM2M:ControllingFunction
saref:SensingFunction	owl:equivalentClass	oneM2M:MesuringFunction

Object Properties of the base ontology, for which domain- and range classes have equivalent classes in the SAREF ontology, can have equivalent Object Properties in SAREF.

Table B.4: Equivalent Property mapping between SAREF and the base ontology

Object Property in SAREF	Mapping relationship	Object Property in Base Ontology
saref:offers	owl:equivalentProperty	oneM2M:hasService
saref:consistsOf	owl:equivalentProperty	oneM2M:consistsOf
saref:represents	owl:equivalentProperty	oneM2M:exposesFunction
saref:hasCommand	owl:equivalentProperty	oneM2M:hasCommand

B.1.3 Mapping SAREF to oneM2M resource structure

B.1.3.1 Introduction

Mapping an ontology to oneM2M describes how an instance of that ontology may be represented under oneM2M resource structure. This clause proposes a recommended way to map SAREF to oneM2M.

B.1.3.2 Mapping rules

Mapping ontologies to the oneM2M resource structure may be provided through a list of mapping rules. The oneM2M Base Ontology instantiation rules in clause 7.1 apply. Currently no additional mapping rules apply.

B.1.3.3 Example showing the uses of the semanticDescriptor resource and instantiation in the oneM2M resource structure

This clause gives an example of how oneM2M resources and their semantic annotations based on the Smart Appliance REFerence Ontology (SAREF) [i.2] can be used to describe a device representing a smart appliance.

It assumes that the Instantiation rules in clause 7 and the sub-class mapping relationship between SAREF and the Base Ontology (clause B.1.2) apply.

The example taken from SAREF is a (simplified) washing machine:

- The washing machine has been manufactured by manufacturer **XYZ**.
- XYZ describes this type of washing machine as "`Very cool Washing Machine`".
- The model of the type of washing machine is **XYZ_Cool.**
- The washing machine has an actuating function: **WashingFunction** which has three commands:
  - **ON_Command.**
  - **OFF_Command.**
  - **Toggle_Command.**
- The related service of the washing machine that represents that actuating function is of class: **SwitchOnService** from SAREF. It has:
  - an InputDataPoint: **BinaryInput** (to expose command ON_Command and OFF_Command); and
  - an Operation: **ToggleBinary** (to expose command Toggle_Command).
- The washing machine has also a function: **MonitoringFunction** that informs the user about the current state of the washing machine.
- The state of the washing machine: **WashingMachineStatus** can take the values "WASHING" or "STOPPED" or "ERROR".
- This state WashingMachineStatus is updated as an the OutputDataPoint of a service **MonitorService** of the washing machine that monitors the washing machine's behaviour.
- The washing machine is located at **My_Bathroom**

NOTE: "InputDataPoint", "OutputDataPoint" and "Operation" are not specified in SAREF, they are classes of the oneM2M Base Ontology.

This example will identify the specific washing machine by the URI: "WASH_XYZ_123". WASH_XYZ_123 that is an instance of class: XYZ_Cool which is contained in XYZ's ontology: http://www.XYZ.com/WashingMachines.

The ontology "http://www.XYZ.com/WashingMachines" that contains the model type "XYZ_Cool" is compliant to SAREF, which is turn is compliant to the oneM2M Base Ontology (see clause B.1.2).

Figure B.2 shows some sub-classing relationships between XYZ_Cool, SAREF and the oneM2M Base Ontology.

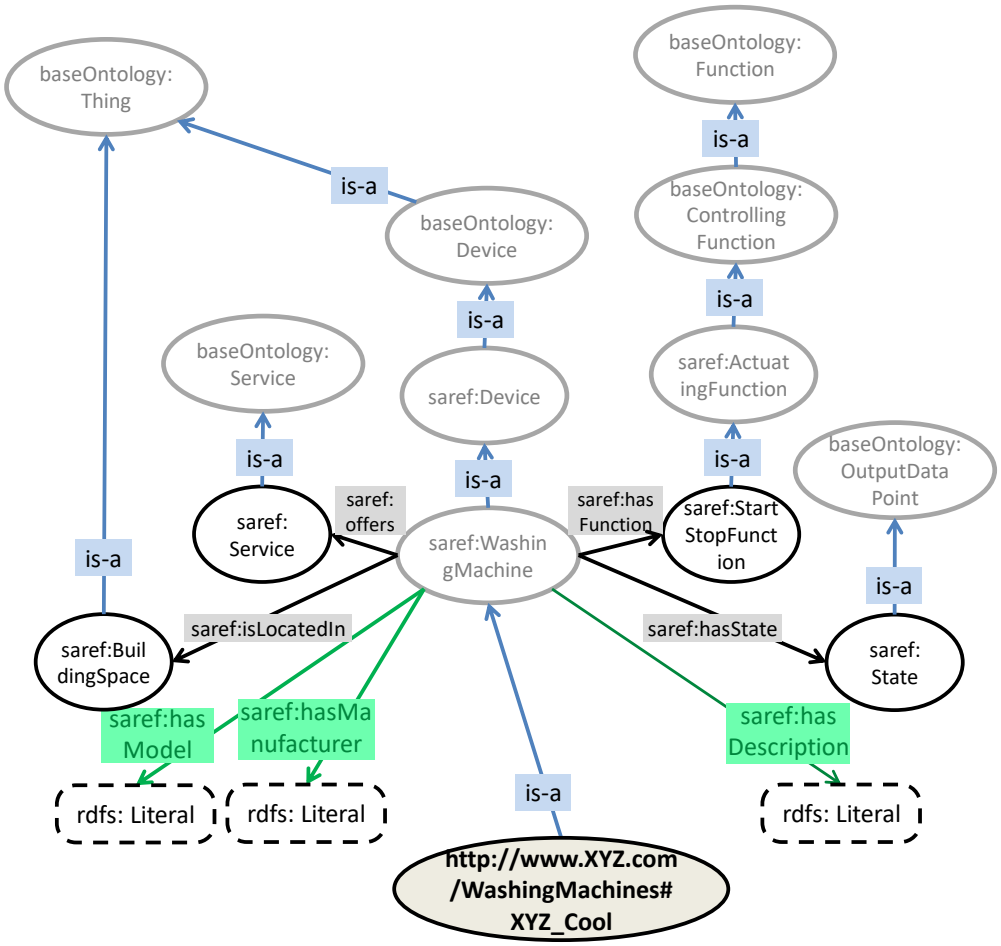

Figure B.2: Sub-classing relationships between XYZ_Cool, SAREF and the oneM2M Base Ontology

According to clause 7.1.1.2 the washing machine - as a sub-class of the oneM2M:Device class - needs to be instantiated in the data of the *descriptor* attribute of a resource of type <*semanticDescriptor*> that is a child resource of an <*AE*>.

EXAMPLE: That <AE> resource could have resourceID = "00000001" and have a resourceName "My-WashingMachine".
Its CSE-relative address would be:
Non-Hierarchical:

- "00000001"

Hierarchical:

- "./My-WashingMachine"

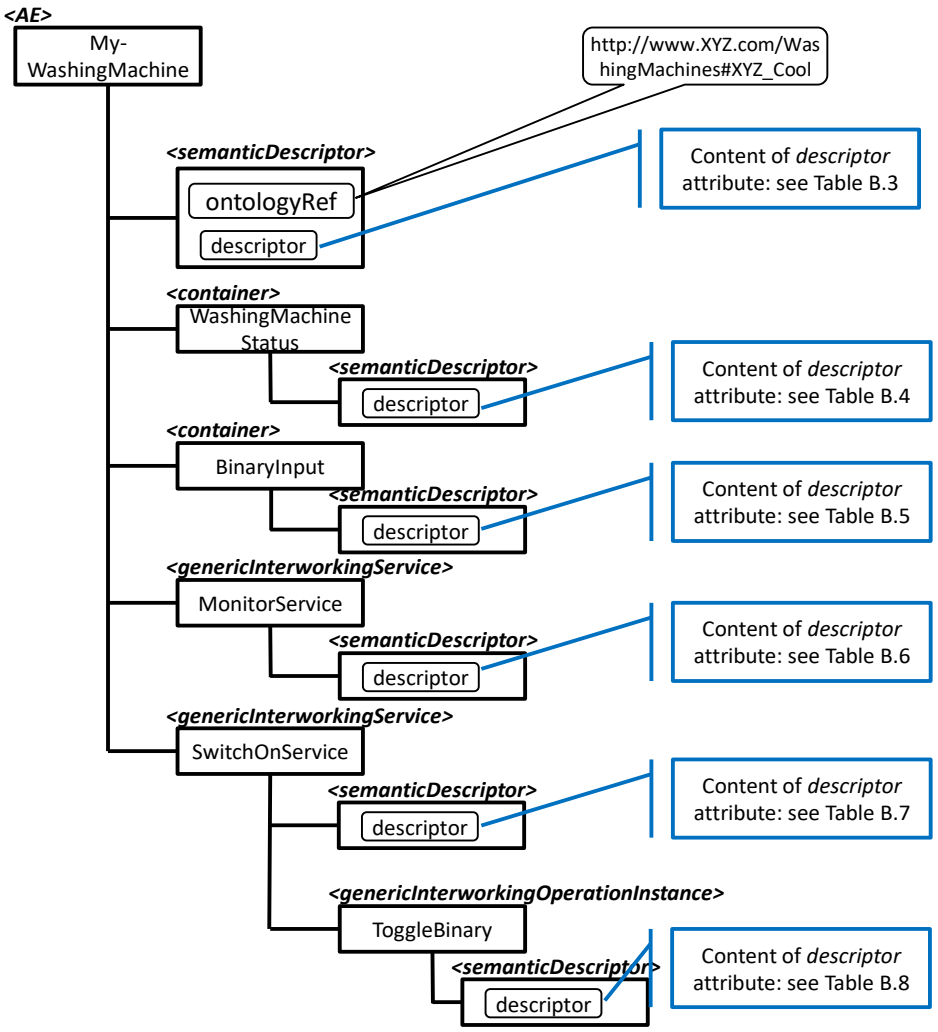

Figure B.3: Resource structure of smart washing machine <AE> and its child-resources

Figure B.3 shows the resource structure of a <AE> resource representing the smart washing machine. It consists of an ontologyRef attribute, which contains the URI of the ontology concept of the smart washing machine, e.g. "http://www.XYZ.com/WashingMachines#XYZ_Cool" (not shown in the figure: the ontologyRef attributes in the semanticDescriptors of the child resources, e.g. http://www.XYZ.com/WashingMachines#WashingMachineStatus). The ontology http://www.XYZ.com/WashingMachines needs to include - and creates sub-classing relationships with - SAREF and the oneM2M Base Ontology.

The <*AE*> resource representing the smart washing machine contains as child-resources:

- a <*semanticDescriptor*> resource that contains the rdf description of the washing machine WASH_XYZ_123 in its descriptor attribute;
- two <*genericInterworkingService*> child-resources and their <semanticDescriptor>s are used for modelling the services SwitchOnService and MonitorService;
- the SwitchOnService in turn has a child resource of type <*genericInterworkingOperationInstance*> which is created whenever a ToggleBinary Operation is invoked;
- two <*container*> child-resources and their <semanticDescriptor>s are used for holding the values for InputDataPoint BinaryInput and OutputDataPoint WashingMachineStatus for their respective services.

The RDF in the following tables shows the semantic annotation stored in the semanticDescriptor resources related to the washing machine.

Table B.5: RDF annotation contained in the descriptor attribute of the <*semanticDescriptor*> resource of a SAREF washing machine <*AE*> *resource*

```
<rdf:RDF
    <rdf:Description rdf:about="WASH_XYZ_123">
        <rdf:type rdf:resource="http://www.XYZ.com/WashingMachines#XYZ_Cool"/>
        <saref:hasManufacturer>XYZ</saref:hasManufacturer>
        <saref:hasDescription>Very cool Washing Machine</saref:hasDescription>
        <saref:hasState rdf:resource="WASH_XYZ_123*WashingMachineStatus"/>
        <saref:hasFunction rdf:resource="WASH_XYZ_123*WashingFunction"/>
        <saref:hasService rdf:resource="WASH_XYZ_123*SwitchOnService"/>
        <saref:hasService rdf:resource="WASH_XYZ_123*MonitorService"/>
        <saref:isLocatedIn rdf:resource="My_Bathroom"/>
    </rdf:Description>

    <rdf:Description rdf:about="WASH_XYZ_123*WashingMachineStatus">
        <oneM2M:resourceDescriptorLink rdf:resource=
        "./My-WashingMachine/WashingMachineStatus/semanticDescriptor"/>
    </rdf:Description>

    <rdf:Description rdf:about="WASH_XYZ_123*WashingFunction">
        <rdf:type rdf:resource="https://w3id.org/saref#ActuatingFunction"/>
        <saref:hasCommand rdf:resource="WASH_XYZ_123*WashingFunction*ON_Command"/>
        <saref:hasCommand rdf:resource="WASH_XYZ_123*WashingFunction*OFF_Command"/>
        <saref:hasCommand rdf:resource="WASH_XYZ_123*WashingFunction*Toggle_Command"/>
    </rdf:Description>

    <rdf:Description rdf:about="WASH_XYZ_123*WashingFunction*ON_Command">
        <rdf:type rdf:resource="https://w3id.org/saref#OnCommand"/>
    </rdf:Description>

    <rdf:Description rdf:about="WASH_XYZ_123*WashingFunction*OFF_Command">
        <rdf:type rdf:resource="https://w3id.org/saref#OffCommand"/>
    </rdf:Description>

    <rdf:Description rdf:about="WASH_XYZ_123*WashingFunction*Toggle_Command">
        <rdf:type rdf:resource="https://w3id.org/saref#ToggleCommand"/>
    </rdf:Description>

    <rdf:Description rdf:about="WASH_XYZ_123*SwitchOnService">
        <oneM2M: resourceDescriptorLink rdf:resource=
        "./My-WashingMachine/SwitchOnService/semanticDescriptor"/>
    </rdf:Description>

    <rdf:Description rdf:about="WASH_XYZ_123*MonitorService">
        <oneM2M: resourceDescriptorLink rdf:resource=
        "./My-WashingMachine/MonitorService/semanticDescriptor"/>
    </rdf:Description>

    <rdf:Description rdf:about="My_Bathroom">
        <rdf:type rdf:resource="https://w3id.org/saref#BuildingSpace"/>
        <oneM2M: resourceDescriptorLink rdf:resource=
        "//m2m.service.com/SomeIN-CSE/ResourceName_of_My_Bathroom/semanticDescriptor "/>
    </rdf:Description>

</rdf:RDF>
```

Table B.6: RDF annotation contained in the descriptor attribute of the <*semanticDescriptor*> resource of the WashingMachineStatus <*container*> resource

```
<rdf:RDF
    <rdf:Description rdf:about="WASH_XYZ_123*WashingMachineStatus">
        <rdf:type rdf:resource="https://w3id.org/saref#State"/>
        <oneM2M:oneM2MTargetURI rdf:resource=
        "./My-WashingMachine/WashingMachineStatus"/>
        <oneM2M:oneM2MAttribute>#latest</oneM2M:oneM2MAttribute>
        <oneM2M:hasDataType>xsd:string</oneM2M:hasDataType>
        <oneM2M:oneM2MMethod>RETRIEVE</oneM2M:oneM2MMethod>
    </rdf:Description>

</rdf:RDF>
```

Table B.7: RDF annotation contained in the descriptor attribute of the <*semanticDescriptor*> resource of the BinaryInput <*container*> resource

```
<rdf:RDF
    <rdf:Description rdf:about="WASH_XYZ_123*SwitchOnService*BinaryInput">
        <rdf:type rdf:resource="https://w3id.org/saref#Property"/>
        <oneM2M:exposesCommand rdf:resource="WASH_XYZ_123*WashingFunction*ON_Command"/>
        <oneM2M:exposesCommand rdf:resource="WASH_XYZ_123*WashingFunction*OFF_Command"/>
        <oneM2M:oneM2MTargetURI rdf:resource=
        "./My-WashingMachine/BinaryInput"/>
        <oneM2M:oneM2MAttribute>#latest</oneM2M:oneM2MAttribute>
        <oneM2M:hasDataType> xsd:hexBinary</oneM2M:hasDataType>
        <oneM2M:oneM2MMethod> CREATE</oneM2M:oneM2MMethod>
    </rdf:Description>

    <rdf:Description rdf:about="WASH_XYZ_123*WashingFunction*ON_Command">
        <oneM2M:resourceDescriptorLink rdf:resource=
        "./My-WashingMachine/semanticDescriptor"/>
    </rdf:Description>

    <rdf:Description rdf:about="WASH_XYZ_123*WashingFunction*OFF_Command">
        <oneM2M:resourceDescriptorLink rdf:resource=
        "./My-WashingMachine/semanticDescriptor"/>
    </rdf:Description>

</rdf:RDF>
```

Table B.8: RDF annotation contained in the descriptor attribute of the <*semanticDescriptor*> resource of the MonitorService <genericInterworkingService> resource

```
<rdf:RDF
    <rdf:Description rdf:about="WASH_XYZ_123*MonitorService">
        <rdf:type rdf:resource="https://w3id.org/saref#Service"/>
        <saref:represents rdf:resource="WASH_XYZ_123*MonitoringFunction"/>
        <oneM2M:hasOutputDataPoint rdf:resource="WASH_XYZ_123*WashingMachineStatus"/>
    </rdf:Description>

    <rdf:Description rdf:about="WASH_XYZ_123*WashingMachineStatus">
        <oneM2M:resourceDescriptorLink rdf:resource=
        "./My-WashingMachine/WashingMachineStatus/semanticDescriptor"/>
    </rdf:Description>

    <rdf:Description rdf:about="WASH_XYZ_123*MonitoringFunction">
        <oneM2M:resourceDescriptorLink rdf:resource=
        "./My-WashingMachine/semanticDescriptor"/>
    </rdf:Description>

</rdf:RDF>
```

Table B.9: RDF annotation contained in the descriptor attribute of the <*semanticDescriptor*> resource of the SwitchOnService <genericInterworkingService> resource

```
<rdf:RDF
    <rdf:Description rdf:about="WASH_XYZ_123*SwitchOnService">
        <rdf:type rdf:resource="https://w3id.org/saref#SwitchOnService"/>
        <saref:represents rdf:resource="WASH_XYZ_123*WashingFunction"/>
        <oneM2M:hasOutDataPoint rdf:resource="WASH_XYZ_123*SwitchOnService*BinaryInput"/>
        <oneM2M:hasOperation rdf:resource="WASH_XYZ_123*SwitchOnService*ToggleBinary"/>
    </rdf:Description>

    <rdf:Description rdf:about="WASH_XYZ_123*SwitchOnService*BinaryInput">
        <oneM2M:resourceDescriptorLink rdf:resource=
        "./My-WashingMachine/BinaryInput/semanticDescriptor"/>
    </rdf:Description>

    <rdf:Description rdf:about="WASH_XYZ_123*SwitchOnService*ToggleBinary">
        <oneM2M:resourceDescriptorLink rdf:resource=
        "./My-WashingMachine/SwitchOnService/ToggleBinary/semanticDescriptor"/>
```

```
    </rdf:Description>

    <rdf:Description rdf:about="WASH_XYZ_123*WashingFunction">
        <oneM2M:resourceDescriptorLink rdf:resource=
        "./My-WashingMachine/semanticDescriptor"/>
    </rdf:Description>

</rdf:RDF>
```

Table B.10: RDF annotation contained in the descriptor attribute of the <*semanticDescriptor*> resource of the ToggleBinary <genericInterworkingOperationInstance> resource

```
<rdf:RDF
    <rdf:Description rdf:about="WASH_XYZ_123*SwitchOnService*ToggleBinary">
        <rdf:type
rdf:resource="http://www.onem2m.org/ontology/Base_Ontology/base_ontology#Operation"/>
        <oneM2M:exposesCommand rdf:resource="WASH_XYZ_123*WashingFunction*Toggle_Command"/>
    </rdf:Description>

    <rdf:Description rdf:about="WASH_XYZ_123*WashingFunction*Toggle_Command">
        <oneM2M:resourceDescriptorLink rdf:resource=
        "./My-WashingMachine/semanticDescriptor"/>
    </rdf:Description>

</rdf:RDF>
```

History

Publication history		
V2.0.0	30-Aug-2016	Release 2 - Publication
V2.2.2	12-Mar-2018	Release 2A - Publication