TTC技術レポート
Technical Report

TR-M2M-0007v2.11.1

抽象化とセマンティクスの
適用性検討

Study of Abstraction and Semantics Enablements

2016年11月30日制定

一般社団法人
情報通信技術委員会

THE TELECOMMUNICATION TECHNOLOGY COMMITTEE

TR-M2M-0007v2.11.1

Study of Abstraction and Semantics Enablements

＜参考＞ [Remarks]

１．国際勧告等の関連 [Relationship with international recommendations and standards]

本技術レポートは、oneM2M で作成された Technical Report 0007 (Version 2.11.1) に準拠している。

[This Technical Report is transposed based on the Technical Report 0007 (Version 2.11.1) developed by oneM2M.]

２．作成専門委員会 [Working Group]

oneM2M 専門委員会 [oneM2M Working Group]

ONEM2M
TECHNICAL REPORT

Document Number	TR-0007-V2.11.1
Document Name:	Study of Abstraction and Semantics Enablements
Date:	2016-August-30
Abstract:	Collect and study the state-of-the-art technologies that may be leveraged by oneM2M to enable its abstraction & semantics capability. This includes a collection of terminology and use cases considered by other standardization or industrial fora working on ontologies, semantics and abstraction, as well as relevant source material proposed by Partner Types 1 for transfer to oneM2M and contributions from Partner Types 2. Evaluate the possibility of leveraging all or part of those technologies and/or solutions by oneM2M architecture and protocols to enable its abstraction & semantics capability.

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

Copyright Notification

Notice of Disclaimer & Limitation of Liability

Contents

1 Scope

The present document describes and collects the state-of-art of the existing technologies on abstraction and semantics capability, evaluates if the technologies can match the requirements defined in oneM2M, analyzes how the technologies can leverage the design of the architecture of oneM2M.

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 reference document (including any amendments) applies.

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

Not applicable.

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 reference 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] ETSI TS 102 690: "Machine-to-Machine communications (M2M); Functional architecture".

[i.2] ETSI TR 101 584: "Machine-to-Machine communications (M2M); Study on Semantic support for M2M Data".

[i.3] Open Geospatial Consortium.

NOTE: Available at http://www.opengeospatial.org/.

[i.4] Open Geospatial Consortium, Sensor Web Enablement DWG.

NOTE: Available at http://www.opengeospatial.org/projects/groups/sensorwebdwg.

[i.5] Open Geospatial Consortium, Sensor Web Enablement architecture, August 2008.

[i.6] Open Geospatial Consortium, Observations and Measurements.

NOTE: Available at http://www.opengeospatial.org/standards/om.

[i.7] Open Geospatial Consortium, OpenGIS Sensor Model Language (SensorML).

NOTE: Available at http://www.opengeospatial.org/standards/sensorml.

[i.8] Open Geospatial Consortium, OpenGIS Transducer Markup Language (TML) Encoding Specification.

NOTE: Available at http://www.opengeospatial.org/standards/tml.

[i.9] Open Geospatial Consortium, Sensor Observation Service.

NOTE: Available at http://www.opengeospatial.org/standards/sos.

[i.10] Open Geospatial Consortium, Sensor Planning Service (SPS).

NOTE: Available at http://www.opengeospatial.org/standards/sps.

[i.11] Open Geospatial Consortium, Sensor Alert Service.

NOTE: Available at http://www.opengeospatial.org/standards/requests/44.

[i.12] W3C Semantic Sensor Network Incubator Group.

NOTE: Available at http://www.w3.org/2005/Incubator/ssn/wiki/Main_Page.

[i.13] Open Geospatial Consortium, Sensor Web Interface for IoT SWG.

NOTE: Available at http://www.opengeospatial.org/projects/groups/sweiotswg.

[i.14] Open Geospatial Consortium Best Practice, Semantic annotations in OGC standards.

NOTE: Available at http://www.opengeospatial.org/node/1790.

[i.15] Open Geospatial Consortium, Geography Markup Language.

NOTE: Available at http://www.opengeospatial.org/standards/gml.

[i.16] Semantic Sensor Network XG Final Report, W3C Incubator Group Report 28 June 2011.

NOTE: Available at http://www.w3.org/2005/Incubator/ssn/XGR-ssn/.

[i.17] Open Geospatial Consortium, Sensor Web Enablement DWG.

NOTE: Available at http://www.opengeospatial.org/projects/groups/sensorwebdwg.

[i.18] SSN Suggested Key Ontology Intro Attributes.

NOTE: Available at http://www.w3.org/2005/Incubator/ssn/wiki/SSN_Suggested_Key_Ontology_Intro_Attributes.

[i.19] W3C Semantic Sensor Network Incubator Group, Semantic Sensor Network Ontology.

NOTE Available at http://www.w3.org/2005/Incubator/ssn/ssnx/ssn.

[i.20] DUL ssn.

NOTE: Available at http://www.w3.org/2005/Incubator/ssn/wiki/DUL_ssn.

[i.21] HGI02029: "Smart Home Architecture and System Requirements".

[i.22] W3C OWL Working Group, OWL 2 Web Ontology Language Document Overview.

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

[i.23] W3C SPARQL Working Group, SPARQL 1.1 Overview.

NOTE: Available at http://www.w3.org/TR/sparql11-overview/.

[i.24] Jack Rusher, TripleStore, Semantic Web Advanced Development for Europe (SWAD-Europe), Workshop on Semantic Web Storage and Retrieval - Position Papers.

NOTE: Available at http://www.w3.org/2001/sw/Europe/events/20031113-storage/positions/rusher.html.

[i.25] oneM2M-TR-0001-UseCase: "oneM2M Use cases collection".

[i.26] Resource Description Framework (RDF): "Concepts and Abstract Syntax".

NOTE: Available at http://www.w3.org/TR/rdf-concepts/.

[i.27] RDF Vocabulary Description Language 1.0: "RDF Schema".

NOTE: Available at http://www.w3.org/TR/rdf-schema/.

[i.28] SPARQL Query Language for RDF.

NOTE: Available at http://www.w3.org/TR/rdf-sparql-query/.

[i.29] Web Ontology Language (OWL).

NOTE: Available at http://www.w3.org/2001/sw/wiki/OWL.

[i.30] Dbpedia.

NOTE: Available at http://dbpedia.org/About.

[i.31] Turtle: Terse RDF Triple Language. Eric Prud'hommeaux and Gavin Carothers. W3C Last Call Working Draft, 10 July 2012.

NOTE: Available at http://www.w3.org/TR/2012/WD-turtle-20120710/.
Latest version available at http://www.w3.org/TR/turtle/.

[i.32] OWL Syntaxes.

NOTE: Available at http://ontogenesis.knowledgeblog.org/88?kblog-transclude=2.

[i.33] Rule Interchange Format (RIF) overview (second version).

NOTE: Available at http://www.w3.org/TR/2013/NOTE-rif-overview-20130205/.

[i.34] SWRL: A Semantic Web Rule Language Combining OWL and RuleML (Version 0.5).

NOTE: Aavailable at http://www.daml.org/2003/11/swrl/.

[i.35] SPIN - Overview and Motivation.

NOTE: Available at http://www.w3.org/Submission/spin-overview/.

[i.36] oneM2M Drafting Rules.

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

[i.37] oneM2M TS 0002: "onM2M Requirements".

[i.38] Broadband Forum TR-069: "CPE WAN Management Protocol v1.1".

[i.39] oneM2M TS 0001: "oneM2M Functional Architecture".

[i.40] D-S4 - SMART 2013-0077 - Smart Appliances - Final Study Report v1.0.

NOTE: Available at https://docs.google.com/file/d/0B2nnxMhTMGh4WTVsSVRsb01ha3c/edit.

[i.41] An exemplar Data Annotation process.

NOTE: Available at http://nemo.nic.uoregon.edu/wiki/Data_Annotation.

[i.42] IETF RFC 3987: "Internationalized Resource Identifiers (IRIs)"..

3 Definitions and abbreviations

3.1 Definitions

For the purposes of the present document, the following terms and definitions apply:

abstraction: process of mapping between a set of Device Application Information Models and an Abstract Application Information Model according to a specified set of rules

attribute: See definition in oneM2M TS 0001 [i.39].

data annotation: process of providing and inserting data into a semantic structure specified by an ontology

NOTE: Therefore, data annotation is the mapping from unstructured raw data into structured semantic data, i.e. from reality to ontology. An example of the Data Annotation process is described in [i.41].

data entity: logical entity which contains data resources and related functionalities

NOTE: It can also contain semantic resources and related functionalities, e.g. a temporary semantic graph store. The semantic functionalities are used to support the data functionalities locally.

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

physical entity: tangible element that is intrinsic to the environment, and that is not specific to a particular M2M application in this environment

NOTE 1: Depending on the environment, the physical entity may be a smart phone, a camera, a smart TV/audio, a piece of furniture, somebody, a room of a building, a car, a street of a city, etc.

NOTE 2: To be part of the M2M/IoT architecture, a physical entity does not need to be connected through a direct network interface, or even to be identified through a universal identification scheme such as RFID/EPC global, provided it can be sensed by sensors that are supposed to be deployed in this environment, and possibly acted upon by actuators.

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

EXAMPLE: "is-part-of", "is-subtype-of".

Semantic Entity: logical entity which contains semantic functionalities, e.g. it can contain a central graph store and related semantic reasoning and analytic functions, such as semantic annotation, semantic mash-up, etc.

thing: element of the environment that is individually identifiable in the M2M system

thing representation: it is the instance of the informational model of the Thing in the M2M System

NOTE: A Thing Representation provides means for applications to interact with the Thing.

3.2 Abbreviations

For the purposes of the present document, the following abbreviations apply:

AC Air Condition
AE Application Entity
API Application Program Interface
API Air Pollution Index
APP Application Program

NOTE: On the user device e.g. a smart mobile terminal.

ASN Application Servive Node

NOTE: See oneM2M TS 0001 [i.39].

BMS Building Management System
CORBA Common Object Request Broker Architecture
COSEM Companion Specification for Energy Metering
CRUD Create, read, update and delete
CSE Common Services Entity

NOTE: See oneM2M TS 0001 [i.39].

CWMP CPE WAN Management Protocol
DI Discomfort Index
DIP Device Interworking Proxy

NOTE: See ETSI TS 102 690 [i.1].

DL	Description Logic
DUL	DOLCE Ultra Lite
FFS	For Further Study
GIP	Gateway Interworking Proxy

NOTE: See ETSI TS 102 690 [i.1].

GML	Geography Markup Language
GRA	Gateway Resource Abstraction

NOTE: See ETSI TR 101 584 [i.2].

GSCL	Gateway Service Capabilities Layer

NOTE: See ETSI TS 102 690 [i.1].

HAN	Home Automation Network
HEMS	Home Energy Management System
HG	Home Gateway
HGI	Home Gateway Initiative
HVAC	Heating, Ventilation and Air Conditioning
IN	Infrastructure Node
IoT	Internet of Things
IP	Internet Protocol
IPE	Interworking Proxy Application Entity
KNX	KNX standard for building automation by the KNX Association
MN	Middle Node
MN/IN	Middle Node or Infrastructure Node
MN-CSE	Middle Node Common Services Entity
NIP	Network Interworking Proxy

NOTE: See ETSI TS 102 690 [i.1].

NSCL	Network Service Capabilities Layer

NOTE: See ETSI TS 102 690 [i.1].

OGC	Open Geospatial Consortium
OWL	Web Ontology Language
OWS	OGC Web Services
PV	Photo Voltaic
QWL	Web Ontology Language
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
RDQL	RDF Data Query Language
REST	Representational state transfer
RIF	Rule Interchange Format
RPC	Remote Procedure Call
SAREF	Smart Appliance REFerence ontology
SAS	Sensor Alert Service
SAT	Semantic Appliance Template
SCL	Services Capability Layer
SHAL	Smart Home Abstraction Layer
SOAP	Simple Object Access protocol
SOS	Sensor Observations Service
SPARQL	SPARQL Protocol and RDF Query Language
SPIN	SPARQL Inferencing Notation
SPS	Sensor Planning Service
SSN	Semantic Sensor Network
SWE	Sensor Web Enablement
SWG	Standards Working Group
SWRL	Semantic Web Rule Language
TML	Transducer Model Language

TV	Television
UML	Unified Modeling Language
URI	Uniform resource identifier
URL	Uniform resource locator
VOC	Volatile Organic Compound
W3C	World Wide Web Consortium
WNS	Web Notification Services
WoT	Web of Things
XG	Incubator Group
XML	Extensible Markup Language
XSD	XML Schema Definition

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.36].

5 Introduction on Abstraction and Semantic Capability Enablement in oneM2M

5.1 Overview

5.1.1 Motivation for Abstraction and Semantics

While M2M systems benefit from the variety of existing connectivity technologies to make any M2M Service work in almost any environment, M2M Applications developers don't expect to get into deep knowledge of each of these technologies for developing their applications. The abstraction of the technologies aims at hiding the complexity of the specific technologies by providing a single format to represent devices and unified methods directly usable by the applications.

Through Abstraction means, a M2M System decouples M2M applications from specific end device implementations - e.g. allows Home control Service to access a 'switch', whatever specific technology is used by the switch (be it KNX, ZigBee, DECT-ULE, or other technologies) because the 'switch' interface is abstracted from any specific technology.

Going further in simplifying the life of the Application developer and of the end-user, Semantics approach consists in getting information on the 'meaning' of M2M data. Semantic mechanisms enable an application to find suitable M2M data/devices and use them (if permitted), and encourage the creation of an open market for M2M data. Moreover, Semantic is essential if the M2M System is expected to interact with real world entities ("things") since a key role of Semantic is to provide a description of the relationship between things/data/information.

The semantics of specific M2M data can be provided by the industry segment that uses these data.This is the reason why oneM2M expects a lot of synergy with vertical domain industries when analysing possible solutions to provide Semantic support to M2M applications data thanks to their semantic description.

Figure 5.1.1-5.1.1-1 is an illustrated example of what is meant by both Abstraction and Semantics in the present document.

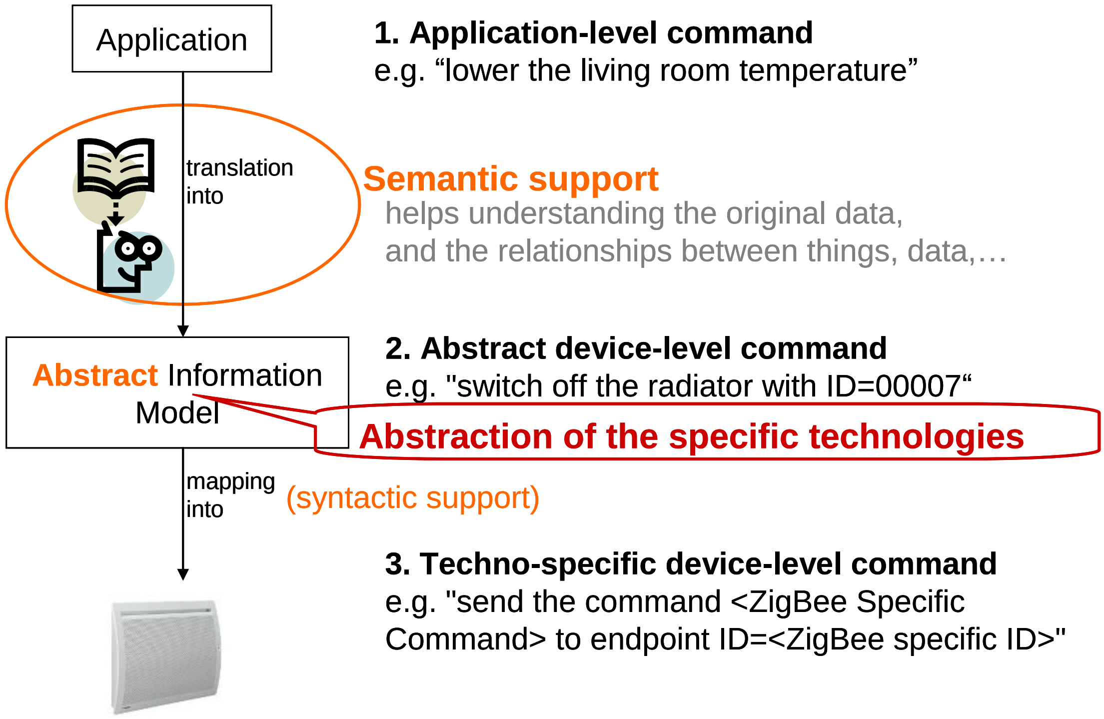


Figure 5.1.1-1: Abstraction versus Semantic for oneM2M

5.1.2 Basic Concept of Semantics

The current approach taken in oneM2M Release-1 treats data as black boxes, i.e. the content is opaque and applications have to a-priori know how to interpret the data. The result is a relatively tight coupling on the logical level (not the communication level) between the producers of data and the consumers of data. The consumer is programmed or configured for certain consumers. This typically requires a-priori agreement between the two regarding the meaning, i.e. the semantics of the data, which is then implicitly coded into the producer and the consumer.

Making the semantics explicit enables the platform to support additional functionalities like discovery, creation of mash-ups, and (big) data analysis.

However, there is not just one single level of semantics that could be attached to a raw data elment. Figure 5.1.5.1.2-1 shows different levels of meaningfulness that can be identified.

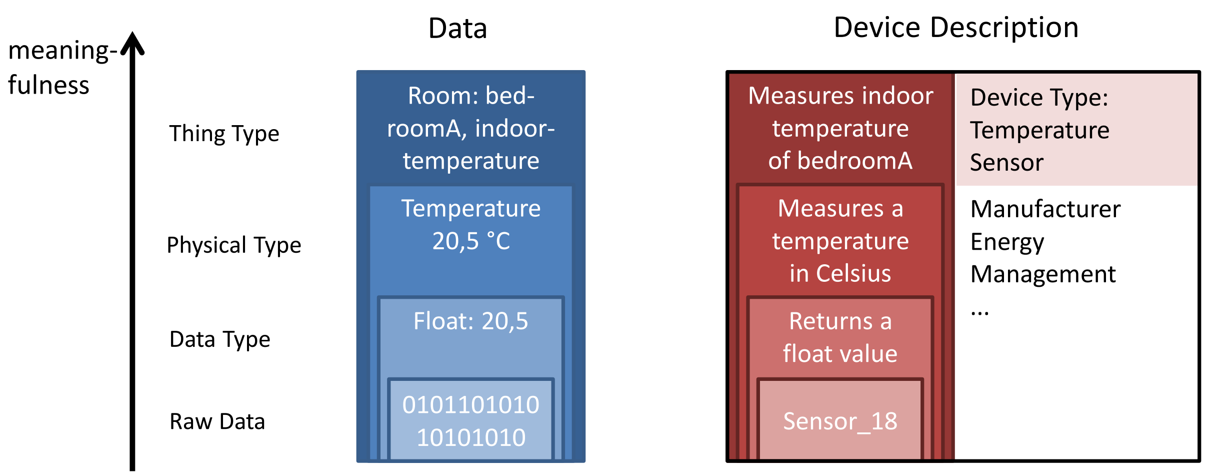


Figure 5.1.2-1: Levels of meaningfullness

The first level is the data type that defines how the raw data has to be interpreted to determine its value. In the example case, the raw data represents the float value 20,5. The physical type provides the meaning of the float value, i.e. that the float value represents a temperature given in Celsius. Finally, the thing type identifies the real-world thing and the aspect that is represented by the value. In the given case, the indoor temperature of the room with the name bedroomA.

Devices can also be semantically described. If a device produces data, the semantic description of the output directly corresponds to the semantic description of the data, as shown in figure 5.1.5.1.2-1. In addition, other aspects can be semantically described, e.g. the device type, the manufacturer, the energy consumption, management information.

5.2 Use Cases Analysis

The following other use cases from onem2M TR-0001 [i.25] have semantic aspects:

- Use Case on Devices, Virtual Devices and Things (see onem2M TR-0001 [i.25], clause 8.2).
- Semantic Home Control (see onem2M TR-0001 [i.25], clause 9.6).
- Semantic Device Plug and Play (see onem2M TR-0001 [i.25], clause 9.7).
- Vehicle Diagnostic & Maintenance Report (see onem2M TR-0001 [i.25], clause 10.1).

Annex A introduces several use cases for abstraction and semantics to identify key functionalities and potential requirements. The following table summarizes key functionalities and potential requirements for all use cases described in the annex.

Table 5.2-1: Use Cases Analysis

Clause	Name	Key functionalities	Potential requirements
A.1	Home Environment Monitoring Service using semantic mash-up	Sematic mash-up	-
A.2	Semantic Home Control (see note)	Ontology model (ThingType, DeviceType) Semantic query	-
A.3	Gym Use Case	Ontology model of treadmills Semantic query	• The M2M System shall provide the capability to publish semantic descriptions. • The M2M System shall support parsing and interpreting semantic descriptions. • The M2M System shall support resource discovery based on semantics.
A.4	Intelligent Alarm Service using Semantic Discovery and Mash-up	Semantic discovery Semantic mash-up Ontology model	• The M2M System shall provide capabilities to represent device and service information using ontology for service discovery, mash-up and data analysis.
A.5	Semantic Home Automation Control	Semantic appliance template Home automation ontology model Semantic query	• The M2M system shall be able to support semantic modelling device template for diverse M2M devices (e.g. household appliances). • The M2M System shall support common ontology to model the semantic information of M2M devices and the real-world entities (e.g. rooms) that associate with M2M devices. • The M2M System shall support semantic query to enable the discovery of target M2M devices based on their semantic information.
A.6	Semantic smart building light control	Semantic annotation Triple stores with the relationship of multiple ontologies Semantic query Reasoning	• The oneM2M systems shall support the reasoning capability for deriving implicit knowledge from semantically annotated information according to referenced ontologies.
A.7	Smart Home load control	Context-based reasoning Semantic query	• The M2M System shall provide the capability for entities of the M2M system (e.g. AEs, or CSEs) to publish semantic descriptions within the M2M system. • The M2M System shall support parsing and interpreting semantic descriptions. • The M2M System shall support resource discovery based on semantics.

Clause	Name	Key functionalities	Potential requirements
A.8	Intelligent Indoor Air Cleaning Use Case	Semantic mash-up Device control	• The M2M System shall provide the capability of mashing up virtual device resources based on existing device resources using semantic description and reasoning. • The M2M System shall provide the capability to embed semantic mash-up processing into M2M System to enable data analytics (e.g. calculation). • The M2M System shall provide the capability of interpreting and applying service logic (e.g. rules/policies of triggering operations upon other resources or attributes according to the change of the monitoring resource) described with semantic annotation and ontology.
A.9	Semantic home care for smart aging	Ontology repository	• The M2M System shall support storing ontology information which can be discovered and retrieved by M2M application or the third party system.
NOTE: This use case extends a use case of the same title in oneM2M TR-0001 [i.25], clause 9.6.			

5.3 Benefits of Abstraction and Semantics

By hiding the complexity of underlying networks, the Abstraction feature simplifies M2M for users and Applications developers. As services become independent of the various specialized technologies, it gives the opportunity to the Applications developers to focus on innovation of new services, which eventually fosters the development of the M2M market.

Semantic support for M2M, by describing the meaning of M2M data, that will also re-use existing semantics from vertical domains, is a way to enhance interoperability between initially "siloed" applications. Another key benefit from Semantic is that it enables Applications to directly interact with real-world entities, through their virtual annotated-representation.

6 Abstraction Technologies

6.1 Overview

6.1.0 Introduction

Device Abstraction is a M2M Service that allows an M2M Application to use a generic, "abstract" interface to access the functions a set of devices irrespective of the specific technology they support.

6.1.1 Basics about Interworking and Abstraction

Many systems (standardized and proprietary) that have been defined outside of oneM2M will require interworking with the M2M System. For example in the area of Home Automation ZigBee Smart Energy (SE2.0) and BACnet provide standards that describe devices and functionality to manipulate electrical machines in the home. OneM2M will need to interwork with both.

Interworking

Interworking the oneM2M System with these external systems/technologies allows am M2M Application to use devices from other technologies (e.g. ZigBee or BACnet) that are attached to the M2M System. Interworking is accomplished by Interworking Proxy functions, which could be realized as a M2M Application or as part of some CSE, that map the native interface of the device (e.g. ZigBee, BACnet, etc.) into oneM2M resources that can be accessed by M2M Applications. These resources are called a oneM2M Representation of the Information Model of the native device. Therefore, for interworked devices the M2M Application does not have to communicate with the device via its native interface but via interfaces (X and possibly Y) provided by oneM2M - the oneM2M Representation. Still, the M2M Application needs to understand the information model and the semantics of the native interface, even if the external (non-oneM2M) devices can be accessed through oneM2M mechanisms.

Abstraction

In addition to interworking the target of Abstraction is to enable an M2M Application to access the external (non-oneM2M) devices ***without the need*** to understand the information model and the semantics of the native interface. To meet that goal "abstract" devices are created in the oneM2M System. These Abstract Devices are M2M Applications or functions, located in a CSE, that translate access to its interfaces into access to interfaces of a native device. An Abstract Device can do this translation to devices from a multitude of external technologies.
Instead of communicating with the native device - or, to be more precise, with the related Interworking Proxy function - the M2M Application communicates with the Abstract Device and only needs to understand the information model of the Abstract Device.

Figure 6.1.1-1 gives an overview on Interworking and Abstraction.

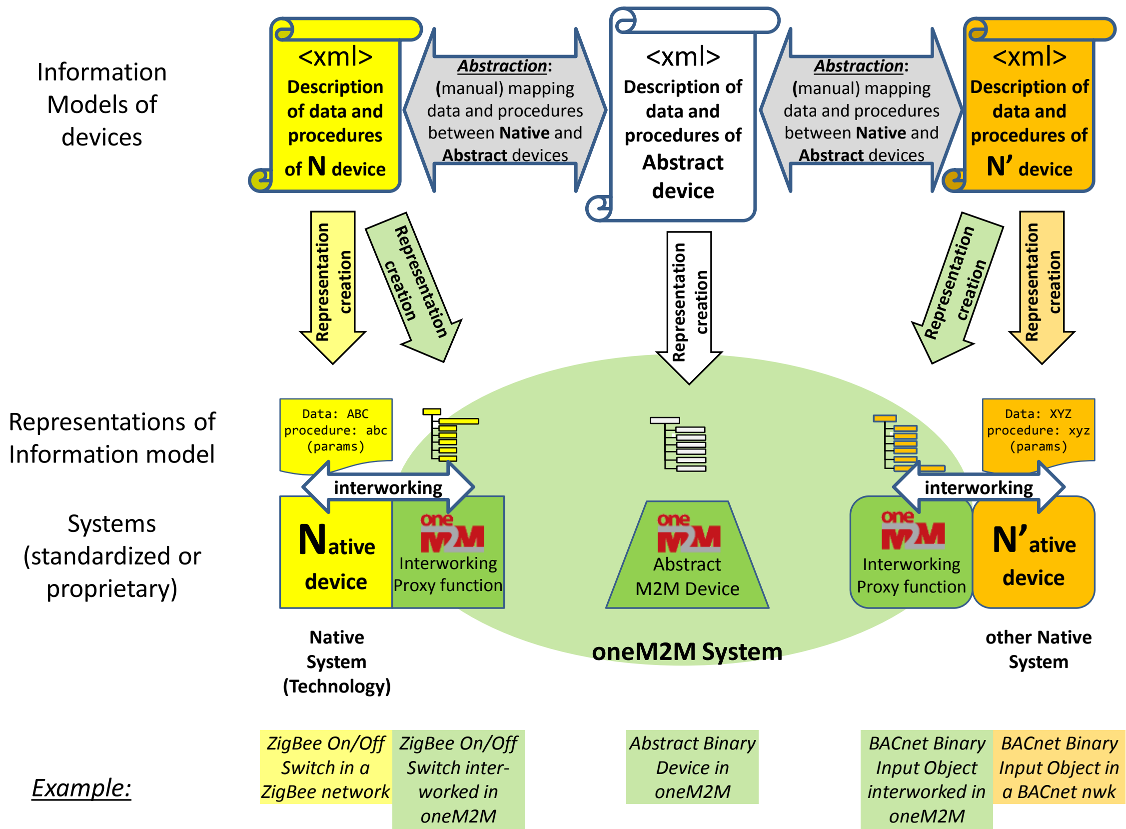


Figure 6.1.1-1: Interworking and Abstraction

6.1.2 Information models for Interworking

6.1.2.0 Introduction

In this clause the difference between Information Model and a Representation of that Information model is explained.

Additionally, it is suggested that any Information Model, that can be used for interworking purposes, needs at least to be capable to differentiate between pure data operations (Create/Read/Update/Deletw) and procedure calls. While CRUD operations are sufficient in pure RESTful systems, RPC based systems require an Information Model that allows to describe procedure calls.

Information Model

An Information Model is an "abstract, formal representation of entities that may include their properties, relationships and the operations that can be performed on them". In particular, the Information Model describes the interfaces (their datatypes and -structure) with which the entity communicates with other entities.

In general, it is not necessary for an M2M Application (i.e. an entity, that is defined outside of oneM2M) to reveal to the M2M System (i.e. to CSEs via the X reference point) the internal - application specific - structure of the data it wants to exchange with other M2M Applications. In this case the interface to other M2M Applications would consist of an opaque "Container" sub-resource. The M2M System would not know about the internal structure of this container, the container could even be encrypted.

However, in the case of interworking of external systems/devices with the M2M System such information is required by the M2M System to be able to realize the related Interworking Proxy functions.

The information on the structure of an interface to an entity (e.g. application, device, etc.) is called it's "Information Model". It describes the names of parameters, its value ranges, substructures, etc. If the interface is using procedure calls it has to contain information about whether parameters are input- or output-parameters.

Representation

NOTE: However, that the Information Model of an interface is independent of any concrete system to which it may apply (e.g. ZigBee, oneM2M, etc.). It is merely describing what data are transferred across an interface. The system specific implementation of the Information Model in a specific system is called the "Representation" of the Information Model in that system.
For example, the Information Model of a ZigBee On/Off Switch has (naturally) a Representation in a ZigBee network, but can also have a Representation in the M2M System (for the purpose of interworking with ZigBee).

Requirements on the Information Model

It should be possible to describe the Information Model of any kind of interface - whether it observes REST principles or is procedure based - in a common format. In addition the format in which Information Models are described should be machine readable to enable automatic creation of a Representations of the Information Model in the target system.

A natural choice for such a format would be XML. More specifically, since an Information Model contains only the structure and not the actual values of an interface it can be described as an XML Schema, as an XSD file (see http://www.w3.org/XML/Schema).

Examples for existing XML Schemas are:

- XSD for BBF's TR-069 [i.38] CWMP data model for device management: http://www.broadband-forum.org/cwmp/cwmp-datamodel-1-1.xsd.
- etc.

A shortcoming of existing published XML Schemas is that they usually describe only data and do not contain a description of procedures. The difference in describing procedures as compared with other data structures is that:

- Input- and output-parameters need to be clearly separated from each other.
- While the input parameters of a procedure can be individually set, one at a time, this setting of parameters does not yet imply that the procedure is executed. Only "invoking" the procedure executes it.

- A procedure may take some time to execute, so after invocation the output, relating to that invocation, may not be available until the procedure has finished.
- After invocation, a procedure may have different states (e.g. "invoked", "started", "paused", "finished", etc.) that may be relevant (e.g. to interrupt procedure execution).

For that reason it is important that an Information Model clearly distinguishes between data and procedures.

The xsd for a procedure could roughly look like:

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="procedure" type="procedureType"/>
 <xsd:complexType name="procedureType">
  <xsd:sequence>
   <xsd:element name="procedureName" type="xsd:string"
      minOccurs="1" maxOccurs="1"/>
     <xsd:element name="procCallParams" type="procCallParamsType"
      minOccurs="0" maxOccurs="1"/>
     <xsd:element name="methodResponse" type="methodResponseType"
      minOccurs="0" maxOccurs="1"/>
  </xsd:sequence>
 </xsd:complexType>

<xsd:element name="procCallParams" type="procCallParamsType"/>
 <xsd:complexType name="procCallParamsType">
  <xsd:sequence>
   <xsd:element name="params" type="paramsType"
      minOccurs="0"/>
  </xsd:sequence>
 </xsd:complexType>

 <xsd:complexType name="paramsType">
  <xsd:sequence>
   <xsd:element name="param" type="paramType"
      minOccurs="1"/>
  </xsd:sequence>
 </xsd:complexType>

 <xsd:complexType name="paramType">
  <xsd:sequence>
   <xsd:element name="value" type="valueType"
      minOccurs="1" maxOccurs="1"/>
  </xsd:sequence>
 </xsd:complexType>

 <xsd:complexType name="valueType">
  <!--
     Here the data types from other XSDs (from existing data models) can be included
   -->
 </xsd:complexType>

<xsd:element name="methodResponse" type="methodResponseType"/>
 <xsd:complexType name="methodResponseType">
  <xsd:sequence>
   <xsd:element name="params" type="paramsType"
      minOccurs="0"/>
  </xsd:sequence>
 </xsd:complexType>

</xsd:schema>
```

In the past, several attempts have been made also to describe procedures in XML, e.g. XML-RPC (http://en.wikipedia.org/wiki/XML-RPC), which later evolved into SOAP/WSDL. However these later evolutions were targeting the use of XML for indeed implementing RPCs (i.e. to be used at runtime), which is not needed for the description of the structure of a procedure.

6.1.2.1 Mapping Information Models into oneM2M Resources

For interworking purposes for a given Information Model oneM2M Resources need to be created according to the structure given by the Information model.

In particular, when the Information Model is specified by an XSD file the following rules should apply:

- Attribute names shall be the same ones as given in the element name of the XSD description.
- Simple atomic attribute types (like "Boolean", "integer", "string", etc.) shall be indicated in the Description of the resource.
- Attributes of type "sequence" shall be mapped into a Group Resource with the same name as the element name of the XSD description.
- Resources that represent Procedures contain:
  - A sub-Resource which contains 0..1 sub-Resources "procCallParams".
  - A sub-Resource which is a collection of execution Instances:
    - Each execution Instance contains 0..1 sub-Resource "methodResponse".
  - A special attribute "execEnable". An UPDATE on that attribute will trigger execution of the procedure with the current "procCallParams" parameters.

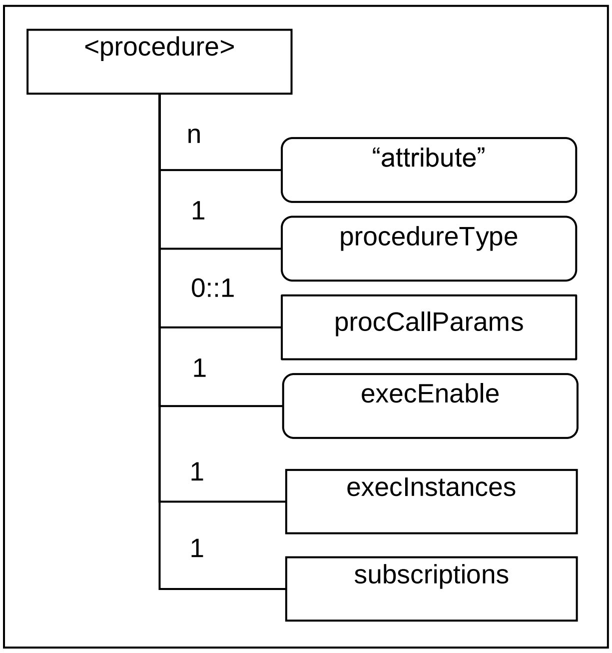


Figure 6.1.2.1-1

6.1.2.2 Proposal

It is proposed that oneM2M should encourage external bodies, who wish to interwork with oneM2M, that similar approach should be taken and Information Models should be published as XSD files.

A convention on how to express the xsd description of procedures - similar to the one given above - needs to be established.

6.1.3 Abstraction and Semantics

6.1.3.0 Introduction

A common format for describing Information Models, as described above, facilitates creation of system specific interface Representation (e.g. as oneM2M Resources) for external systems/entities. This allows easy creation of Interworking Proxy functions.

However, from the oneM2M Resource representation it cannot be deduced that a particular resource has been built according to a published Information Model. E.g. although all the parameters in a resource are called exactly the same way as described in BBF's TR-069 [i.38] CWMP data model this still may have happened by coincidence.

If it is desired to indicate that Resources have been built according to a specified Information Model, these Resources should contain a special attribute that contains a link to the XSD of the Information Model. That way it is always possible to ensure compliance of the Resource structure with that Information Model.

In addition, this link would allow to search for entities that comply with that Information Model (e.g. devices of a particular BBF TR-069 [i.38] Device Type).

Semantics

The Information Model (the XSD file as described above) only describes the structure of the interface to entities (oneM2M or external). However it does not describe any semantics (i.e. the meaning of data/behaviour of entity types). To enable the described abstraction, the semantic relation between each concrete instance (e.g. the ZigBee Information Model) and the common super type (the Abstraction Information Model) has to be modelled.

For example, within a home environment one light switch might implement the ZigBee protocol, another one the BACnet protocol. The first one would have a type "ZigBee On/Off Switch", the type of the second one would be "BACnet Binary Input Object". Only knowing their types and Information Model would not allow the conclusion that both are light switches.

This additional (semantic) information could be added by adding references to an ontology, that defines "ZigBee On/Off Switch" and "BACnet Binary Input Object" as sub-classes of "binary switch".

Depending on how the Information Model of an Abstract Device for a binary switch would in the future look like, another sub-class "ABSTRACT Binary Input Object" could be created in that ontology and would be a sub-class (or maybe an equivalent class) to "binary switch".

Such semantic information, which basically would consist of vocabulary of class-type names and their relationship (e.g. " is sub-class of") can be formally described in an ontology. The most common languages for describing ontologies are RDF(S) and OWL. RDF (Resource Description Framework) allows making statements as triples consisting of subject, predicate and object. RDFS (Resource Description Framework Schema) provides a vocabulary for structuring RDF Resources. This includes the modelling of classes (rdfs:Class), the rdf:type property that links instances to a class and the rdfs:subClassOf property, which allows the specification of class hierarchies. SPARQL is a query language for RDF triples that takes into account the subclass relations. OWL (Web Ontology Language) goes a step further enabling ontology reasoning. OWL offers different sublanguages with different levels of expressiveness and related properties regarding reasoning completeness and time complexity. Ontology reasoning can be used to deduce subclass relations, to determine whether something is an instance of a class and to check consistency.

While the capability for semantic search (e.g. give me all instances of classes that are sub-classes of "binary switch") is clearly of importance for oneM2M as it will allow satisfying identified search requirements, further aspects related to the need of supporting ontology reasoning require some further study.

Abstraction

Also, ontologies, depending on the granularity of their entries, may contain information on the mapping between Abstract Device Information Models and 'real' Device Information Models. E.g. it could be stated that the "On" state of the ZigBee On/Off Switch corresponds to the "TRUE" state of the "ABSTRACT Binary Input Object".

6.1.3.1 Proposal

For the purpose of abstraction, it is not absolutely necessary to have semantic information available in the oneM2M System. Nevertheless, it should already now be foreseen in the design principles to add semantic aspects to the resources in the oneM2M System.

We propose to add semantic information by linking resources to ontology concepts, e.g. as specified in RDF or OWL ontology files.

6.2 Introduction of Existing Technologies

6.2.1 Introduction to ETSI M2M Device Abstraction

6.2.1.1 Architecture

Native devices (type d) can host several applications. For example, a ZigBee device can have several on/off switches. Each switch is a distinct application and needs to be registered to the Gateway as well as the Network. As specified in the ETSI TS 102 690 [i.1], clause 6.1, the GIP capability provides interworking between non ETSI compliant devices and the GSCL.

Figure 6.2.1.1-1 from ETSI TR 101 584 [i.2] shows a high-level architecture for supporting device abstraction. Native devices (e.g. ZigBee devices) are first registered in the GSCL as native applications through the GIP capability. These native applications are then abstracted in corresponding abstract resources through a capability supporting device abstraction, which is called the Gateway Resource Abstraction (GRA) capability. Both native and abstracted applications are then registered (or announced) to the NSCL via mId interface. Both GSCL and NSCL have abstract resources in their resource tree.

This architecture provides both legacy M2M applications, which have access network specific knowledge, and standard M2M applications to have an access to native resources. The legacy M2M applications can access through the native applications while the standard M2M applications do through the abstracted resources.

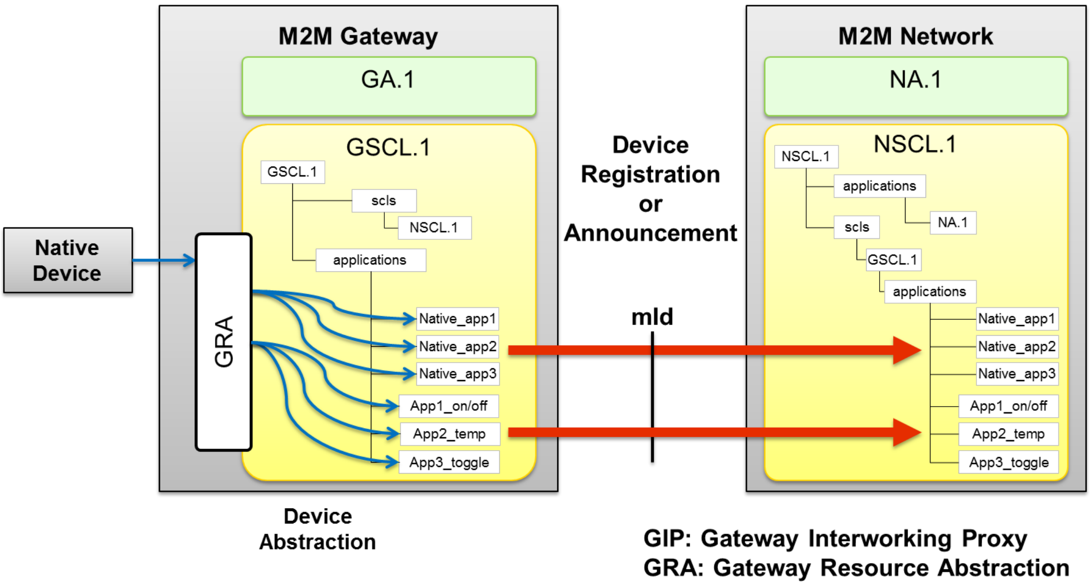


Figure 6.2.1.1-1: High-level architecture for supporting device abstraction

6.2.1.2 Interworking with legacy devices (d) through abstract devices

Figure 3 provides a resource-entity model that represents an M2M area network. In this model, each device in the network has native data and methods which are provided via access network-specific interfaces to applications. In order to provide interworking with M2M network applications that do not understand access specific technologies, the model defines an abstract application and linked it to its native application.

Since not all native applications are directly mapped to an abstracted application, the model provides 1 (native application) to 0..n (abstract application) relationship. All child entities of both native and abstract application such as interface, data field and method have the same 1 to 0..n relationship.

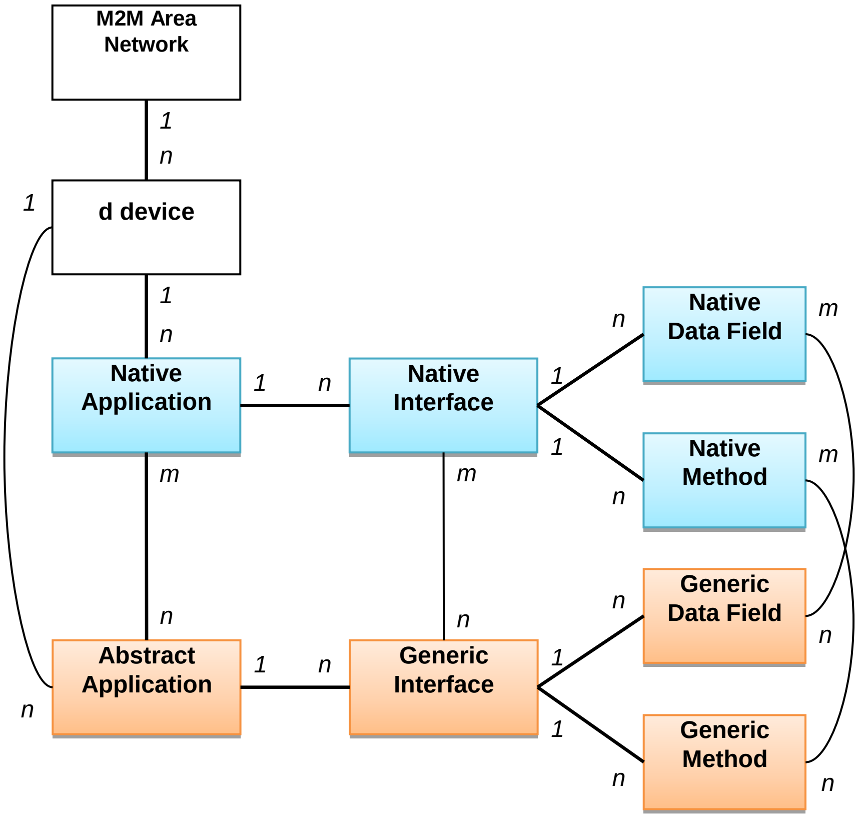


Figure 3.2.1.2-1: Generic entity-relation diagram for an M2M Area Network and its resources

This entity-relation diagram is applicable to the following M2M Area Networks:

- ZigBee.
- DLMS/COSEM.
- Zwave.
- BACnet.
- ANSI C12.
- mBus.

Native resource

Native resource is an Application resource specified in the ETSI TS 102 690 [i.1] that shall store network specific information about the Application. Same as application resource, native resource is created as a result of successful registration of an Application with the local SCL. M2M network applications that understand network specific information can interwork with legacy devices (d) through this native resource.

Abstract resource

An abstracted resource shall point to the native resource hosted in another SCL or in the same SCL. The abstracted resource is a virtual resource which consists of a set of generalized attributes instead of local area network specific attributes, such as, the *searchStrings*, the *abstractLink* to the original resource, a set of *genericCommands*, which are visible to applications (e.g. toggle, on and off), and the *accessRight*. The purpose of the abstracted resource is to represent the original resource without any network-specific information, so that the issuer does not need to know about any prior knowledge of the used underlying network technology. An abstracted resource itself shall be considered the same as other native resources that are located in the same SCL. When an abstracted resource is discovered, it returns a direct reference to the native resource.

6.2.1.3 Gateway Resource Abstraction (GRA) Capability

At start-up of forming a local area network, the GIP capability detects new devices that have joined the network and creates original M2M resources on GSCL, which are specific to the local network technology. When the GIP capability creates the original resources, the GRA capability detects new resources, creates their corresponding abstract resources and registers them in proper SCLs.

The GRA Capability in the M2M Gateway is an optional capability, i.e. deployed when needed/required by policies.

The GRA Capability provides the following functionalities:

- Detects any additions of new native resources in the GSCL.
- Generates an abstracted resource from the native resource, which is non ETSI compliant resource.
- Links native resources to their corresponding abstract resources.
- Registers abstract resources to the NSCL.
- Subscribes to native resources to be notified any updates.
- Synchronize abstracted resources to their native resources.
- Provides functional mapping between the abstracted information (i.e. generic attributes and commands) and the underlying network specific information.
- GRA may either be an internal capability of GSCL or an application communicating via reference point dIa with GSCL. GRA can also be merged with the xIP (i.e. GIP, NIP and DIP) capability, so that provides resource abstraction and interworking capabilities together.

6.2.1.4 Subscription of Abstract Resources

Any xA in the ETSI M2M architecture should be able to create a subscription to an abstract resource. The xSC is responsible for managing the subscription. Any xA that subscribes to an attribute value can be notified when the value changes.

6.2.1.5 Mapping Principle

This clause describes the mapping principles that are used to map a generic M2M abstract resource into a native M2M resource. There exist two ways of describing an abstract device. The first one is to consider each abstract device as an application. The second mapping method uses the subcontainers resource so that each abstract device is considered as a *container* resource and registered to the network application where they are belonging to.

Representing the M2M Area Network using Link:

Each abstract application belonging to a Device (N.B.: they are not ETSI M2M Applications) is modeled with an ETSI M2M <abstract-application> resource. The URI used to access this <abstract-application> resource has the following format:

```
<sclBase>/applications/<networkX_deviceY_abstract-applicationZ>
```

The <abstract-application> resource contains an ETSI M2M <container> sub resource. The URI used to access this <container> resource has the following format:

```
<sclBase>/applications/<networkX_deviceY_abstract-applicationZ>/containers/descriptor
```

The <container> resource contains one or more <contentInstance> sub resource. The "content" attribute of this sub resource contains the representation of the Application. In particular, since an Application can implement several Interfaces, each of them modeled with ETSI M2M resources (see next bullet for description), the "content" attribute of the <contentInstance> resource may contain the URIs of the ETSI M2M resources representing these Interfaces. The URI used to access the <contentInstance> resource containing the current representation of the Application has the following format:

```
<sclBase>/applications/<networkX_deviceY_abstract-applicationZ>/containers/descriptor/contentInstances/latest
```

The <contentInstance> resource pointed by the "latest" attribute of the contentInstances resource contains always the current representation of the Device.

Each Data Field and each Method belonging to an Abs_Interface is generalized from their corresponding native Data Field and method. Same as to the native one, they can be mirrored or retargeted.

If the Data Field or the Method is mirrored the ETSI M2M <abstract_application> resource modeling the Application contains an ETSI M2M <container> sub resource for each interface element mirrored (either Data Field or Method). The URI used to access this <container> resource has the following format:

```
<sclBase>/applications/<networkX_deviceY_abstract-applicationZ>/containers/<abs_interfaceW_datafieldN>
```

or

```
<sclBase>/applications/<networkX_deviceY_abstract-applicationZ>/containers/<abs_interfaceW_methodM>
```

The <container> resource contains one or more <contentInstance> sub resource. The "content" attribute of this sub resource contains the representation of the Data Field or the Method; for the Data Field it is its value, for the Method it is the actual parameters used for a Method invocation or the result of a Method invocation. The URI used to access the <contentInstance> resource containing the current representation of the Data Field or the Method has the following format:

```
<sclBase>/applications/<networkX_deviceY_abstract-applicationZ>/containers/<abs_interfaceW_datafieldN>/contentInstances/latest
```

or

```
<sclBase>/applications/<networkX_deviceY_abstract-applicationZ>/containers/<abs_interfaceW_methodM>/contentInstances/latest
```

The ETSI M2M <abstract_application> also has a link to its native <application>. The URI used to access the <native_application> resource containing the native representation of the resource has the following format:

```
<sclBase>/applications/<networkX_deviceY_native-applicationZ>
```

Figure 6.2.1.5-1 provides an overview of the resources used to model an example of an abstract device.

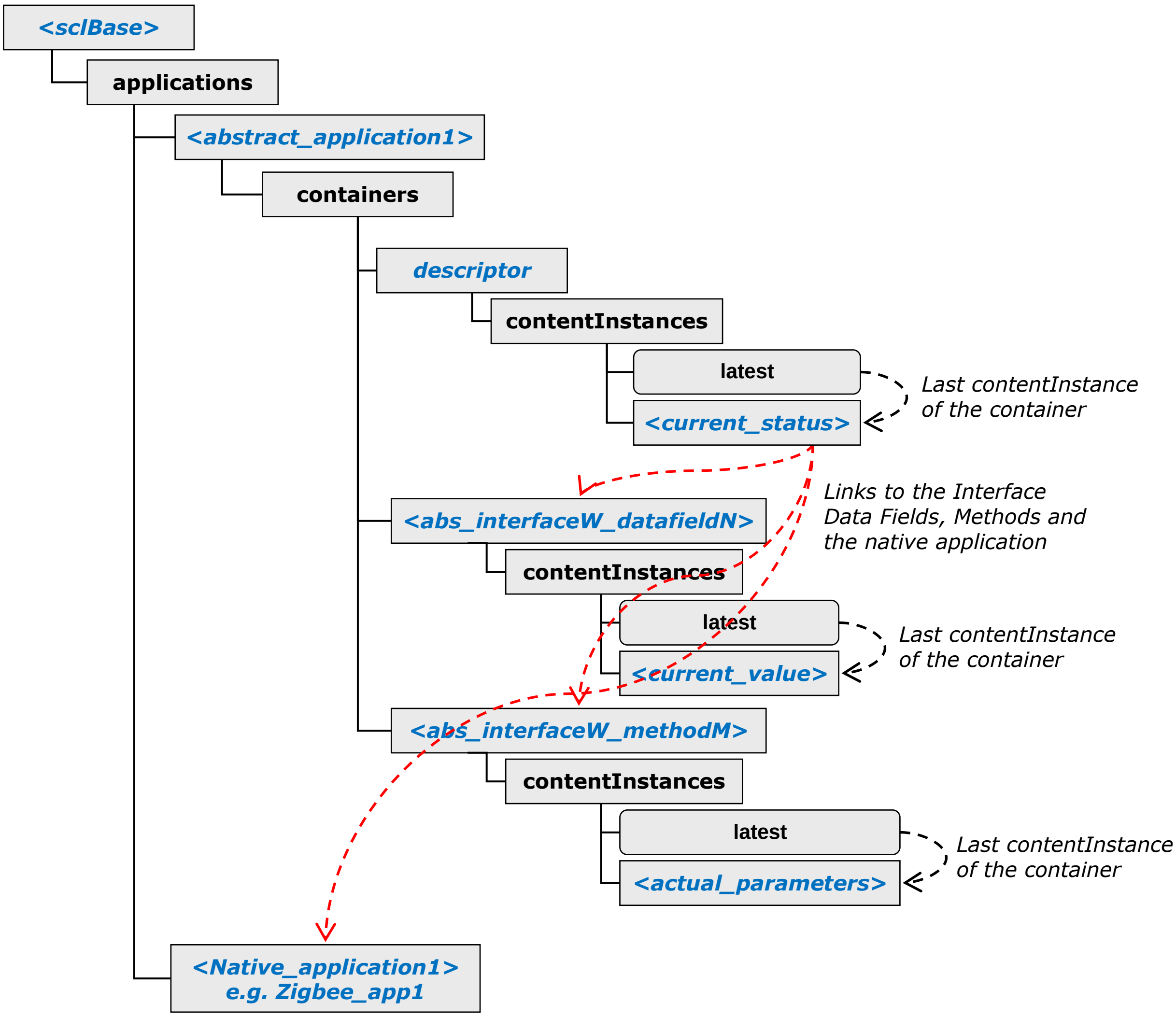


Figure 6.2.1.5-1: Linking an abstract resource to its native resource based on the ETSI M2M resource architecture

Representing abstract device using subcontainers:

In this representation method, the "*subcontainers*" resource can be used instead of the link. The *subcontainers* resource is a resource that is used to represent a collection of sub-container <*container*> resources. Since the *subcontainers* resource links a <*container*> resource with sub-container <*container*> resources, e.g. *../containers/<parentcontainer>/subcontainers/<container>*, all abstract devices and original devices are represented as a container.

For example, in the representation using subcontainers, each device regardless of type (i.e. abstract or original) is described as a container and included in the *subcontainers* of the M2M Area Network application resource. The URI used to access the <container> resource of an abstract device Y has the following format:

```
<sclBase>/applications/<networkX >/subcontainers/<networkX_deviceY_abstract_container>
```

while the <container> resource of an original device Y has the following format:

```
<sclBase>/applications/<networkX >/subcontainers/<networkX_deviceY_container>
```

The <subcontainers> resource contains one or more containers for devices. Figure 6.2.1.5-2 provides an overview of the resources used to model an example of an abstract device using the *subcontainers* resource.

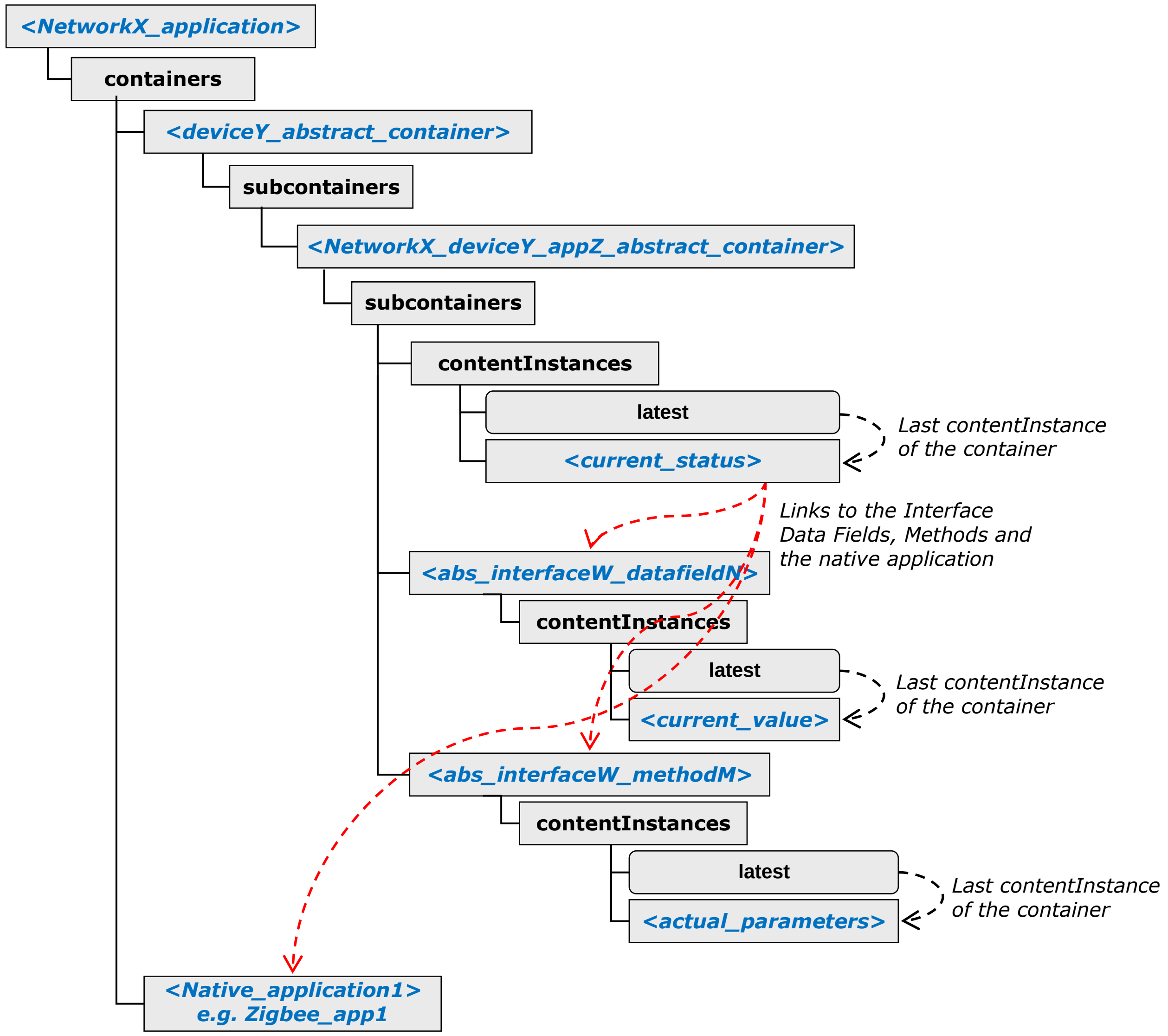


Figure 6.2.1.5-2: Mapping of an abstract device to the ETSI M2M resource architecture using the *subcontainers* resource

6.2.2 Introduction to Home Gateway Device Abstraction Concept

6.2.2.1 Architecture

Smart Home Abstraction Layer (SHAL) maps appliances to a common representation independent of the home automation technology. SHAL translates protocol-independent requests from applications to protocol-specific ones and then forwards them to the appropriate driver. SHAL represents an Abstract Application Interface for appliances - a technology agnostic description of appliances. Figure 6.2.2.1-1shows a high-level conceptual architecture for Home Gateway device abstraction technology [i.21].

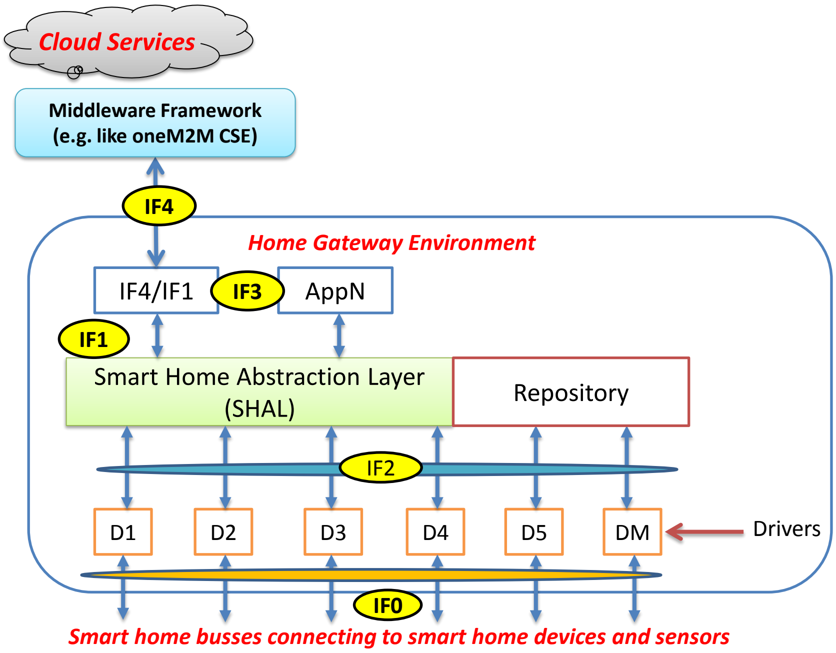


Figure 6.2.2.1-1: A high-level conceptual HGI architecture

IF1 Abstraction Application Interface: provides a common representation of appliances in the Home Domain to the Execution Environment, so that HG Applications can be independent of the different home automation technologies. For example a ZigBee lamp and a ZWave lamp are represented in the same way through an Abstraction Layer Service, so that an application can switch both off without dealing with Zigbee/ZWave specifics.

IF2 Device Application Interface: In some cases (mainly for management purposes) it is useful to have direct access to the home automation protocol, in order to do for example protocol-specific configuration or troubleshooting.

IF3: provides "higher-level" service application interface.

IF4 Remote Representation: defines the representation of the abstract application interfaces for the backend over a remote protocol. The data corresponds to the data available through IF1, 2 and 3. This reference point also maps remote protocol events to a suitable local notification service for the HG apps.

IF0 External Reference Point: defines the external reference point with bindings to selected protocols.

SHAL Middleware translates ALL of protocol and data models into IF1 primitives what is needed by local networked Appliances. The Home Gateway supports the Cloud protocol(s) over IF4, providing required handshaking and (proxied) status information from Appliances. There is a need to extract the data and commands from the Cloud protocol and translate ALL aspects to primitives on IF1 software interface. Each driver for a local network technology has to properly translate IF1 primitives into the (proprietary or standardized) signalling on the local network.

Main goals of SHAL are as follows:

1) To provide unified APIs for application developers to command, control and query home appliances.
2) Independence of underlying HAN technologies so that an application developer doesn't need to know anything about Zigbee, Z-Wave, wireless m-bus, etc.
3) To enable applications to be portable across different HGI compliant devices.
4) To enable extending the system with additional HAN technology support without service interruption.
5) Application should be able to use a pass-through mechanism to use technology-specific functions.

The abstract appliance interface descriptions should be mappable to various environments such as Java and/or OSGi, other execution environments (i. e. iOS, Android), REST APIs and other remote protocols (SOAP, CORBA, etc.).

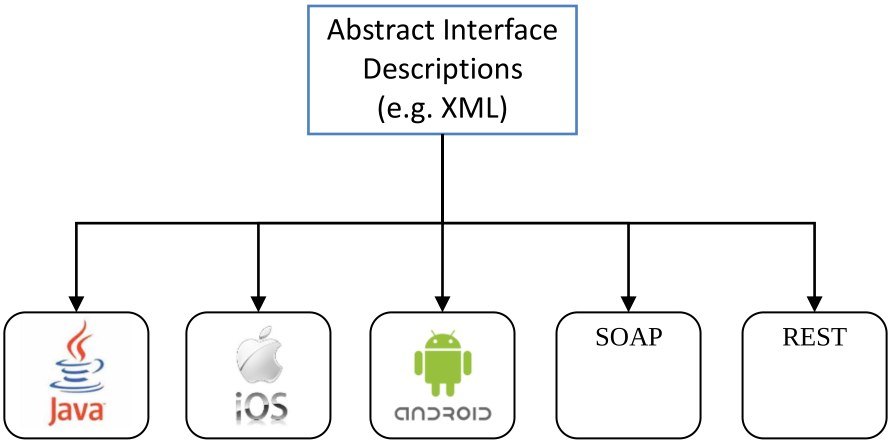


Figure 6.2.2.1-2: Abstract Interface Descriptions

6.3 Specific requirements for abstraction

Table 6.3-1: requirements for abstraction from oneM2M TS 0002 [i.37]

Requirement ID	Description	Release
ABR-001	The M2M system shall provide a generic structure for data representation.	
ABR-002	The M2M system shall be able to provide translation mechanisms among Information Models (including meta-data) used by M2M Applications, M2M Devices/Gateways, and other devices.	
ABR-003	The M2M System shall provide capabilities to represent Virtual Devices and Things, (which are not necessarily physical devices.)	

7 Technologies for Semantic M2M System

7.1 Overview

7.1.1 Introduction to Semantics technologies

Semantics can provide machine interpretable descriptions using meta-data and annotations. These descriptions contain various information for data, users, devices, applications, environments. Semantics also can help describing different attributes of M2M devices and data.

Basically, semantic technology aims to provide semantic interoperability for accessing resources/services and interpreting data from different stakeholders.

The following key technologies for supporting semantics have been identified.

- **Semantic annotation** for providing semantic information of various entities (e.g. data, user, application, etc.) that complement M2M data of these entities.
- **Use of Ontologies** for modelling semantics of physical, virtual and abstract entities. Ontologies need to have machine-readable representations, using standard languages, for use in oneM2M.
- **Semantic processing:**

  1) **Semantic discovery**, enhancing the M2M discovery mechanism, to allow locating and linking resources or services based on their semantic information.

  2) **Semantic reasoning** to derive new relations and classifications of semantically annotated data.

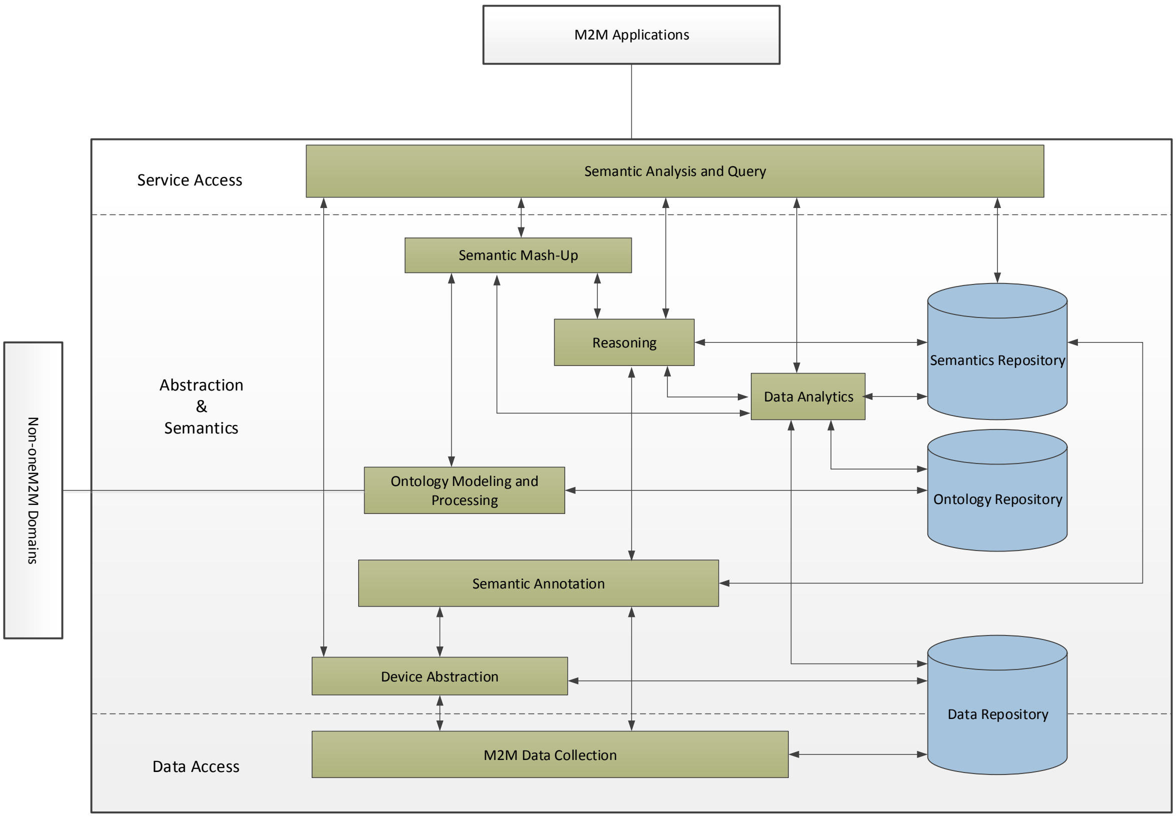
3) **Semantic mash-up** for creating a new virtual devices and offering new M2M services.

7.1.2 Key functionalities for Semantics

7.1.2.0 Introduction

Figure 7.1.2.0-1 shows a generic functional model to support semantics for various M2M applications. The functionalities of figure 7.1.2.0-1 are logically composed of three main parts:

- **Service access** which provides an interface with various M2M applications;
- **Abstraction & semantics** which perform main functionalities for semantics to M2M data and resources;
- **Data access** which provides connections with a device and/or a gateway for accessing M2M data.



NOTE 1: In this figure, "Data Analytics" is targeting to semantics. Therefore, "Data Analytics" provides the capability of interpreting and applying service logic based on semantic annotation and ontology as well as computation from data from semantics requirements and use cases.

NOTE 2: The entities and relationships depicted in the above figure are useful for analytic purposes of this present document but is not meant to be used for the purpose of normative standardization.

Figure 7.1.2.0-1: Generic functional model for supporting semantics

7.1.2.1 Semantic Analysis and Query

In semantic analysis and query, the requests from an M2M application are analyzed semantically. Based on the analysis, it creates semantic query messages and sends the messages to functional components (e.g. ontology repository, reasoning, semantic mash-up, etc.) in abstraction and semantics for requesting semantic information. After obtaining the requested information, it responds to the M2M application.

7.1.2.2 Reasoning

Reasoning is a mechanism to derive a new implicit knowledge from semantically annotated data and to answer complex user query. It can be implemented as a piece of software to be able to infer logical consequences from a set of asserted facts or axioms.

7.1.2.3 Ontology Repository

Ontology repository is storage of ontologies. Ontology is a formal specification of a conceptualization that is defining concepts as objects with their properties and relationships versus other concepts. Therefore, Ontology can be defined as a linguistic artifact that defines a shared vocabulary of basic concepts for discourse about a piece of reality (subject domain) and specifies those concepts including operations.

Ontology repository provides a way for storing, retrieving and maintaining of ontology which is described as OWL or RDF. It should be able to handle large-scale data sets with a lot of concepts for various purposes (e.g. publishing, sharing, indexing, searching, reuse of ontology, etc.). It support languages for query (e.g. RDF Data Query Language (RDQL), QWL Query Language (QWL-QL), SPARQL Protocol And RDF Query Language (SPARQL), etc.).

7.1.2.4 Ontology Modelling and Processing

Ontology modeling is the process for building an ontology which is used to model a domain and support reasoning about concepts. Examples of languages for ontology modeling are XML-based RDF, RDF Schema(RDFS), OWL, etc.

Ontology Processing is the process of classifying, storing and providing discovery function of published/modeled ontologies from external and internal of the M2M domain. The ontologies are converted and stored in Ontology repository in a unified langugage (e.g. RDFS/OWL) that can be shared and used to enable semantics to resources.

7.1.2.5 Semantic Mash-up

Semantic mash-up provides functionalities to support new services through the creation of new virtual devices, which do not exist in physical world, by obtaining semantic information through semantic descriptions from existing M2M resources in the M2M System.

7.1.2.6 Semantic Annotation

Semantic annotation of M2M resources is a method for adding semantic information to M2M resources so that it provides consistent data translation and data interoperability to heterogeneous M2M applications. Semantically annotated M2M resources can be contacted by an M2M application that understands what data are provided by the resources and what these data means. These annotations provide more meaningful descriptions and expose M2M data than traditional M2M system alone. Semantic information is annotated using Resource Description Framework (RDF) or Web Ontology Language (OWL).

7.1.2.7 Semantics Repository

7.1.2.7.0 Introduction

Semantics repository stores the semantics annotations of resources. Semantics repository also stores the new implict semantics information from the result of reasoning. It supports languages for query (e.g. RDF Data Query Language (RDQL), QWL Query Language (QWL-QL), SPARQL Protocol And RDF Query Language (SPARQL), etc.).

7.1.2.7.1 SPARQL Update Language

7.1.2.7.1.0 Introduction

SPARQL 1.1 specified by W3C is used to facilitate querying and manipulating RDF graph content on the Web or in a RDF store. A SPARQL update language is specified to support two categories of update operations on a Graph Store, i.e. Graph Update and Graph Management. In the the present document, Graph Update is introduced to manage the addition or removal of triples from some graphs within the Graph Store. The introduction on Graphy Management is out of the scope of the present document.

In the following clauses a list of graph update operations offered by SPARQL are introduced.

7.1.2.7.1.1 INSERT DATA Operation

The `INSERT DATA` operation is typically used to add some new triples, given inline in the request, into a graph stored in the Graph Store. If the graph does not exist and it can not be created for any reason, then a failure is returned.

The operation can be executed by running this command: `INSERT DATA triple_name { GRAPH <g> {PAYLOAD}}`

where *triple_name* is the name of a graph into which you want to create new triples.

Many exeception cases can be handled such as:

- If the data with *triple_name* is requested to insert a graph that does not exist in the Graph Store, then the graph should be created if creation of a new graph is permitted, otherwise failure message will be responded.
- If the *triple_name* is NULL, then the default graph is presumed.

The operation can be executed using HTTP verbs by running the below command:

- Case1: a specific graph is targeted by <graph_uri>

```
PUT/rdf-graph-store?graph=_graph_uri_.. HTTP/1.1
    Host: host_name_
    Content-Type: text/turtle

    ... #RDF payload ...

    DROP SILENT GRAPH <graph_uri>;
INSERT DATA { GRAPH <graph_uri> { .. RDF payload .. } }
```

- Case2: the default graph is targeted:

```
PUT/rdf-graph-store?default HTTP/1.1
    Host: host_name_
    Content-Type: text/turtle

    ... #RDF payload ...

    DROP SILENT DEFAULT;
    INSERT DATA { .. RDF payload .. }
```

7.1.2.7.1.2 DELETE DATA Operation

The `DELETE DATA` operation is responsible for removing some triples, given inline in the request, from a graph stored in the Graph Store.

The operation can be executed by running this command: `DELETE DATA triple_name { GRAPH <g> {PAYLOAD}}`.

As with `INSERT DATA, DELETE DATA` is meant for deletion of ground triples which results in that *triple_name* that contains blank nodes is disallowed in `DELETE DATA` operations.

The operation can be executed using HTTP verbs by running the below command:

- Case1: a specific graph is targeted by <graph_uri>

```
DELETE/rdf-graph-store?graph=_graph_uri_.. HTTP/1.1
    Host: host_name_
    Content-Type: text/turtle

    ... #RDF payload ...

    DELETE DATA { GRAPH <graph_uri> { .. RDF payload .. } }
```

- Case2: the default graph is targeted

```
DELETE/rdf-graph-store?default HTTP/1.1
    Host: host_name_
```

```
    Content-Type: text/turtle

    ... #RDF payload ...


    DELETE DATA { { .. RDF payload .. } }
```

7.1.2.7.1.3 DELETE/INSERT Operation

The `DELETE/INSERT` operation can be used to remove or add triples from/to the Graph Store based on bindings for a query pattern specified in a `WHERE` clause. In other words, the `DELETE/INSERT` operation can be used to overwrite a specific triples stored in a Graph Store. According to the definition of `DELETE/INSERT` operation, `DELETE` and `INSERT` operation are executed sequently and if the `DELETE` clause is omitted then only `INSERT` operation only inserts data. Same action happens as `INSERT` operation is omitted.

In order to fit the requirement for updating (overwriting) a specific semantic instance, it is assumed that both `DELETE/INSERT` operation can be executed sequently and guarantee that `DELETE` and `INSERT` are supposed to always be executed.

The operation can be executed using HTTP verbs by running below command:

- Case1: a specific graph is targeted by <graph_uri> (change graph):

```
POST/rdf-graph-store?graph=_graph_uri_.. HTTP/1.1
    Host: host_name_
    Content-Type: text/turtle

    ... #RDF payload ...

    DELETE DATA { GRAPH <graph_uri> { .. RDF payload 1.. } }
    INSERT DATA { GRAPH <graph_uri> { .. RDF payload 2.. } }
```

- Case2: the default graph is targeted (change graph):

```
POST/rdf-graph-store?default HTTP/1.1
    Host: host_name_
    Content-Type: text/turtle

    ... #RDF payload ...

    DELETE DATA { { .. RDF payload 1.. } }
    INSERT DATA { { .. RDF payload 2.. } }
```

- Case3: the default graph is targeted (add new graph):

```
POST/rdf-graph-store?default HTTP/1.1
    Host: host_name_
    Content-Type: text/turtle

    ... #RDF payload ...

    INSERT DATA { { .. RDF payload .. } }
```

7.1.2.8 Device Abstraction

Device abstraction is a process of mapping between a set of Device Application Information Models and an Abstract Application Information Model according to a specified set of rules. It allows to communicate with multiple, different but semantically similar devices through a virtual device that offers the functionality of the abstracted Application Information Model.

7.1.2.9 Data Repository

Data repository basically stores new data. In addition, it also provides functions to support the search, modification and deletion of the stored data.

7.1.2.10 M2M Data Collection

From devices with sensors and/or gateways, raw data are collected and stored in data repository.

7.1.2.11 Data Analytics

Data analytics is a process of deriving desired results through examining data. It may assist reasoning for semantic (mash-up) services. It also provides the capability for calculation as well as interpreting and applying service logic (e.g., rules/policies) based on semantic descriptions.

7.2.1 Enhancing a REST based (e.g. ETSI M2M) system into a Semantic M2M System

7.2.1.1 System Overview

Figure 7.2.1.1-1 taken from [i.2] describes a high-level architecture of Semantic M2M with internal components. The architecture uses a REST/resource based M2M system, as e.g. specified by ETSI M2M, as a basis. Figure 7**.2.1.1-1** [i.2] shows RESTful access of a typical ETSI M2M Application via the mIa reference point to a ETSI M2M Network Service Capability Layer (NSCL). Additionally to the ETSI architecture resources are enhanced (annotated with) semantic information and a Semantic engine is included into the ETSI M2M NSCL to process semantic queries that can be issued by applications that are capable to handle semantic information (Semantic Application).

Such a REST based M2M system that is enhanced with semantic capabilities will be called a Semantic M2M system.

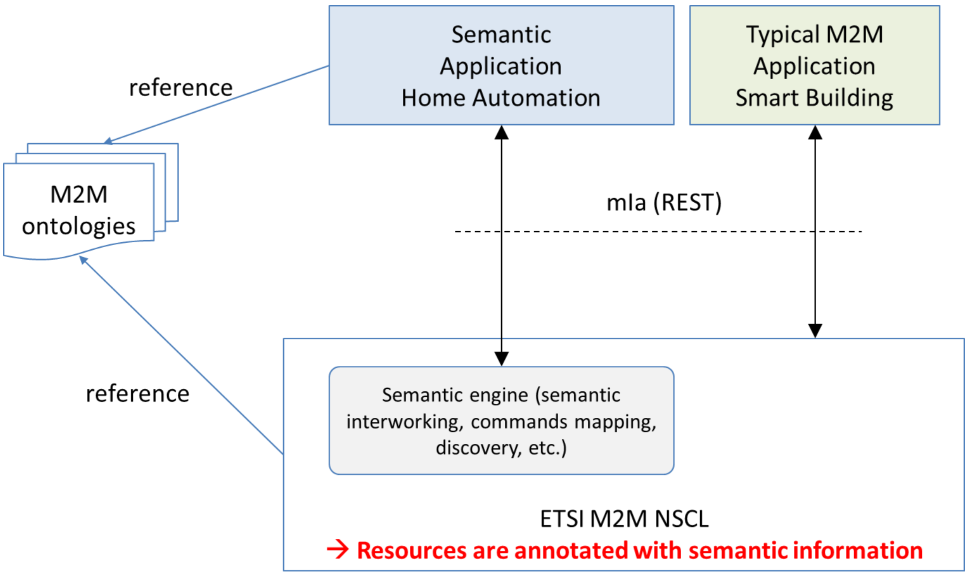


Figure 7.2.1.1-1: Semantic M2M system overview

Semantic engine:

Semantic engine plays a key role in Semantic M2M system. Similar to interworking proxy capabilities (xIP) in NSCL [i.1], semantic engine can be deployed in NSCL. For discovery, the engine receives a semantic query; handles the query and returns results.

Semantic engine provides functionalities as follows:

- validate semantic attributes (according to semantic model, e.g. RDF and OWL, either defined by ETSI M2M or outside of ETSI M2M);
- process semantic queries, for example decomposing a query into multiple sub-queries, aggregating the results from sub-queries.

M2M ontologies:

M2M ontology is a formal description of M2M resources, of the structures of things, properties, processes and their relationships in a domain. Within a Semantic M2M system M2M ontologies can be referenced by SCLs and Applications.

7.2.1.2 Semantic Annotation

Semantic annotation of M2M resources is a method for adding semantic information to M2M resources so that provides consistent data translation and data interoperability to heterogeneous M2M applications. Semantically annotated M2M resources can be contacted by an M2M application that understands what data are provided by the resources and what these data means. These annotations provide more meaningful descriptions and expose M2M data than traditional M2M system alone.

In brief, M2M resources usually consist of sensor devices monitoring and reporting a specific data and actuators executing a given command. Comparing with other semantic services, such as Semantic Web and Semantic Sensor Web, semantic M2M needs to provide semantic information for both data and commands.

In many cases, M2M resources are annotated with semantic information using RDF because RDF provides a general, flexible way to decompose any knowledge into discrete pieces and can be stored in many different formats. In addition, RDF is useful to encode information about relations between things which includes a lot of semantic information. A triple store is usually selected in order to store and retrieve such relational information. However, ETSI M2M uses a hierarchical resource tree to store resource information and provide discovery of these resources.

The first step towards a semantic M2M system is to annotate semantic information to its managing resources. Semantic information is retrieved from the relations between M2M resources and can be annotated as an attribute of the resources. The ETSI M2M system uses a hierarchical tree structure to store and represent its resources. Thus, in the ETSI M2M system, semantic information can be retrieved from the relations between M2M resources and embedded as an attribute of the resources.

- The *semanticInfo* attribute contains the semantic description of the thing that is represented by the annotated resource in the M2M system. The semantic description is a reference to a concept within an ontology. This ontology shows what is the meaning of the thing. Since a concept with the same name might appear in another ontology too this semantic description is expressed with namespace prefix to avoid name conflicts.

- To describe relationships with other things, the *relations* attribute can be introduced, this attribute can have a pair format, i.e. <*relation* : *link to other thing*>.

EXAMPLE 1: A Zigbee temperature sensor that is controlled by a Zigbee controller 1 can be described in the following format:

EXAMPLE 2: "m2m:isControlled - Zigbee-Controller-1"..

These *senaticInfo* and *relations* attributes are a available for resource discovery so that any applications can easily discover ETSI M2M resources without any domain specific expert knowledge.

The object of *relation*, i.e. *link to other thing*, can be any type of resources. All type of things in ETSI M2M, physical thing, abstract devices and virtual things, can be used as a *link to other thing*. This field should be an (absolute or relative) URI pointing to another ETSI M2M resource. In the previous example, the Zigbee controller 1 is an actual thing that exists in the ETSI M2M system. Virtual things can be used to add more semantic information to the actual things.

For example, if a sensor is deployed in a room-1, semantic annotation between the sensor and the room-1 can add semantic information about the location. In this example, room-1 is not a physical object but a virtual thing. Through annotating the relationship between the sensor and room-1, user can discovery the sensor when asking sensors in the room-1. The relationship can be described in the following format:

EXAMPLE 3: "m2m:isDeployed - room-1".

A new semantic information can be easily created, updated and deleted through using ETSI M2M supported RESTful commands, CREATE, UPDATE and DELETE, respectively. In order to avoid name conflicts between vocabularies used in *semanticInfo* and *relations*, namespace prefix and the namespace URI are also defined. For example, *isDeployed* could be defined differently in two different domains: ***sns:isDeployed*** and ***m2m:isDeployed***. This means that other could define *isDeployed* with other namespace prefix. If semantic information is provided together with this namespace prefix, a reader could be able to understand that they are different semantic information even though they have the same name.

The namespace URI can also be introduced as an attributed. The following figure shows how *semantinInfo, relations* and *namespaceURI* can be expressed within the ETSI M2M resource tree. In this case, <*namespaceURIs*> contains a list of namespace URIs. For example:

```
<m2m=http://www.m2m-semantic.org/sensor#>
<sns=http://www.homeautomation.org/sensor#>
```

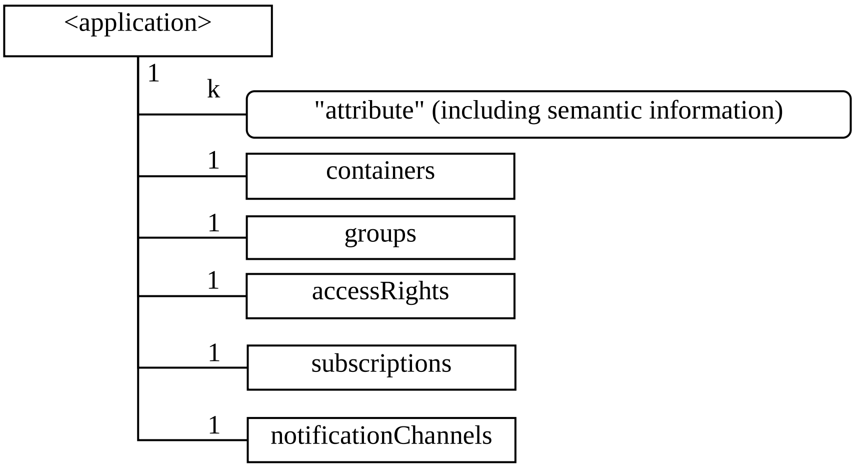


Figure 7.2.1.2-1: An example of Namespace URI as part of a sub-resource of the application resource

The attributes used for annotating semantic information are described in table 7.2.1.2-1.

Table 7.2.1.2-1: Attributes for annotating semantic information

Name	Description
semanticInfo	This attribute contains the semantic description of the thing. This ontology shows what is the meaning of the thing.
Relations	This attribute is used to describe the relationships with other things. This attribute can have a pair format, i.e. <*relation* : *link to other thing*>. For example, if a zibgee sensor is controlled by a Zigbee controller #1, <m2m:controlledBy - URI to Zigbee controller #1> can be a way of expressing a relationship to Zigbee controller.
namespaceURIs	This attribute is used to describe namespace URIs. This attribute contains the information about namespace prefix and the namespace URI. For example, the m2m namespace with the URI http://www.etsi-m2m.org can be expressed as follows: m2m=http://www.etsi-m2m.org/sensor#.

Now semantic information of resources is stored in the ETSI M2M system. However, since legacy M2M systems do not support semantic queries, such as SPARQL, they need to provide a way to deliver semantic queries to the ETSI M2M system. For this purpose, the semantic M2M system should provide a capability to provide a unified access point of a semantic query to M2M applications.

When a semantic query arrives at the semantic engine, it parses the query and generates RESTful sub-queries. The engine then processes the RESTful queries and gets resources from SCLs. The returned resources are checked for semantic information.

7.2.1.3 Semantic Mashups for Virtual Things

In the domain of Web Service, mashup is a method composing web data from more than one web resources to create a new service. Examples include metacrawlers that blends web search results from multiple search engines and news aggregators that aggregate integrated web contents in a single location. Similarly, the mashup technique can be used to create a new M2M resource in the M2M System.

In the Semantic M2M System, a M2M application can publish "virtual things" that act similar to physical resources and provide new information such as: number of vehicles that passed during the last minute/hour, average speed of vehicles, etc. These "virtual things" can be searched and discovered in the M2M System same as other M2M resources. However, in contrast to the physical things, virtual things are only implemented as software and do not require a network connectivity.

When a new virtual thing is registered (or published) to the Semantic M2M system, a list of member M2M resources is stored together as an attribute of the thing. If the virtual thing collects information dynamically at the time of receiving a query, a pre-programmed query that collects member resources is also stored along with other information.

Once a virtual thing is added to the NSCL, it is handled and processed the same as all other M2M resources. This means that virtual things are exposed to M2M applications to be discovered. An example of the semantic virtual mashup process is shown in figure 7.2.1.3-1.

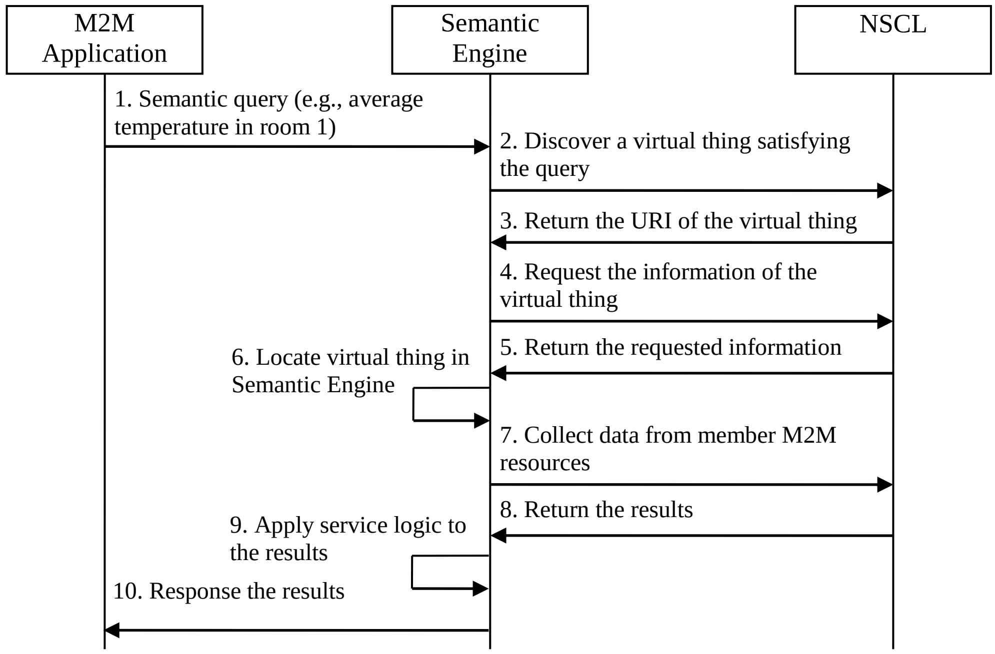


Figure 7.2.1.3-1: Semantic virtual mashup procedure

Step 1: M2M application sends a semantic query to the Semantic M2M system, for example, "Get the temperature of the room 1".

Step 2: semantic engine handles this like a normal semantic query so that sends a discovery request to the NSCL.

Step 3: the NSCL returns the URI of a virtual thing that provides the temperature of the room 1.

Step 4: semantic engine sends a request to the NSCL to retrieve the information of the virtual thing, i.e. service logic, mashup type (either static or dynamic) and pre-programmed queries.

Step 5: the NSCL returns the requested information.

Step 6: semantic engine instantiates the virtual thing. For a virtual thing that is frequently requested, it can be cached in semantic engine and handles the request directly.

Step 7: the virtual thing at semantic engine collects required data from its member resources using the pre-programmed query.

Step 8: the NSCL returns the results from member resources.

Step 9: the virtual thing applies its service logic (e.g. calculating the average value) to the received data and calculates the results.

7.2.2 Introduction to OGC Sensor Web Enablement

7.2.2.1 Overview

A sensor Web simply refers to web accessible sensor networks for enabling an interoperable usage of sensor resources by enabling Web-based discovery, access, tasking and alerting using standard protocols and Application Program Interfaces (APIs). It enables the advancement of Web applications through improved situation awareness.

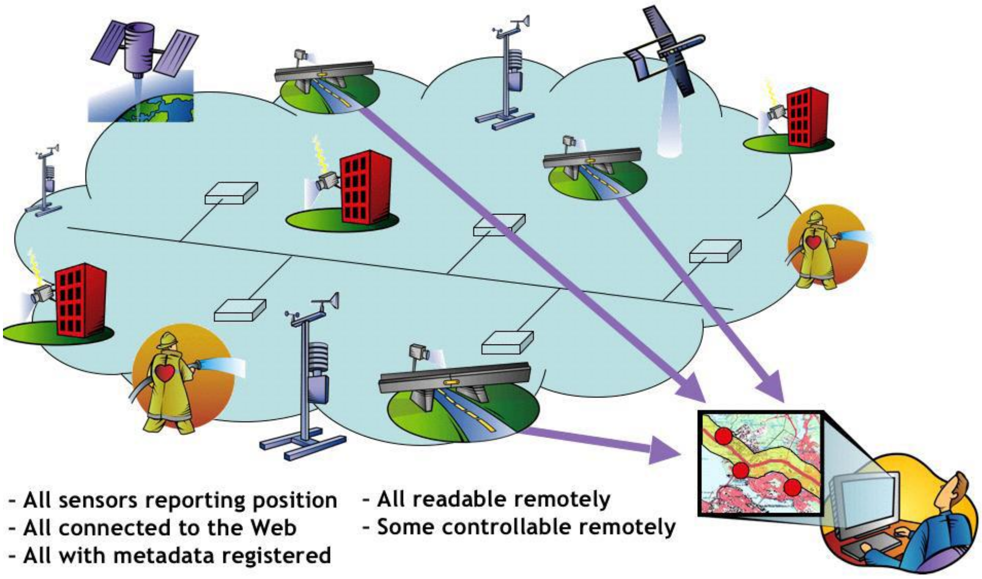


Figure 7.2.2.1-1: Sensor Web Concept

The Open Geospatial Consortium (OGC) [i.3] has worked to specify interoperability interfaces and metadata encodings that enable real time integration of heterogeneous sensor Web into the information infrastructure. The OGC has developed publicly available encoding and interface standards. These standards can be used so that developers create applications, platforms, and products involving Web-connected devices so as to make complex spatial information and services accessible and useful with all kinds of applications.

The Sensor Web Enablement (SWE) [i.4], a suite of web service interfaces and communication protocols, presents many opportunities for adding a real-time sensor dimension to the Internet and the Web as a unique and revolutionary framework of open standards for exploiting Web-connected sensors and sensor systems of all types. This has extraordinary significance for various vertical applications and services in M2M domains.

7.2.2.2 Sensor Web Enablement (SWE) Languages

The models, encodings, and services of the SWE architecture enable implementation of interoperable and scalable service-oriented networks of heterogeneous sensor systems and client applications. The OGC's SWE initiative has focused on developing standards to enable the discovery, exchange, and processing of sensor observations, as well as the tasking of sensor systems. The functionality that OCG has targeted within a sensor Web includes [i.5]:

- Discovery of sensor systems, observations, and observation processes that meet an application's or user's immediate needs.
- Determination of a sensor's capabilities and quality of measurements.
- Access to sensor parameters that automatically allow software to process and geo-locate observations.
- Retrieval of real-time or time-series observations and coverage in standard encodings tasking of sensors to acquire observations of interest.
- Subscription to and publishing of alerts to be issued by sensors or sensor services based upon certain criteria.

In SWE, the OGC has made a framework of open standards for exploiting Web-connected sensors and sensor systems of all types. This framework is called a Sensor Web, and refers to web accessible sensor networks and archived sensor data that can be discovered and accessed using standard protocols and APIs. SWE is composed of three languages and four service specifications:

- **Observations & Measurements (O&M) [i.6]:** Standard models and Extensible Markup Language (XML) schema for encoding observations and measurements from a sensor, both archived and in real time:
  - An observation is an event with a result that has a value describing some phenomenon. The observation is modeled as a feature within the context of the ISO/OGC General Feature Model. An observation feature binds the result to the feature of interest, upon which it was made. An observation uses a procedure to determine the value, which may involve a sensor or observer, analytical procedure, simulation or other numerical processes.
- **Sensor Model Language (SensorML) [i.7]:** Standard models and XML schema for describing sensor systems and processes associated with sensor observations in order to provide information required for the discovery of sensors, location of sensor observations, processing of low-level sensor observations, and listing of taskable properties, as well as supporting on-demand processing of sensor observations:
  - Within SensorML, everything including detectors, actuators, filters, and operators are defined as process models. A Process Model defines the inputs, outputs, parameters, and method for that process, as well as a collection of metadata useful for discovery and human assistance. The inputs, outputs, and parameters are all defined using SWE Common data types. Process metadata includes identifiers, classifiers, constraints (time, legal, and security), capabilities, characteristics, contacts, and references, in addition to inputs, outputs, parameters, and system location.
- **Transducer Model Language (TransducerML or TML) [i.8]:** A conceptual model and XML schema for describing transducers and supporting real-time streaming of data to and from sensor systems:
  - TML defines:
    - A set of models describing the hardware response characteristics of a transducer.
    - An efficient method for transporting sensor data and preparing it for fusion through spatial and temporal associations.
- **Sensor Observations Service (SOS) [i.9]:** A standard Web service interface for requesting, filtering, and retrieving observations and sensor system information as the intermediary between a client and an observation repository or a near real-time sensor channel:
  - The SOS includes three core operations:
    - The GetObservation operation provides an interface to query over observation data and returns an O&M document.
    - The DescribeSensor operation provides an interface to query for the description of a sensor and returns a SensorML document.
    - The GetCapabilities operation provides an interface to query for the description of a SOS. GetCapabilities allows clients to retrieve service metadata about a specific service instance and returns a GetCapabilites response document.
- **Sensor Planning Service (SPS) [i.10]:** A standard Web service interface for requesting user-driven acquisitions and observations as the intermediary between a client and a sensor collection management environment.
- **Sensor Alert Service (SAS) [i.11]:** A standard Web service interface for publishing and subscribing to alerts from sensors.
- **Web Notification Services (WNS):** A standard Web service interface for asynchronous delivery of messages or alerts from SAS and SPS Web services and other elements of service workflows.

The OGC SWE standards became a basis for the World Wide Web Consortium (W3C) Semantic Sensor Network (SSN) - Incubator Group (XG) [i.12] to study and recommend methods for using the ontology to semantically enable applications through the extension of the SenosrML for supporting semantic annotations.

As the OGC has identified the need for standardized interfaces for sensors in the Web of Things (WoT), the OGC has created a new Standards Working Group (SWG) on the sensor Web interface for the Internet of Things (IoT). The **sensor Web interface for the IoT SWG** [i.13] aims to develop such a standard based on existing WoT portals with consideration of the existing OGC SWE standards. This group is developing a candidate standard for access to sensor observations including location information well-suited to IoT and WoT deployment environments.

7.2.2.3 Semantic annotations in OGC standards

Introducing annotation

In the OGC Best Practice [i.14], the OGC has introduced the notion of **semantic annotation**.

NOTE 1: Annotation of Web Services or data compliant to OGC standards refers to the task of attaching meaningful descriptions to the service and the served geospatial data or processes. The OGC has extended the expressiveness of such annotations by including **more sophisticated (semantic) descriptions**.

Semantic annotations enable data providers to connect the standardized service descriptions to the modeled knowledge. Semantic annotations establish a connection between the geospatial resource, its metadata, and the ontology. Figure 7.2.2.3-1 illustrates the three different levels of (semantic) annotations which are possible for OGC Web Services.

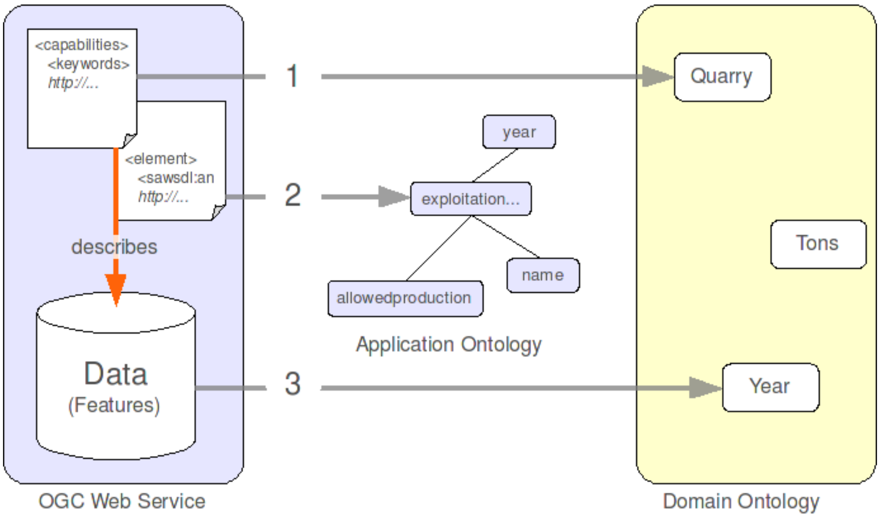


Figure 7.2.2.3-1: Semantic annotations at three different levels

It is possible to distinguish between three locations where particular information about the resource can be acquired.

- *The first source of information*: the Capabilities-document which comprises functional properties telling the user how to access and invoke the service, as well as some resource **metadata** with information about the service provider, licensing, a title and description, or a keyword section.
- *The second source of information*: a XML-based schema representing the **data model**, which comprises a description of the data model with focus on syntax and structure.

NOTE 2: The **metadata** and the **schema** are describing the underlying data, and therefore explicitly linked (the orange arrow in figure 7.2.2.3-1).

- *The third source of information*: the **data entities** itself, encoded in the format predefined in the data model specified in the data schema.

There are two types of references (see figure 7.2.2.3-2):

- **Domain reference:** links between a local, application-specific model and a global, shared vocabulary. It can be also expressed in form of complex rules.
- **Model reference:** bridge different languages. It is always a link between two models (e.g. XML Schema or UML modelling data, RDF modelling a domain vocabulary, and so on) and a unique URI which points to the corresponding element in another model.

NOTE 3: The two types of references can overlap. In many cases a domain reference is in the same time also a model reference, since the reference performs both tasks (bridging in between two languages and linking from local to global). In this case, the domain reference should be used.

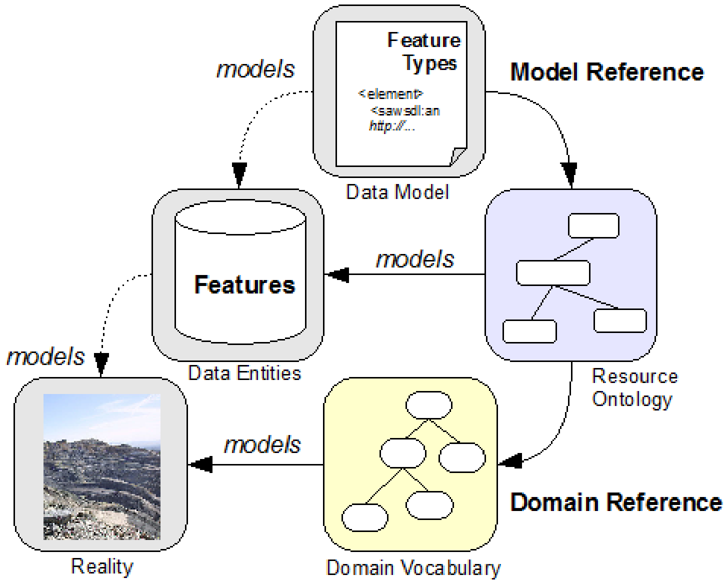


Figure 7.2.2.3-2: Two types of references

Semantic annotations at three different levels

Modeling knowledge and publishing it in a well-defined and machine interpretable format can result in increased usability of OGC Web Services. OGC considers ontologies as most promising format to capture such knowledge. But the possibilities to connect ontologies with OGC services are manifold. The following discuss how to annotate OGC services on the already mentioned three levels:

- **Level 1 - The service metadata:**
  - Adding knowledge model references in one of the existing sections in the **data** and/or **services metadata** (e.g. by extending the keyword section of OWS Capabilities) as the most pragmatic and still useful approach.

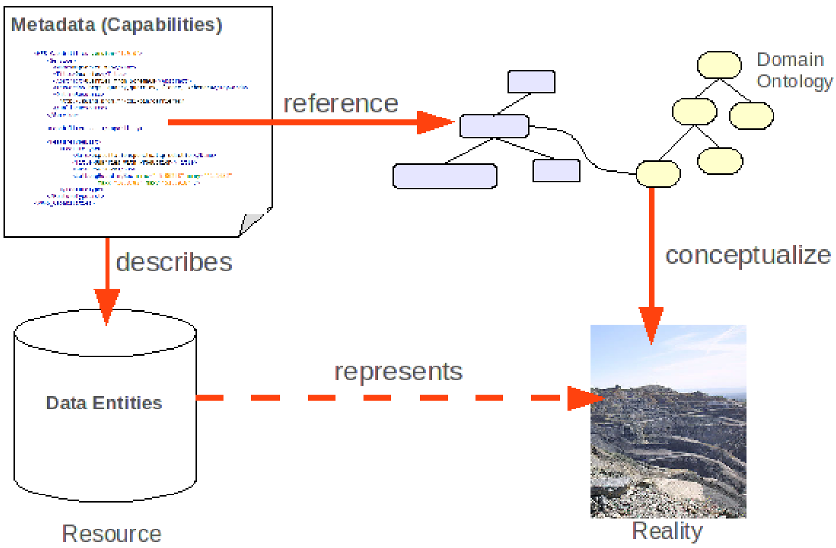


Figure 7.2.2.3-3: Annotations as part of the resource metadata

- **Level 2 - The data models and process descriptions:**
    - Semantic annotation on the **data models** and the **service operations** in order to relate service metadata more efficiently to domain knowledge. Considering data structures (e.g. Geography Markup Language (GML) [i.15] application schema) for semantic annotations enables reasoning on data model level.

NOTE 4: The GML is an XML grammar for expressing geographical features.

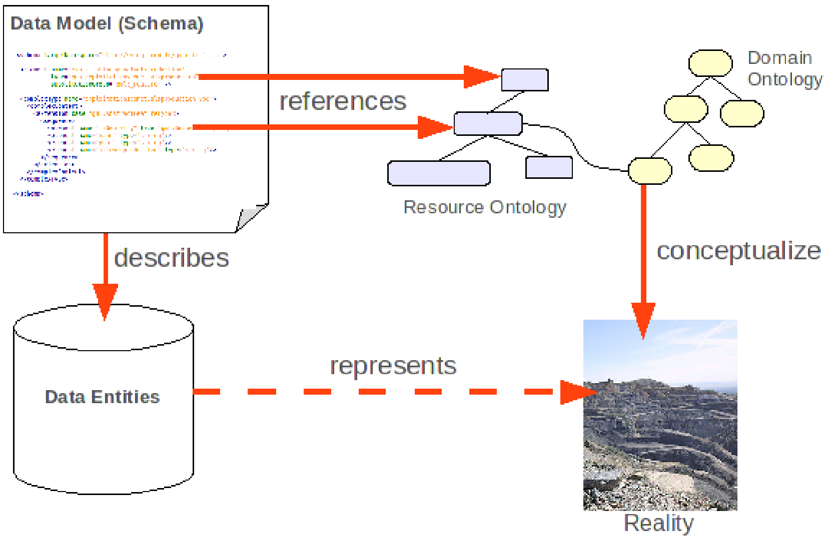


Figure 7.2.2.3-4: Referencing elements in the data model to domain ontologies

- **Level 3 - The actual data instances in the database:**
    - Semantic annotations are also possible on the **data entity** level (e.g. GML features and feature attributes in OGC Web Service).

The three annotations levels differ in their potential. This is due to different reasoning capabilities, i.e. the abilities to infer either new knowledge (making implicit knowledge explicit) or to identify conflicts in existing models. Accordingly, the applications for semantically supported OWS discovery and workflow validation vary.

7.2.3 Introduction to W3C Semantic Sensor Network Ontology

7.2.3.1 Overview

The World Wide Web Consortium (**W3C**) **Semantic Sensor Network (SSN) - Incubator Group (XG)** [i.12], [i.16] had worked on two main objectives:

1) to develop an ontology to describe sensors and sensor networks; and

2) to study and recommend methods for using the ontology to semantically enable applications through the extension of the Sensor Model Language (SenosrML) [i.7] developed in the Open Geospatial Consortium (OGC) [i.3] for supporting semantic annotations.

The W3C SSN-XG had recognized that several Sensor Web Enablement (SWE) [i.17] standards developed by the OGC should be replaced by approaches based on the semantic Web languages developed by W3C. Furthermore, the group were also motivated the fact that the development of ontologies and of mechanisms to support semantic annotations for sensors could improve interoperability and integration of various services with enhanced capabilities such as reasoning and automation.

For developing an ontology, the W3C SSN-XG considered the following two axes for identifying a set of use cases:

1) various users who use and manipulate data as well directly manage sensor and sensor networks; and

2) technical evolution toward the integration of sensor Web and semantic Web. The following provides four groups of use cases identified for the modeling of sensors in the group:

- **Data discovery and linking** to find all observations that meet certain criteria, and possibly link them to other external data sources.
- **Device discovery and selection** to find all the devices that meet certain criteria.
- **Provenance and Diagnosis** to provide extra information about the instrument to better evaluate or process the data.
- **Device Operation Tasking and Programming** to command a device's operation using its description and information on its conditions of use.

Based on use cases, the W3C SSN-XG had developed the ontology (called **SSN ontology**, see clause 7.2.3.2) which provides a high-level schema to describe sensor devices, their operation and management, observation and measurement data, and process related attributes of sensors. The SSN ontology supports a domain-independent and end-to-end model for sensing applications, and it can be used with domain ontologies and other ontologies to model the observation and measurement data produced by the sensors. For ontology specification, Web Ontology Language Description Logic (OWL DL) was selected to encode sensor descriptions considering mapping between the ontologies and OGC models.

7.2.3.2 The Semantic Sensor Network Ontology

Introduction

Before starting the work on the SSN ontology, the W3C SSN-XG extensively reviewed ontologies and data models describing sensors and their capabilities as well as observation, using attributes [i.18]. After reviewing them, the group identified that there are important concepts that should be included, but found that none of the existing ontologies supported all of those required concepts. Therefore, the group started to newly develop a formal OWL DL ontology for modeling sensor devices (and their capabilities), systems and processes.

NOTE 1: Section 4 in [i.16] provides details reviewed senor ontologies and observation ontologies as well as surveys of various ontologies.

For describing the physical and processing structure of sensors, the ontology is based around concepts of systems, processes, and observations. There had been considered various kinds of sensors (e.g. a device, computational process or combination) including typical physical sensing devices and anything for estimating or calculating the value of a phenomenon. The representation of a sensor in the ontology links the followings:

1) what it measures (the domain phenomena);

2) the physical sensor (the device); and

3) its functions and processing (the models).

NOTE 2: The ontology is available as a single OWL file: SSN ontology [i.19] and a semi-automatically generated documentation [i.18] derived from it is also provided as a standalone document.

NOTE 3: In section 5 in [i.16], there are five worked examples to illustrate different parts of the SSN ontology: University deployment, Smart product, Wind sensor, Agriculture Meteorology, and Linked Sensor Data.

Ontology structure

The SSN ontology revolves around the central Stimulus-Sensor-Observation pattern.

NOTE 4: The Stimulus-Sensor-Observation Ontology Design Pattern aims at all kind of sensor or observation based ontologies and vocabularies for the Semantic Sensor Web and especially Linked Data.

- **Stimuli:** detectable changes in the environment (i.e. in the physical world).

- **Sensors:** physical objects that perform observations (i.e. they transform an incoming stimulus into another, often digital, representation).

- **Observations:** act as the nexus between incoming stimuli, the sensor, and the output of the sensor (i.e. a symbol representing a region in a dimensional space).

Several conceptual modules (see figure 7.2.3.2-1) build on the pattern to cover key sensor concepts. Figure 7.2.3.2-1-2 which shows the relationships between modules contains an overview of the main classes and properties inside the ontology modules.

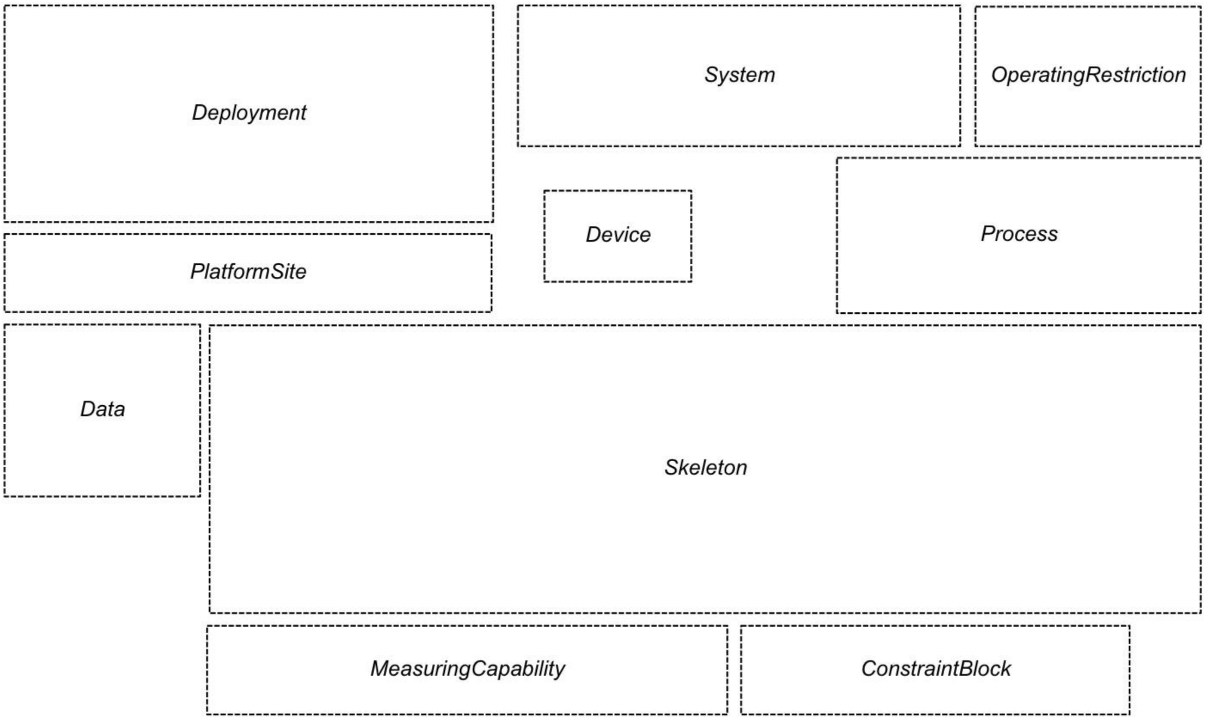


Figure 7.2.3.2-1: Overview of the Semantic Sensor Network ontology modules

The ontology can be used for a focus on any (or a combination) of a number of perspectives:

- **Sensor perspective:** what senses, how it senses, and what is sensed.

- **Data or observation perspective:** observations and related metadata.

- **System perspective:** systems of sensors.

- **Feature and property perspective:** features, properties of sensors, and what can sense those properties.

The modules allow further refining or grouping of these views on sensors and sensing. The description of sensors may be detailed or abstract. The ontology does not include a hierarchy of sensor types.

NOTE 5: Domain experts can define further details. Thus, the ontology could be a simple hierarchy or a more complex set of definitions based on the workings of the sensors.

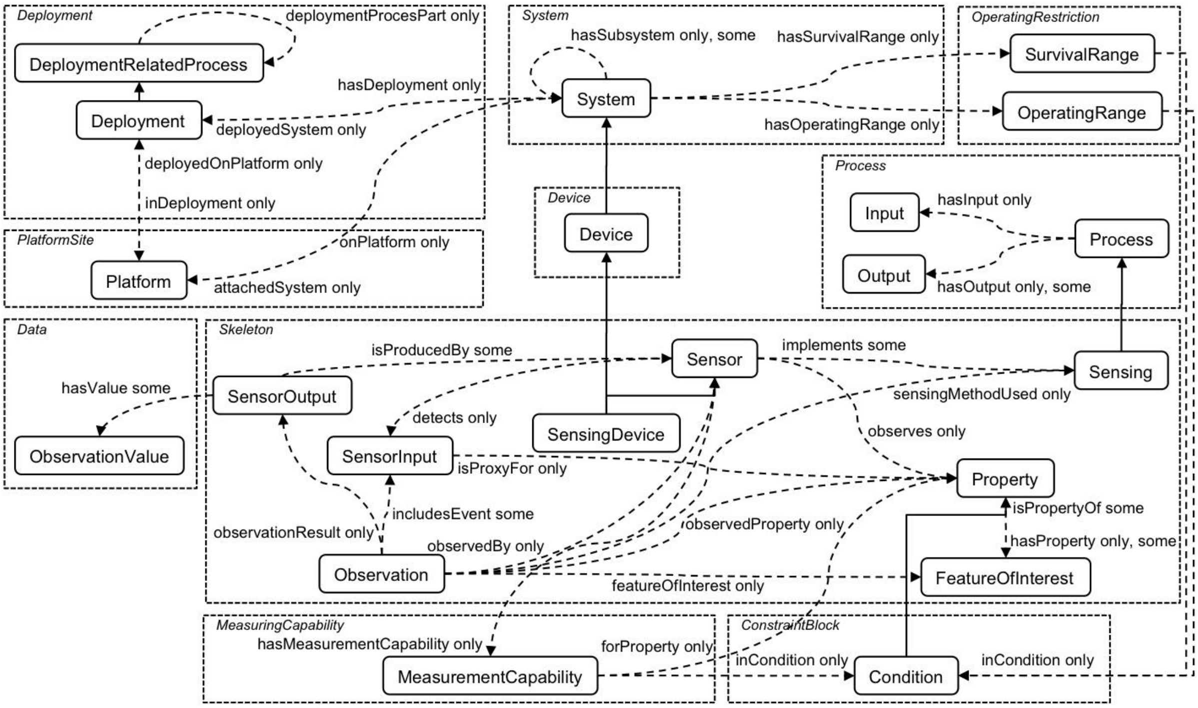


Figure 7.2.3.2-2: Overview of the Semantic Sensor Network ontology classes and properties

The modules contain the classes and properties that can be used to represent particular aspects of a sensor or its observations: for example:

- Sensors.
- Observations.
- Features of interest (i.e. entities in the real world that are the target of sensing).
- The process of sensing (i.e. how a sensor operates and observes).
- How sensors are deployed or attached to platforms.
- The measuring capabilities of sensors.
- Their environmental, and survival properties of sensors in particular environments.

Figure 7.2.3.2-3 shows a detailed enumeration of these properties.

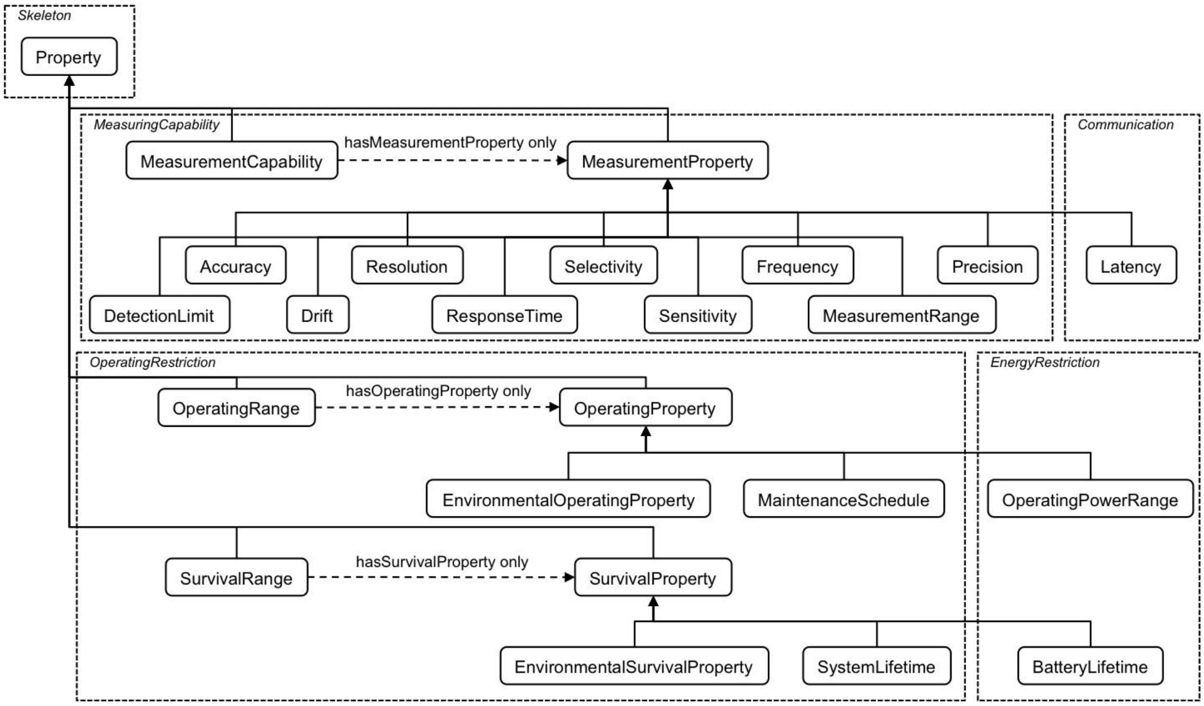


Figure 7.2.3.2-3: Enumeration of the measurement, environmental and survival properties

The main classes of the SSN ontology have been aligned with classes in the DOLCE Ultra Lite (DUL) foundational ontology to facilitate reuse and interoperability [i.20]. Figure 7.2.3.2-4 shows in blue arrows the subclass properties used to align these two ontologies.

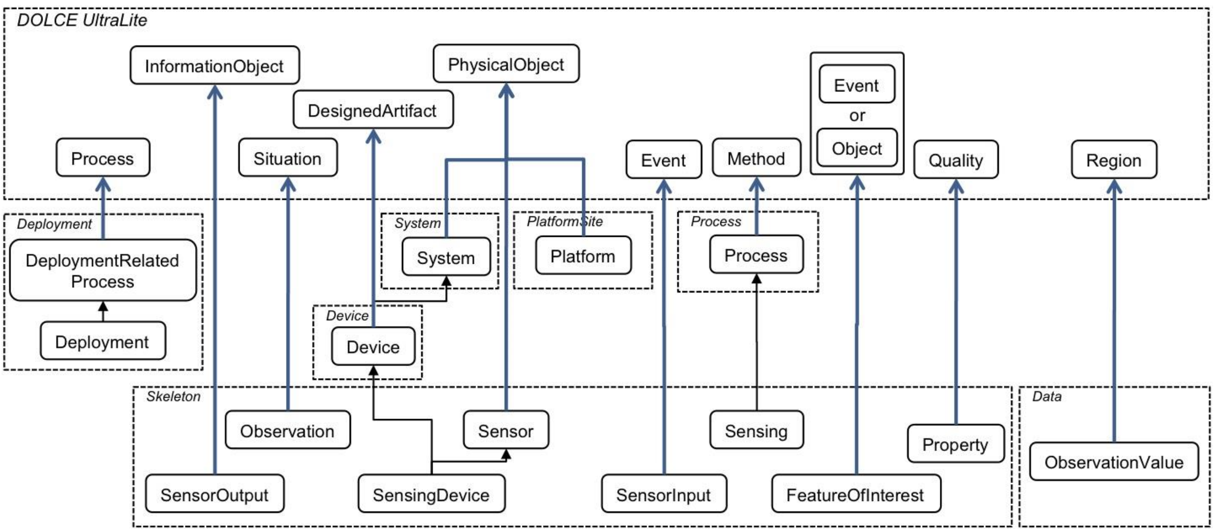


Figure 7.2.3.2-4: Alignment of the Semantic Sensor Network ontology to DOLCE Ultra Lite

Finally, figure 7.2.3.2-5 shows how the ontology modules defined above support the use cases described in the clause 7.2.3.1.

- Orange color users developing Semantic Sensor Web applications: more specifically interested in the modules connected to the Data Discovery and Linking and Provenance and Diagnosis uses cases.
- Green color users developing Semantic Sensor Web applications: need the modules connected to the Device Discovery and Selection and Device Operation Tasking and Programming uses cases.

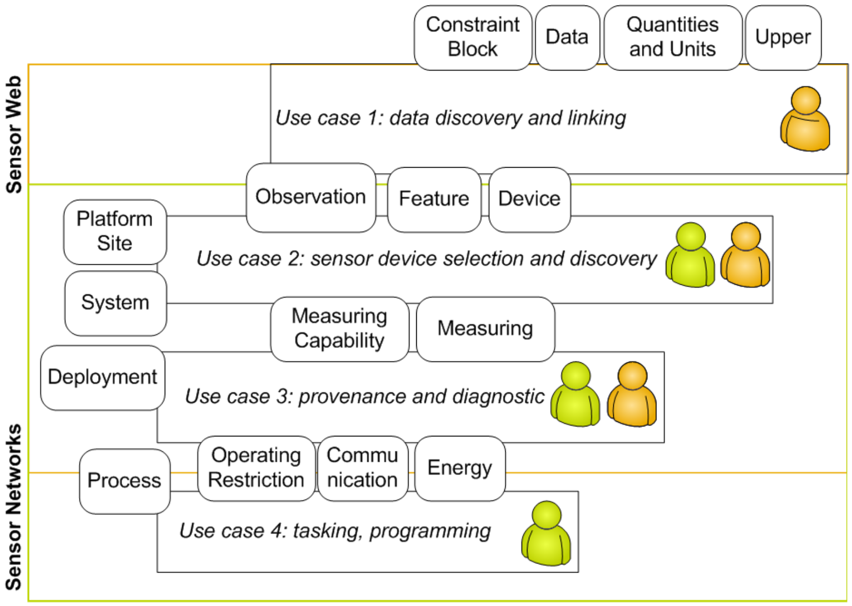


Figure 7.2.3.2-5: Use cases and ontology modules

7.3 Key Issues on Standardizing Semantics and Ontologies

7.3.1 How to share common understanding of the structure of information among m2m nodes?

For example, suppose several different Home Automation Systems contain temperature information or provide Energy Saving applications. If these applications share and publish the same underlying ontology of the terms they all use, then applications can extract and aggregate information from these different applications.

oneM2M has to provide a solution for the following issues:

- Should oneM2M provide the ontology for a specific domain or verticals?
    - Maybe 'YES' for the oneM2M common service layer domain but in general 'NO'. oneM2M needs to provide a means for sharing pre-defined or existing ontologies.
- Other issues are FFS.

7.3.2 How to enable the reuse of domain knowledge?

For example, many different domains need to represent time in their model. This representation includes the notions of time intervals, points in time, relative measures of time, and so on. If one vertical develops such an ontology, others do not need to develop another ontology but can simply reuse it for its domain.

Additionally, if oneM2M or a vertical need to build a large ontology, several existing ontologies can be integrated for the large domain.

oneM2M has to provide a solution for the following issues:

- How to integrate several ontologies in order to build an integrated large ontology.
- How to import (or reference) ontologies defined outside of oneM2M.

7.3.3 How to enable evolving ontologies?

Ontologies should be changed easily over the time if the knowledge about a domain changes. oneM2M should consider flexibility when designing ontologies or importing external ontologies.

7.3.4 How to analyse domain knowledge (ontologies)?

Ontology itself is a means for developing semantic M2M systems. Developing an ontology is a process for defining a set of semantic concepts and the relationships between concepts. For example, let us consider that there is an ontology of home appliances (such as refrigerator and washing machine). This ontology can then be used as a basis for some M2M applications in energy savings: One application could create suggested schedules for the home appliances during the day in order to save energy. Another M2M application could analyse available home appliances and suggest which appliances are to be replaced.

oneM2M has to provide a solution for the following issues:

- What platform mechanisms are needed to support such analyses?

7.3.5 How to make semantic annotations?

As described in clause 7.1.2.6, "semantic annotation" of M2M resources is a method for adding semantic information to M2M resources. Semantic annotations can be used as a basis to provide consistent data translation and data interoperability to heterogeneous M2M applications.

Since oneM2M represents M2M resources as a tree structure, methods for adding semantic information to corresponding M2M resources have to be standardized.

NOTE: Typically semantic information is represented using RDF or OWL.

oneM2M has to provide a solution for the following issues:

- A method for annotating the oneM2M architecture resource tree with semantic information.
- A solution showing how to describe semantically annotated M2M resource using RDF or OWL.

7.3.6 How to generate (or register) new M2M resources/applications based on existing M2M resources/applications?

According to the definition in clause 7.1.2.5, "semantic mash-up" is a functionality to support new services through the creation of new virtual devices, which do not exist in the physical world, by obtaining semantic information through semantic descriptions from existing M2M resources in the M2M System.

oneM2M has to provide a solution for the following issues:

- How can new virtual devices be created?
- Which attributes and resources does a virtual device have to include.
- What is the life-cycle of a virtual device and what are the relationships with other M2M resources out of which the virtual device is composed.
- How to define the role of virtual devices?
- What are the impacts to existing ontologies?

7.3.7 How make reasoning?

As described in clause 7.1.2.2, "reasoning" is a mechanism to derive new implicit knowledge from semantically annotated data based on a set of asserted facts or axioms. For example, if the semantic annotation of resource "R" indicates that "R" is an instance of class "A", and the referenced ontology indicates that "A" is sub-class of "B", then using "reasoning" can derive that "R" belongs to class "B".

In practice, the semantic annotation of resources and the semantic query request from applications may use different vocabularies in same ontologies or different ontologies, e.g. the semantic annotation for a blood pressure monitor indicates that it is an instance of "blood pressure monitor" and its "precision" is "high", while the semantic query requests may use a different vocabulary "advance blood pressure monitor" which is sub-class of "blood pressure monitor" defined as "blood pressure monitor" with "high" "precision". Without "reasoning", M2M applications may not be able to correctly find target M2M resources via semantic query.

oneM2M has to provide a solution for the following issues:

- Should reasoning function of semantic engine be provided in oneM2M systems?
  - Maybe "YES" since it is too complicated for applications to only use basic concepts with multiple logical rules as filter in each semantic query request.
- Should reasoning function of semantic engine update the semantic annotation information based on reasoning results?
  - Maybe "YES" since updating semantic annotation information via reasoning could improve the speed of semantic query.
- Should reasoning function of semantic engine be able to provide reasoning among multiple compatible ontologies?
- Should reasoning function of semantic engine only be triggered along with semantic query process?
  - Maybe "NO" since data analytics may also need the assistants of reasoning function of semantic engine.
- How to trigger reasoning function of semantic engine?

7.3.8 How to enable semantic rule?

In general, ontologies concentrate on classification methods, putting an emphasis on defining 'classes', 'subclasses', on how individual resources can be associated to such classes, and characterizing the relationships among classes and their instances.

Semantic Rules, on the other hand, concentrate on defining a general mechanism on generating new relationships based on existing concepts and relationships in ontologies using logical manners. Typical formats for representing semantic rules include RIF [i.33], SWRL [i.34] and SPIN [i.35], which are all compatible with RDF and OWL.

Compared with ontologies, semantic rules can be exchanged locally and temporarily, so using semantic rules can facilitate defining dynamic or private relationships. For example, an M2M application can exchange one semantic rule with the semantic engine in M2M platform, and the semantic engine will use this semantic rule only for processing the request from that M2M application. It is very flexible. In addition, the semantic rules can also be used to associate the concepts in multiple different ontologies.

oneM2M has to provide a solution for the following issues:

- Should semantic rule be supported in oneM2M systems?
  - Maybe "YES" since it is more flexible than only using ontologies, e.g. applications can dynamically define new relationships according to their own requirements via using semantic rules, which do not need to revise ontologies.
- Should reasoning function of semantic engine support the reasoning simultaneously with semantic rules and ontologies?
- How do semantic rules be exchanged in oneM2M systems?

7.4 Specific potential requirements for semantics

7.4.1 Semantics Related Requirements

7.4.1.1 Semantic Annotation Requirements

1) The M2M System shall support semantic annotation of application related data (e.g. containers) that are handled (e.g. transferred) by the M2M System based on related ontologies.

7.4.1.2 Ontology Requirements for Semantics Support

1) The M2M System shall be described by an ontology containing the entities of the M2M System, their information models and their relationships.

2) The M2M System shall provide support for linking of the ontology describing the oneM2M System with ontologies describing other information models.

3) The M2M System will be able to support extensible ontologies with new domain concepts, in order to support newly created oneM2M Applications.

4) The M2M System shall be able to support the ontologies that contain entities that are not represented by resources of the M2M System.

5) The M2M System shall be able to support common ontologies (e.g. location, time ontologies, etc.) which are used commonly in M2M Applications. (Source: clause 7.3.2.)

6) Requirements on integration of ontologies (Source: clause 7.3.2):

   - The M2M System shall be able to support simultaneous usage of multiple semantic ontologies for the same M2M resource.

   - The M2M System may support creation, usage, and publication of new ontologies based on semantic reasoning.

7) The M2M System shall be able to import (or reference) ontologies defined outside of oneM2M. (Source: clauses 7.3.1. and 7.3.2.)

8) The M2M System should be able to adapt to changes of ontologies defined within or outside the M2M System, preferably without human interaction. (Source: clause 7.3.3.)

7.4.1.3 Semantic reasoning requirements

1) The M2M System shall be able to support ontology reasoning based on semantically annotated data. (Source: clause 7.3.6.)

2) The M2M System shall be able to support semantic rule based semantic reasoning. (Source: clause 7.3.7.)

7.4.1.4 Semantic mash-up requirements

1) The M2M System shall provide the capability of mashing up virtual device resources based on existing device resources using semantic description and reasoning.

2) The M2M System shall provide the capability to embed semantic mash-up processing into M2M System to enable data analytics (e.g. calculation).

3) The M2M System shall provide the capability of interpreting and applying service logic (e.g. rules/policies of triggering operations upon other resources or attributes according to the change of the monitoring resource) described with semantic annotation and ontology.

4) The M2M System shall be able to aggregate semantic data from more than one resources in order to create a new service. (Source: clause 7.2.1.3.)

5) The M2M System may support creation of virtual devices for new services by obtaining semantic information. (Source: clause 7.1.2.5.)

8 Support for Abstraction and Semantics in oneM2M

8.1 Summary of Requirements

Table 8.1-1 lists the requirements on Abstraction and Semantics. It is based on those identified in oneM2M TS-0002 [i.37]. As the result of the work described in this Technical Report, the original requirements have been refined, new requirements have been added and the overall consistency has been checked. The requirements have been categorized according to key semantic aspects identified in clause 7.1, i.e. semantics annotation, ontology, semantics query, semantics mashup, data analytics and semantic reasoning.

Table 8.1-1

No.	Category	Requirement
1	Semantics Annotation	The M2M System shall provide capabilities to manage semantic information about the oneM2M resources, e.g. create, retrieve, update, delete, associate/link.
2	Ontology	The M2M System shall support modelling semantic descriptions of Things (including relationships among them) by using ontologies.
3	Semantics Annotation	The M2M System shall support a common language for semantic description, e.g. RDF.
4	Ontology	The M2M System shall support a common modeling language for ontologies (e.g. OWL).
5	Ontology	The M2M System should be able to provide translation capabilities from different modeling languages for ontologies to the language adopted by oneM2M if the expressiveness of the imported ontology allows.
6	Semantics Query	The M2M System shall provide capabilities to discover M2M Resources based on semantic descriptions.
7	Ontology	The M2M System shall provide the capability to retrieve semantic descriptions and ontologies stored outside of the M2M System.
8	Data Analytics	The M2M System shall be able to support capabilities (e.g. processing function) for performing M2M data analytics based on semantic descriptions from M2M Applications and /or from the M2M System.
9	Semantics Mashup	The M2M system shall provide the capability to host processing functions for mash-up.
10	Semantics Mashup	The M2M system shall enable Applications to provide processing functions for mash-up.
11	Semantics Mashup	The M2M system itself may provide pre-provisioned or dynamically created processing functions for mash-up.
12	Semantics Mashup	The M2M system shall be able to create and execute mash-ups based on processing functions.
13	Semantics Mashup	The M2M system shall be able to expose mash-ups as resources e.g. virtual devices.
14	Semantics Annotation	The M2M System shall support semantic annotation of oneM2M resources for example application related data contained in containers.
15	Semantics Annotation	The M2M System shall support semantic annotation based on related ontologies.
16	Ontology	The M2M System shall provide support for linking ontologies defined in the context of the M2M system with ontologies defined outside this context.
17	Ontology	The M2M System shall be able to support extending ontologies in the M2M system.
18	Ontology	The M2M System shall be able to use ontologies that contain concepts representing aspects (e.g. a room) that are not represented by resources of the M2M System.
19	Ontology	The M2M System shall be able to re-use common ontologies (e.g., location, time ontologies, etc.) which are commonly used in M2M Applications.
20	Ontology	The M2M System shall be able to support simultaneous usage of multiple ontologies for the same M2M resource.
21	Semantics Reasoning	The M2M system shall be able to update ontologies as a result of the ontology reasoning.
22	Ontology	The M2M system shall provide the capability for making ontology available in the M2M System, e.g. through announcement.

No.	Category	Requirement
23	Ontology	The M2M system shall be able to support mechanisms to import external ontologies into the M2M system.
24	Ontology	The M2M System shall be able to support update of ontologies.
25	Semantics Reasoning	The M2M System shall be able to support semantic reasoning e.g. ontology reasoning or semantic rule-based reasoning.
26	Semantics Reasoning	The M2M System shall be able to support adding and updating semantic information based on semantic reasoning.
27	Data Analytics	The M2M System shall provide the capability of interpreting and applying service logic (e.g. rules/policies of triggering operations upon other resources or attributes according to the change of the monitored resource) described with semantic annotation and ontology.
28	Data Analytics	The M2M system shall support a standardized format for the rules/policies used to define service logic.
29	Ontology	The M2M System shall enable functions for data conversion based on ontologies.
30	Semantics Annotation	The M2M System shall provide the capability for making semantic descriptions available in the M2M System, e.g. announcement.
31	Semantics Annotation	The M2M system shall enable applications to retrieve an ontology representation related to semantic information used in the M2M system.
32	Ontology	The M2M system shall be able to model devices based on ontologies which may be available outside the M2M system (e.g. HGI device template).
33	Ontology	The M2M System shall support storage, management and discovery of ontologies.

Figure 8.1-1 shows dependencies and relations between the requirements listed in table 8.1-1. It can be used as a basis for prioritizing the addressing of requirements and to make sure that all preconditions are addressed first.

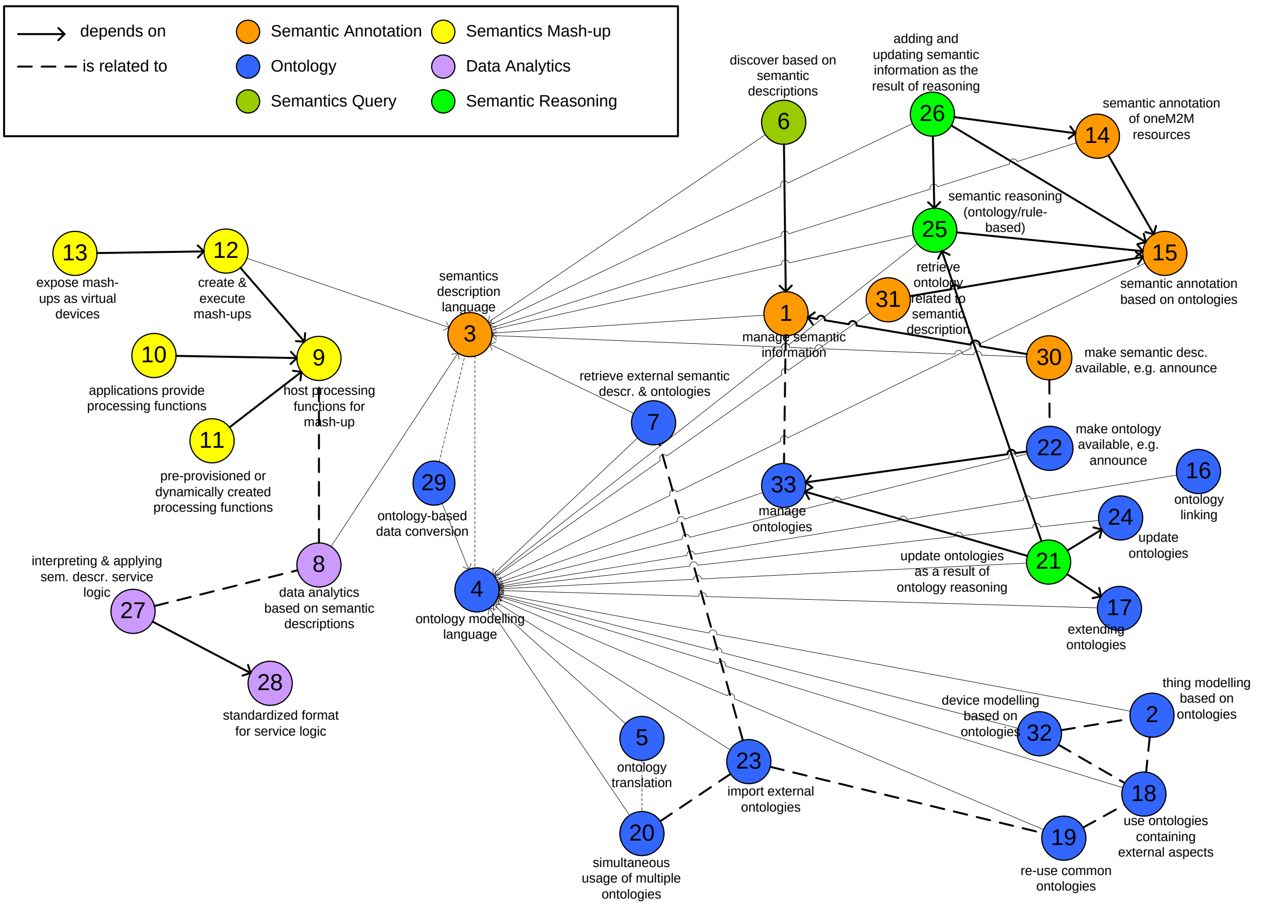


Figure 8.1-1: Dependencies and relations between requirements

8.2 General Modeling Aspects

8.2.1 Overview of modelling aspects

In Release 1, the oneM2M System enables applications to interact with each other using opaque, application specific containers, based on the resource concept. In order to enable interaction between applications that use similar data (e.g. temperature) but employ different encoding for these data a common abstraction layer can be built that provides services to convert the data into a abstracted, common format. With such an abstraction layer, regarding communication, application developers do not have to consider the underlying heterogeneity of their communication partners as this is abstracted away. However, applications in most cases still need to know a priori their communication partners as the Rel-1 discovery functionality is very limited and does not allow discovering other applications based on their semantics, e.g. the services they support. Data is treated as a black box without providing information about its structure and semantics. Thus this information needs to be known in advance by the respective applications and cannot be discovered. For a more flexible system that allows the reuse of information and functionality by different applications the structure and semantics of application data exchanged in the oneM2M System can be made explicit. Providing common abstractions not just for the purpose of interaction, but also for sharing information about the provided functionality is one aspect of it.

M2M allows applications to extend their reach into the physical world. Devices enable the sensing of, as well as the actuation on relevant aspects of the physical world. The physical world is populated by Things in a broad sense, including buildings, rooms, windows, roads, bottles etc. Ultimately, the applications interact with these Things, using the Devices for mediating this interaction. Figure 8.2-1 shows the underlying domain model.

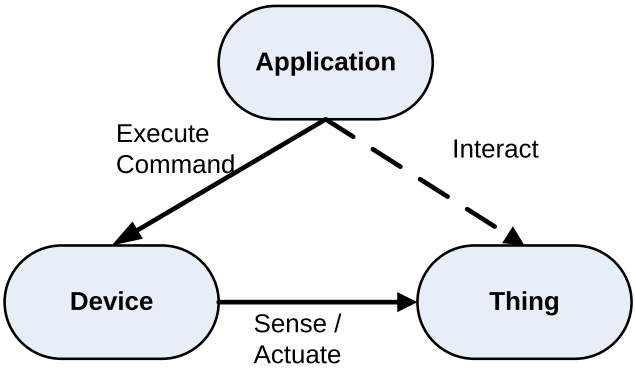


Figure 8.2-1: Domain Model

Whereas the oneM2M Services are directly involved in the communication between application and Device, they nevertheless should model and provide support for real-world aspects as ultimately M2M is about supporting a mediated interaction between the applications and Things.

In the following subclauses there will be further discussion on how Devices and Things can be modelled in an appropriate way, how they can be represented and how the respective models can be used in future releases of the oneM2M System.

8.2.2 Modelling of Devices and Things

8.2.2.1 Modelling of Concepts

In this clause, some more details for the identified key concepts Device and Thing are introduced. These details are emphasized as they are needed for enabling functionalities that are proposed for being supported by the oneM2M System. First, only the most basic structure for modelling Devices and Things is described.

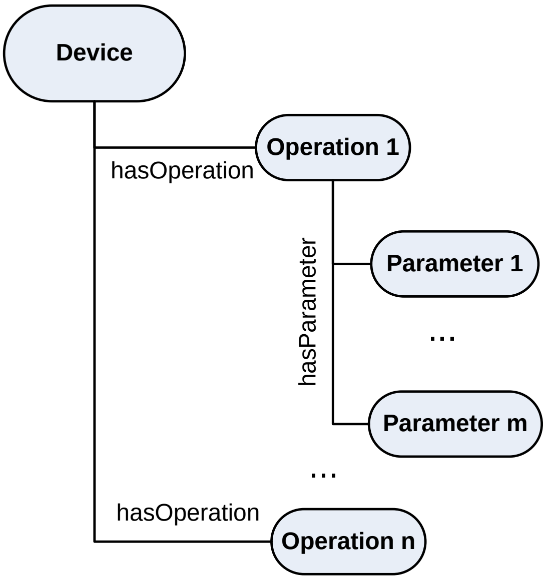


Figure 8.2.2.1-1: Device Structure

Figure 8.2.2.1-18.2.2.1-1 shows some key information about describing Devices. Devices support different Operations, Operations have different Parameters that can be used for input or output purposes. The structure is the basis for defining specific Device Types and successively Operation and Parameter Types. The specific Device Types together with the Operations they support etc. can be defined by other, e.g. application area-specific, standardization organizations. For oneM2M, it is important that they follow the structure as the structure can then serve as the basis for creating the resource representation for Device Instances of the given Device Type.

To support other aspects of Devices more relations and concepts have to be added. For example, the identifier(s), the producer, the energy consumption may be needed for other purposes, e.g. discovery. To keep the complexity of the presentation of the modelling *approach* simple, focus is on the Operations and Parameters that are needed for defining the oneM2M resource structure.

It should be noted that the above structure for describing Devices in terms of their Operations can be used for any type of Device, whether oneM2M Device or other Device that is interworked with the oneM2M System. This structure does, however, not yet take into account more detailed aspects of modelling Devices, e.g. direction of information flow etc.

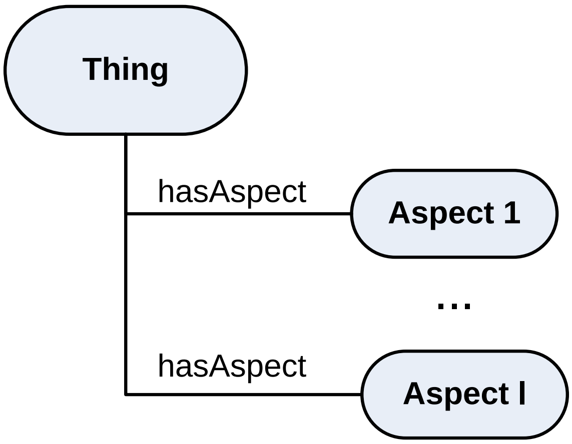


Figure 8.2.2.1-2: Thing Structure

Figure 8.2.2.1-2 shows some key information about Things. Basically Things have different Aspects, e.g. a room can have a size, a height, a temperature, a noise level etc. These Aspects can be related to Devices that provide information, e.g. the current noise level, or they can influence it like the current temperature in case the Device is a heater or air conditioning system. Aspects can also model relations to other Things.

As in the case of Devices, the specific Thing Types together with the Aspects they have etc. can be defined by other, e.g. application area-specific, standardization organizations. For oneM2M, the underlying structure is important.

8.2.2.2 Modelling of Types

Based on the identified concepts, which define the structure, the respective types can be defined. This means, based on the previous subclause, it is known that, for example, Devices have Operations and Operations have Parameters, but what types of Devices exist and what Operations each of these types have is not yet known.

As shown in figure 8.2.2.2-1, Device Types can be hierarchically defined. There is a common Device Super Type, and, in this example, a single Operation is defined for it, i.e. MyOperation 1. In the given example, Device Type 1 and Device Type 2 inherit from the Device Super Type, i.e. instances of these also support MyOperation 1 (shown for Device Type 1). Device Type 1 in addition introduces MyOperation 2, which is supported by all Devices of Device Type 1 and all Device Types inheriting from it. Note that Operations could be mandatory or optional.

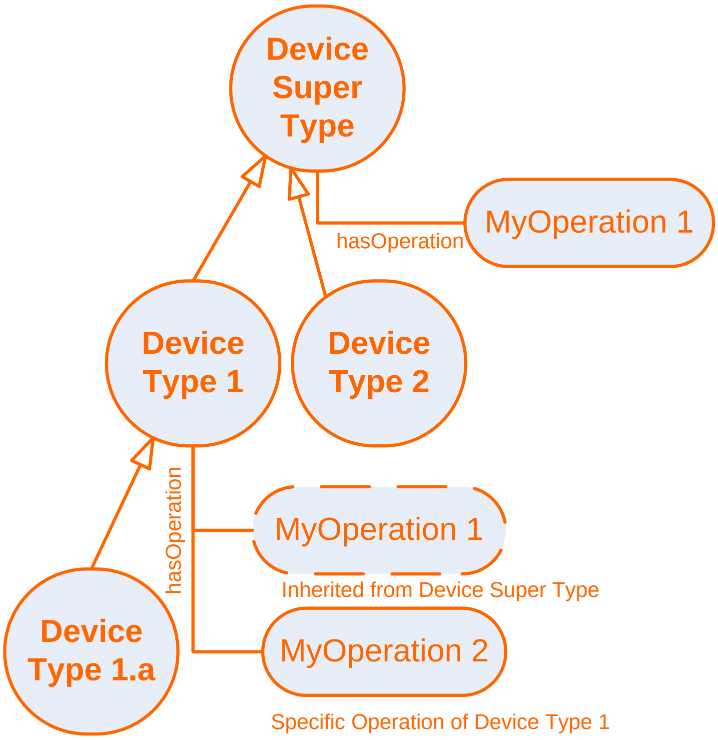


Figure 8.2.2.1-1: Example of Device Types and their Operations

The Operations again have a structure, which is defined in an Operation Type hierarchy as shown in figure 8.2.2.1-2. So, for example, My Operation 1 could be of Operation Type 2 and My Operation 2 could be of Operation Type 1. Operation Types define what Parameters they have, which again would have Parameter types. Parameter Types (sketched with dashed lines) follow exactly the same approach as Device Types and Operation Types and can build type hierarchies. Parameter types include the specification of the data type of the parameter.

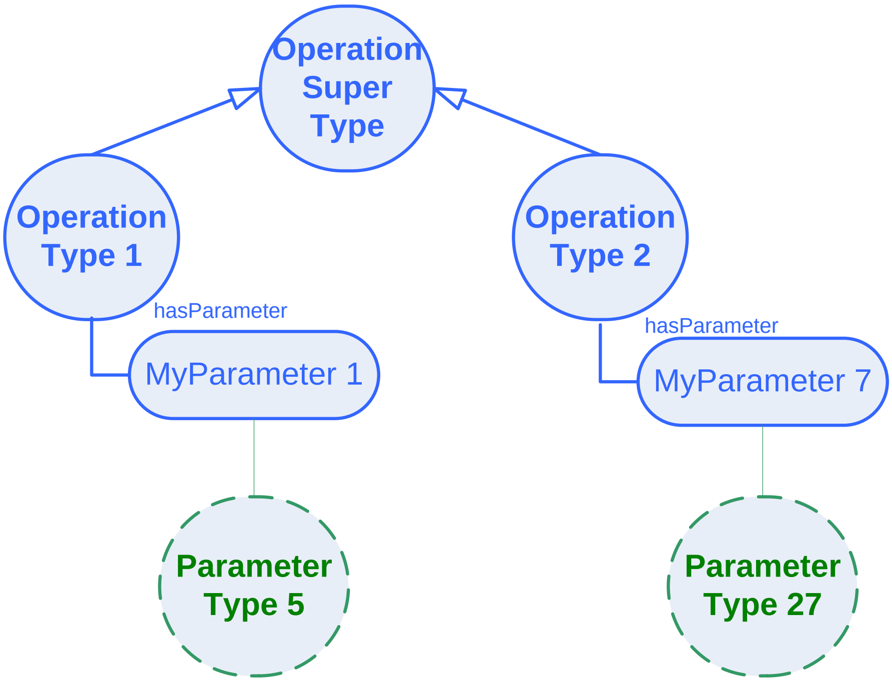


Figure 8.2.2.1-2: Example of Operation Types and their Parameters

Figure 8.2.2.1-3 sketches an example of a Thing Type hierarchy. Some of the Aspects of a given Thing Type will be dynamic like Indoor Temperature, Speed or Occupancy and thus would be associated to Devices sensing/setting these values, others would be static like Area, Volume or Number of Floors. These could nevertheless be relevant for discovery purposes. Finally, Aspects can describe the relation to other Things, e.g. the Part of Building Aspect.

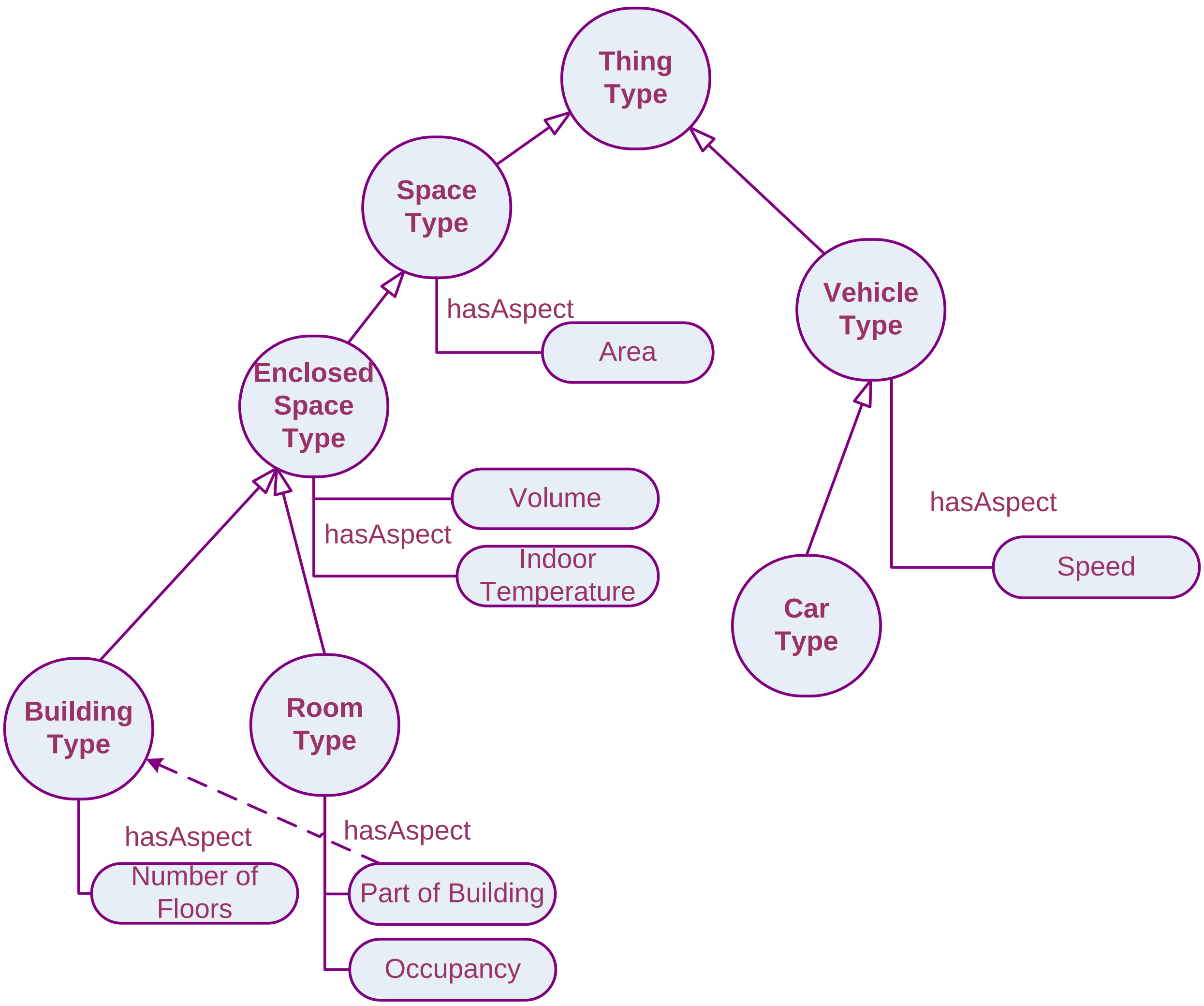


Figure 8.2.2.1-3: Example of Thing Type Hierarchy

Type hierarchies can be defined by different standardization organizations (SDOs). oneM2M can make use of these, if they follow the respective structure as defined through the concepts and their relations. In the example of Device Types, a type hierarchy of Device Types is defined where the supported Operations for each Device Type are identified. It is also possible that a common basic Device Type hierarchy is defined (e.g. by oneM2M itself) that is then extended by different SDOs. Based on these, proprietary extensions are possible, preferably by inheriting from suitable super type(s).

Given that type hierarchies are provided, the oneM2M System can make use of them, supporting multiple of them in parallel - only when it comes to supporting equivalence of types from different type hierarchies or from different extensions of type hierarchies additional work is required.

8.2.2.3 Modelling of Instances

Given the structure provided by the concepts, instances can be modelled based on the types. Figure 8.2.2.3-1 shows the structure of the Device Instance "My Device 234", which is of type Device Type 1. For that reason, it has both Operations "MyOperation 1" (inherited from Device Super Type) and "MyOperation 2", which are of Operation Type 2 and Operation Type 1 respectively.

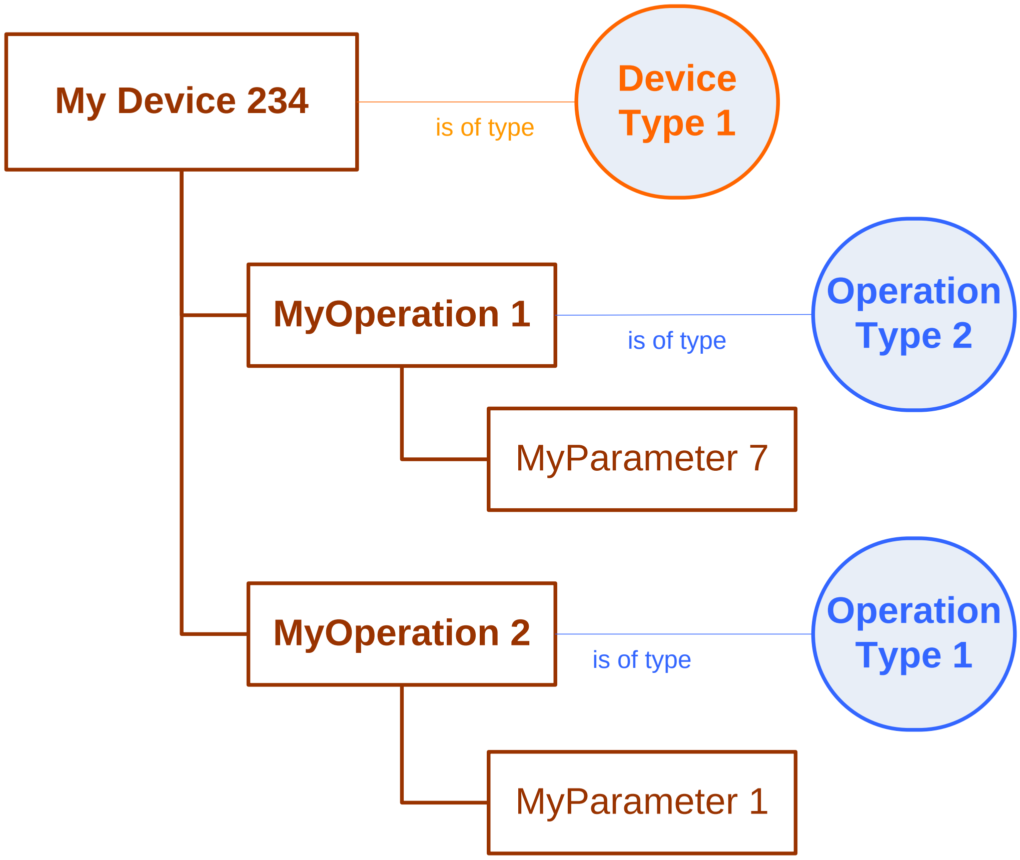


Figure 8.2.2.3-1: Example of a Device Instance based on the type modelling

The modelled Device Instance can directly be mapped to the oneM2M resource structure for the Device. The type information is also highly relevant for discovering suitable instances such as Devices and Things.

8.2.2.4 Combined view of approach

After the step-by-step introduction of concepts, types and instances as the three parts of this modelling approach, this clause provides the combined view.

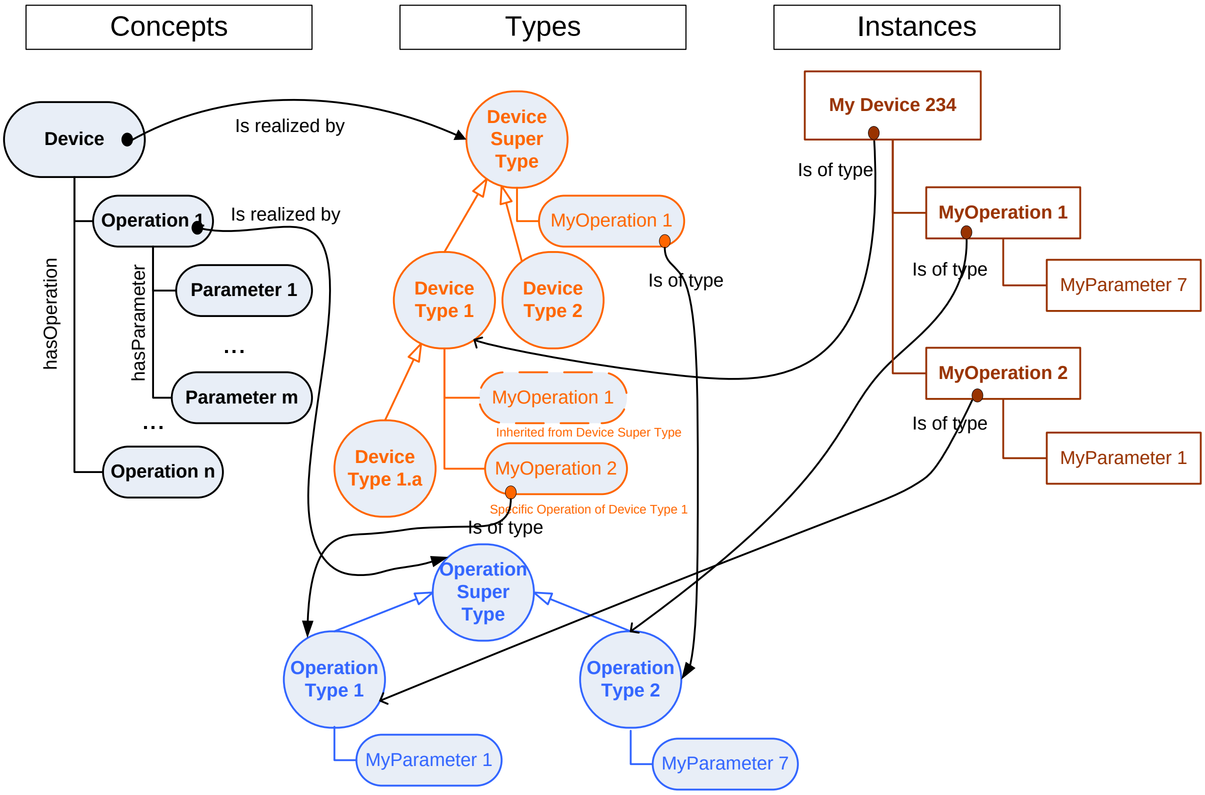


Figure 8.2.2.4-1: Overview and relation of concepts, types and instances for the example of Devices

Figure 8.2.2.4-1 shows the concepts and structure of Devices, which are - for the purpose of introducing the approach - limited to Operations and Parameters. These concepts and structure of Devices would be defined by oneM2M and provide the basis for implementing related functionalities.

Device Types introduce the specific types of Devices that are to be supported by the system together with their specific Operations. They follow the structure given by the concepts of oneM2M. Device Types can be organized in a type hierarchy. Operations again are of a certain type, which are defined in a separate type hierarchy. Operation Types introduce the specific Parameters required, which again have a type (not shown). The Device Types, Operation Types, Parameter Types etc. can be defined by other, possibly application-area specific standardization organization or be proprietary. The type information is of course relevant for the Operation of a semantically enhanced oneM2M System, but it is not statically implemented into the oneM2M system functionalities, but can rather be seen as configuration information.

Finally, the Device Instances model concrete Devices in a oneM2M deployment. They can be directly represented as Resources in the oneM2M System. Their general structure corresponds to the one defined by the concepts, e.g. that Devices have Operations. The specific Operations of a Device are defined by their respective Device Type. Linking the Device Instance to the Device Type may be relevant for oneM2M system functionalities, e.g. discovery and abstraction.

8.2.2.5 Modelling of Associations

As already shown in clause 8.2.2.2, some Aspects of Things concern dynamic values that can be measured or changed by Devices, e.g. the speed of a vehicle can be measured and the indoor temperature of a room may be changed by a heating or air conditioning system.

To capture this aspect in a system, the relation between Things and Devices can be modelled in form of Associations. The Association should describe which Aspect is associated to which Operation of a Device and what the Operation can do with respect to the Aspect, i.e. provide the information (measure, observe, etc.) or change (set, increase, decrease, etc.).

As shown in figure 8.2.2.5-1 there are two main types of associations MeasureAssociation and SetAssociation. A MeasureAssociation represents the case in which the Device Operation measures the Thing Aspect, whereas a SetAssociation represents the case, where a device operation can be used to set the Thing Aspect, i.e. triggers an actuation in the real world so that the state of the real world reflects the target set for the Thing Aspect. Further subtypes of associations can be defined.

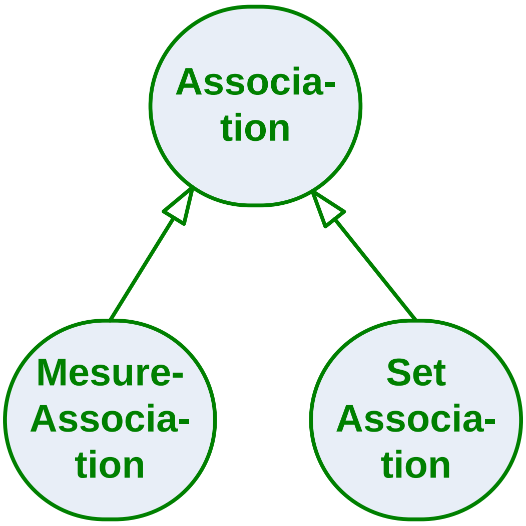


Figure 8.2.2.5-1:Types of Association

Figure 8.2.2.5-2visualizes two examples of associations.

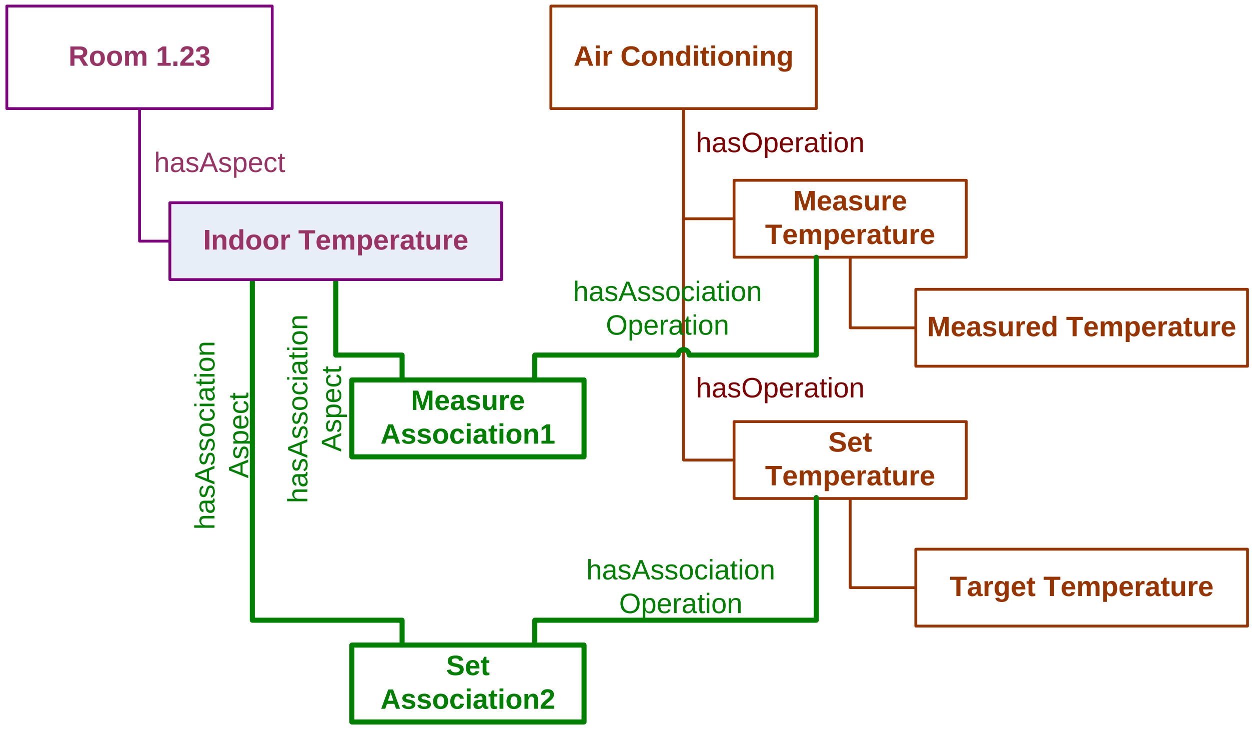


Figure 8.2.2.5-2: Association Example

Even though an association logically represents a relation between a Thing aspect and a device operation, it is modelled as a concept here. The reason is that there may be other information that is related to the association., Such information would otherwise be difficult to represent as it cannot be associated with either the Thing Aspect or the Device Operation.

NOTE: This approach is called reification and is typically applied in such situations.

An example for such information is quality information that is related to the Association, e.g. a measured temperature may have some accuracy that is related to the sensor and thus could be directly associated with the sensor value. However, how well this information reflects the temperature of the room and to what degree it can be trusted is a quality that is neither a property of only the room aspect nor a property of only the measured value, but a property of the relation between the two. Representing this relation as a concept allows it to have the quality aspects as properties.

8.2.2.6 Modelling of mash-up devices

The basic structure for modelling Devices is described in figure 8.2.2.1-1of clause 8.2.2.1. In that structure, specific Device Types and successively Operation and Parameter Types are involved. The structure for modelling mash-up device (i.e. the virtual device created by mash-up) should inherit the basic structure for modelling devices, but may extend the basic structure for describing the mash-up relationship between mash-up devices and other devices involved in mash-up. A concept named Mash-up is introduced for representing the mash-up relationship.

As shown in figure 8.2.2.6-1, the Operation of mash-up device will be linked to the Mash-up concept. Besides Operation of mash-up device, the Mash-up concept is also associated with Operations of other devices involved in mash-up and a Mash-up logic concept.

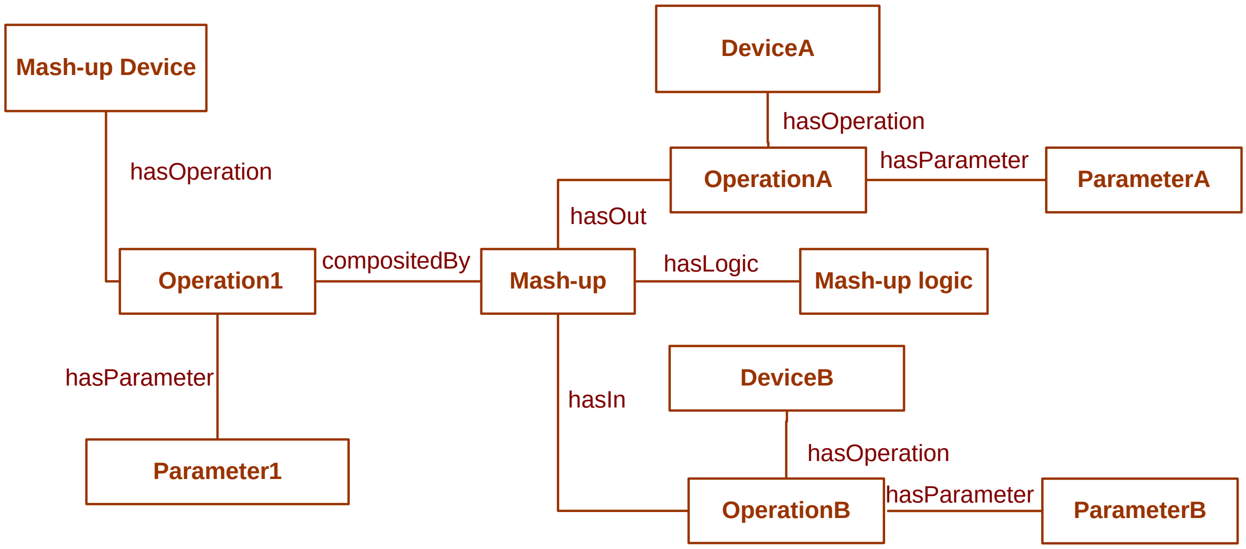


Figure 8.2.2.6-1: Example structure for modelling mash-up devices

There are two types of relations between the Mash-up concept and Operations of other devices involved in mash-up. One type indicates that the Operation plays an input role in the mash-up, i.e. the Parameter of the Operation will be an input parameter in the mash-up. The other type indicates that the Operation plays an output role in the mash-up, i.e. the Parameter of the Operation will be an output parameter in the mash-up.

The Mash-up logic concept is used to describe the service logic used in the mash-up. It will have a data type property (e.g. xsd: string) for logic expression to indicate the mapping relationship among the Parameters of Operations of other devices involved in the mash-up and the Parameters of Operation of Mash-up Devices.

Based on the logic expression of Mash-up logic, the input parameters of Operation of Mash-up Device will be mapped to set the input Parameters of Operation involved in mash-up, and the output Parameters of Operation involved in mash-up can be mapped to the set of output parameters of Operation of Mash-up Device. The Parameters of Operation that plays input role in mash-up can also be mapped to set the parameters of Operation that plays output role in mash-up.

8.2.3 Device and Device Template Modelling Using Ontologies

In this clause an approach of how device types and device instances can be modelled in a homogeneous way using ontologies is shown. The focus is on explaining the approach, not on completeness of the device types and device instance aspects being modelled. Due to their relevance for creating resource structures for representing devices in the oneM2M platform, the focus is on modelling input-output operations, but also the modeling of some manufacturer-specific aspects like manufacturere name and product identifier is shown.

The modelling follows a two-layer approach. The upper layer is for modelling device types, the lower layer for modelling device instances. The upper layer provides a given device type template that is modelled as classes and properties of an ontology. The device types are then modelled as instances of the classes and properties of this ontology. In the lower layer, the same device types that were modelled as instances of the template classes in the upper layer are now interpreted as the classes of the device ontologies. The actual individual devices can then be modelled by creating instances of these classes. The dual character of instance and class can, for example, be modelled in the OWL Full variant of the Web Ontology Language (OWL) [i.29].In the following an example of how a device type can be modelled based on a device type template and an individual device based on a device type is shown.

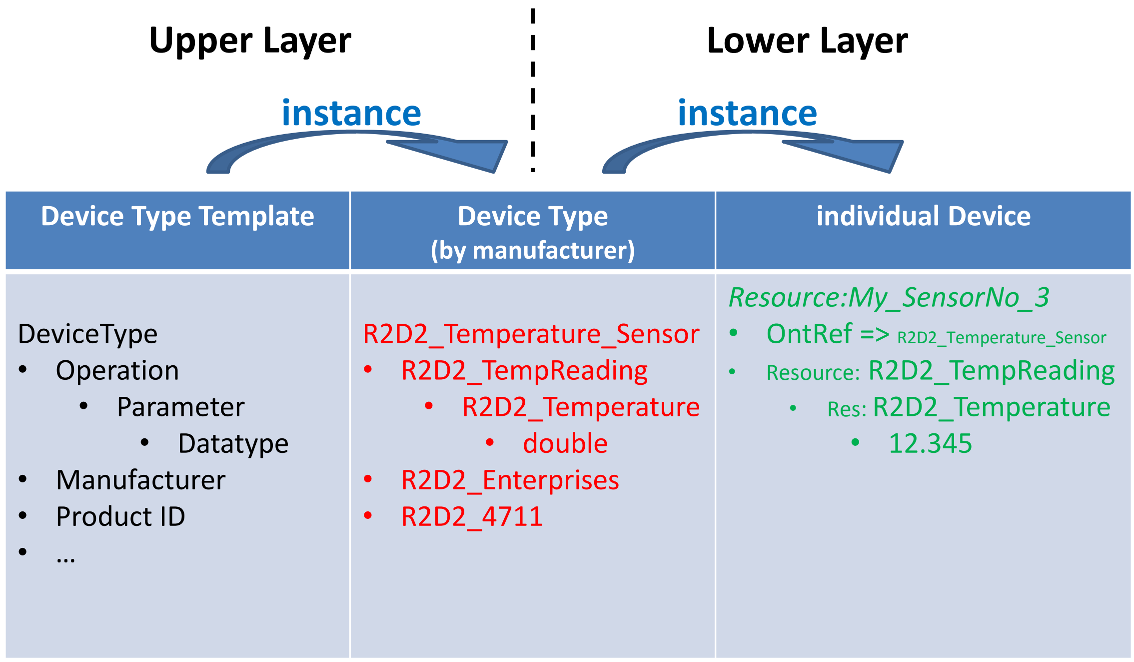


Figure 8.2.3-1: Modelling of device type templates, device types and individual devices

Figure 8.2.3-1 shows the three parts that make up the upper and lower layer, each consisting of classes and the corresponding instances. There are three and not four parts due to the dual character of the device type, which makes up an instance in the upper layer, but provides the class structure for the lower layer.

The device type template provides the structure that models a type of device, e.g. a manufacturer producing a temperature sensor can "fill out" the template by creating an ontology instance. In the case shown in figure 8.2.3-1 R2D2_Enterprises manufactures an R2D2_Temperature_Sensor with the Product ID R2D2_4711. It has an operation called R2D2_TempReading, which in turn has a parameter R2D2_Temperature of type double. This information in turn serves as the structure for describing an individual temperature sensor of the type R2D2_Temperature_Sensor. It provides the basis for creating the resource structure representing the individual device in oneM2M, which is depicted on the right side of figure 8.2.3-1. The value of the R2D2_Temperature parameter, which refers to the temperature measured, is given as 12.345 represented as a double.

The example has been modelled as a set of OWL Full ontologies, using imports for including required classes and properties.

Tables 8.2.3-1 and 8.2.3-2, describe the ontology classes (table 8.2.3-1) and properties (table 8.2.3-2) for describing the device template in more detail. Where appropriate, subclasses can be used to model specializations of common classes. In this case, this is done to distinguish different types of operations.

Table 8.2.3-1: Ontology Classes for Device Template

Class => SubClass	Explanation
DeviceType	Manufacturer defined name/ID for a class of alike devices (= type) that are e.g. described in a product description
Operation	Identifies an operation of the device
=> OutputOperation	The operation produces only an output message. The device does not expect correlated input/ack
=> InputOperation	The operation consists of an input message only. The device does not produce correlated input/ack
=> In-OutOperation	The operation receives an input message and produces a correlated output/ack
Parameter	Identifies a parameter of the operation
=> InputParameter	Indentifies a parameter related to the input of the operation
=> OutputParameter	Indentifies a parameter related to the output of the operation
DataType	Identifies the datatype of the parameter (e.g. xsd: double)
Manufacturer	Name/ID of the manufacturer
Product ID	Manufacturer defined handle/ID to identify the type of the device, e.g. Type/Model-number.

The class of Parameter is divided into the subclass of InputParameter and OutputParameter. InputParameter is related to the input of the operation, and OutputParameter is related to the output of the operation.

Table 8.2.3-2: Object and Datatype Properties for Device Template

Domain	Property	Range	
DeviceType	hasOperation	Operation	Object Properties
Operation	hasParameter	Parameter	
Parameter	hasParameterType	Datatype	
DeviceType	*hasManufacturer*	*xsd:string*	Datatype Properties
DeviceType	*hasProductID*	*xsd:string*	

Tables 8.2.3-4and 8.2.3-5, describe the ontology classes (table 8.2.3-4) and properties (table 8.2.3-5) for describing individual device instances of the R2D2_Temperature_Sensor. A hierarchy of superclasses may be defined, e.g. Temperature_Sensor and Device superclasses could be introduced, which could be useful for discovering different types of temperature sensors from different manufacturers. Also, an Abstract Temperature Sensor providing a standardized interface to applications could be introduced, either on the same level as the R2D2_Temperature_Sensor, i.e. as a subclass of TemperatureSensor, or in a separate hiereachy. In the latter case the link to the concrete temperature sensors has to be explicitly modelled.

Table 8.2.3-4: Ontology Classes for R2D2_Temperature_Sensor

Class => SubClass	Explanation
Device => Temperature_Sensor =>R2D2_Temperature_Sensor	User defined name/ID for a specific instance of the R2D2_Temperature_Sensor instance., e.g. My_SensorNo_3
R2D2_TempReading	Specific operation of the instance of R2D2_Temperature_Sensor
R2D2_Temperature	Specific parameter of the instance of R2D2_TempReading
Metadata	Metadata related to the value of an R2D2_Temperature

Table 8.2.3-5: Object and Datatype Properties for R2D2_Temperature_Sensor

Domain	Property	Range	
R2D2_Temperature_Sensor	hasTemperatureOperation	R2D2_TempReading	Object Properties
R2D2_TempReading	hasTemperatureParameter	R2D2_Temperature	
R2D2_Temperature	hasMetadata	Metadata	
R2D2_Temperature	*hasValue*	*xsd:double*	Datatype Properties

The listing below gives an example of how an individual device instance of the R2D2_Temperature_Sensor is modelled in OWL, using the more concise and readable Turtle representation [i.31], instead of the more commonly used RDF/XML notation [i.32].

Device Instance (in OWL/Turtle)

```
@prefix : <http://InstanceOntology#> .

@prefix owl: <http://www.w3.org/2002/07/owl#> .

@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

@prefix dev: <http://DeviceOntology#> .

@prefix dev-temp: <http://DeviceTemplateOntology#> .

@base <http://InstanceOntology> .

<http://InstanceOntology>      rdf:type        owl:Ontology;
                               owl:imports     <http://DeviceOntology>

:Resource:My_SensorNo_3        rdf:type        dev-temp:R2D2_Temperature_Sensor;
                               owl:NamedIndividual;
                                               dev:hasTemperatureOperation:R2D2_TempReading.

:R2D2_TempReading              rdf:type        dev-temp:R2D2_TempReading;
                               owl:NamedIndividual;
                                               dev:hasTemperatureParameter:R2D2_Temperature.

:R2D2_Temperature              rdf:type        dev-temp:R2D2_Temperature;
                               owl:NamedIndividual;
                                               dev:hasValue"23.45"^^xsd:double.
```

8.3 Approaches to various levels of semantic support

8.3.1 Introduction

The clauses 8.3.2, 8.3.3 and 8.3.4 sketch three examples of varying complexities representing semantic support in oneM2M in future releases. The idea is to show the spectrum of architectural options that oneM2M could support and thus serve as a basis for the discussion on how to proceed regarding semantic support in oneM2M. Other configurations can be envisioned for future release planning.It should be taken into account that these are only sketches highlighting core aspects and are not to be interpreted as being complete in any way.

8.3.2 Semantic Annotations

In this architectural option, the semantic support is limited to having semantic annotations within the oneM2M platform. For Release 1 of the oneM2M speficiation, an ontologyRef attribute for <instance> resources, <container> resources and <application> resources is foreseen. The ontologyRef is a URI that identifies the ontology that is used for representing the respective information.

Applications can read this attribute, identify the semantics of the information and use the URI to retrieve additional information, e.g. interpreting the URI as a URL and fetch additional information from there or use the URI as an identifier for looking up additional information in a semantic database. The additional information may relate to the semantic type, the structure of the data and relations to other information.

Figure 8.3.2-1 visualizes how semantic annotations are used. Ontology references point to an ontology that is part of some kind of semantics infrastructure. An application can read the ontology reference and then use it to access more information from the semantics infrastructure.

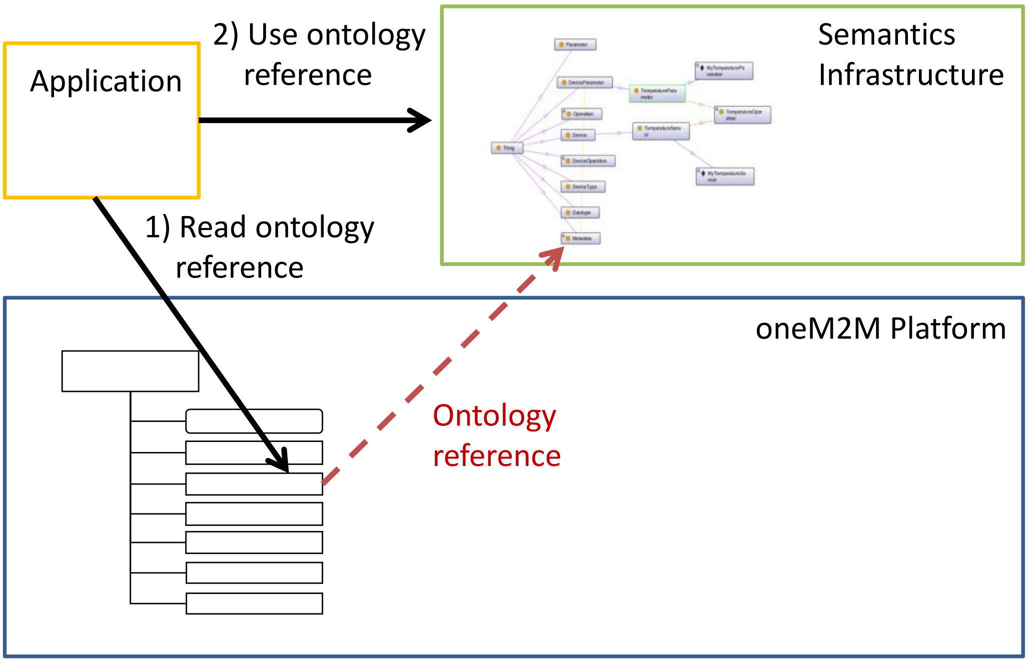


Figure 8.3.2-1: Semantic Annotations

This architectural option does not provide any semantic functionality within the platform itself, but, with the semantic annotations, applications can use semantic functionalities on top of the platform, e.g. for creating index structure or reasoning. Without platform support this may not be very efficient, e.g. if a large amount of information first has to be retrieved from the oneM2M platform and/or the semantic infrastructure to be able to do reasoning.

8.3.3 Use of Semantic Technologies for Platform Functionalities

In this architectural option there are some oneM2M platform functionalities that make use of semantics. This may be by using semantic functionalities and/or semantic modelling. Typically the semantic aspects are then exposed through the interface. The use of semantic aspects does not imply that the complete platform functionality uses semantics. It may be limited to some of the functions exposed through the interface.

In this architectural option, the semantic modelling is typically targeted to explicitly support the specific functionality. This often means that existing ontologies cannot be used out of the box as the oneM2M platform specifics have to be taken into account.

Two examples for semantically enchanced functionalities are the discovery functionality and the use of semantics for modelling device templates as the basis for resource creation.

To enable an expressive discovery functionality, the query could be formulated in a semantic form. This could be an existing query language like SPARQL or something oneM2M-specific. The query results would then point to oneM2M resources.

Semantic modelling can be used for defining the structure of the resources representing a specific device instance. The advantage of such a model is that concepts and relations can be explicitly modelled and this may later be reused for other aspects like the discovery functionality. The semantic model may include generic parts from existing ontologies, but core aspects have to fit the oneM2M resource structure.

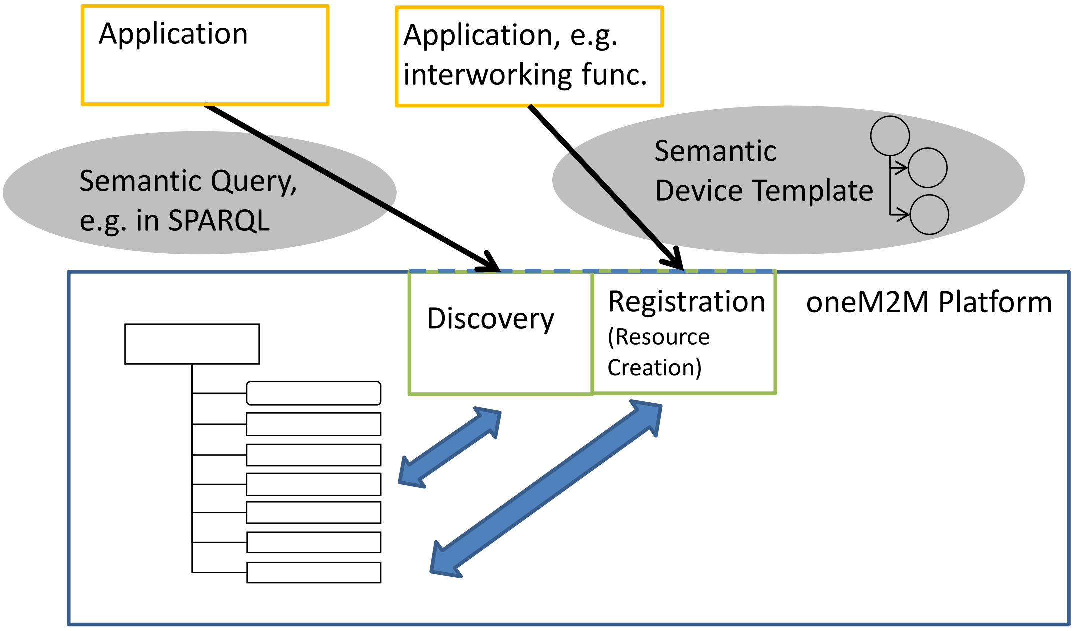


Figure 8.3.3-1: Use of Semantic Technologies for Platform Functionalities

Figure 8.3.3-1 visualizes the platform with two functionalities that have semantic elements. The applications interact with these functionalities using semantically modelled aspects in the interface. In the case of discvoery this may be a semantic query - in the case of the Registration it may be a semantically modelled device template.

8.3.4 Full Semantic Platform

In the full semantic platform architectural option, the whole platform exposes all aspects in semantic form. All information is ontology-based and where possible existing ontologies are used. Instead of having specific oneM2M interfaces common semantic interface and tools are used for interacting with the platform. Such a semantic oneM2M platform could be easily integrated with existing semantic platforms like the Semantic Web.

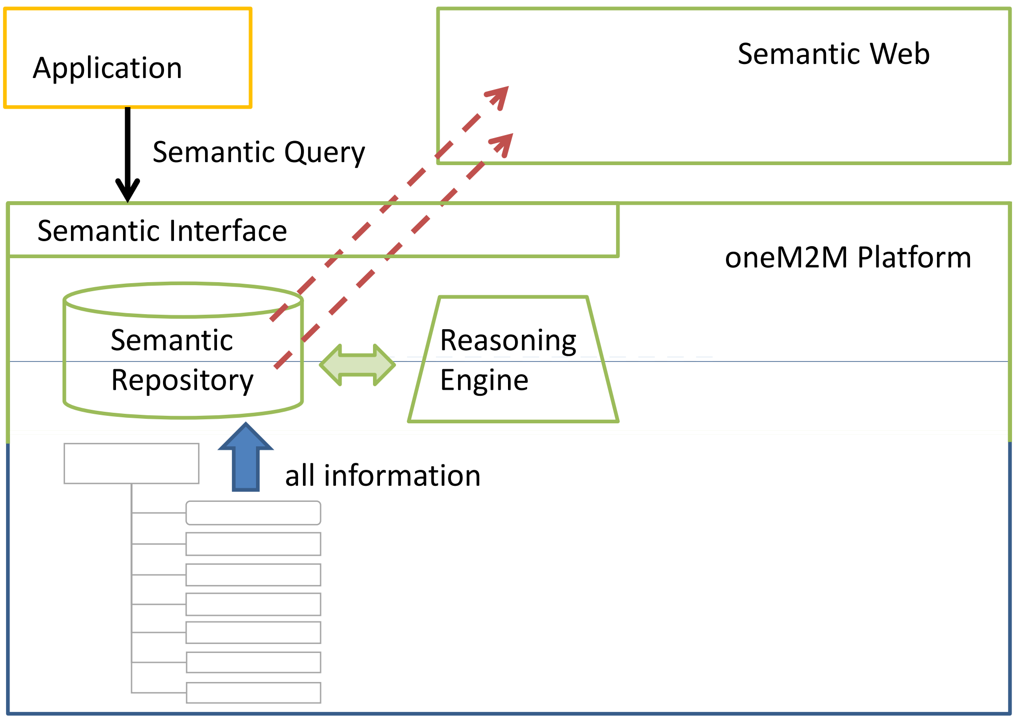


Figure 8.3.4-1: Full Semantic Platform

Figure 8.3.4-1 shows a sketch of the full semantic platform architectural option. Applications interact with the platform through a semantic interface. General semantic functionality like a reasoning engine allows the deriving of additional information. The information can easily be interlinked with the Semantic Web. Existing ontologies can be reused as much as possible.

8.4 Ontology-based Modeling

8.4.1 Support of common languages

RDF(S) and OWL have been introduced in clause 6.1.3 as most common languages for describing ontologies.

The Resource Description Framework (RDF) is a framework for representing information on the Web. It is essentially a data-model using as building block a resource-property-value triple, called a statement.

In the example shown below, an RDF graph is used to depict the statement that Sample1 is measured on "2014-08-21". RDF graphs are powerful for representing information, but abstract and better suited for human analysis, not for application exchange.

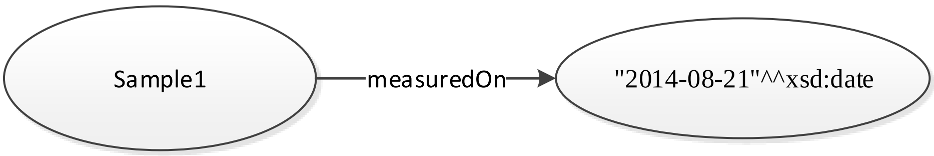


Figure 8.4.1-1: A semantic example in RDF Graph form

RDF statements could also be represented in a concrete format, such as a file or other byte stream. The most popular used expressive formats are RDF/XML and the Terse RDF Triple Language (Turtle). The figures below show the RDF statements in RDF/XML and Turtle respectively.

```
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:xsd ="http://www.w3.org/2001/XLMSchema#"
xmlns:m2m="http://www.mydomain.org/m2m-ns">
<rdf:Description rdf:about="Sample1">
< m2m:measuredOn>"2014-08-21"^^xsd:date"</m2m:measuredOn>
<m2m:unit>"mmHg"</m2m:unit>
</rdf:Description>
</rdf:RDF>
```

Figure 8.4.1-2RDF Statement in RDF/XML

```
@prefix rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#  .
@prefix rdfs: http://www.w3.org/2000/01/rdf-schema#  .
@prefix m2m: http://www.mydomain.org/m2m-ns .


m2m:Sample1
    m2m:measuredOn "2014-08-21"^^xsd:date
    m2m:unit "mmHg"
```

Figure 8.4.1-3: RDF Statement in Turtle

Based on the RDF triples (subject, predicate and object) resources can be structured using a RDFS (RDF Schema) vocabulary. RDF use of a loose set of relations (triples) make it ideal for the integration of possibly heterogenous information on oneM2M platforms.

RDF is domain-independent, so no assumptions about a particular domain of use are made. It is up to the users to define their own terminology in a schema language called RDF Schema (RDFS). RDFS defines the vocabulary used in RDF data models. RDFS can define the vocabulary, specify which properties apply to which kinds of objects and what values they can take, and describe the relationships between objects.

In the example above xmlns:m2m in is the self-defined domain name, in which the property measuredOn is defined as part of the RDFS.

Semantic technologies use a combination of a schema language and an ontology language to provide enhanced capabilities. RDFS vocabulary can be extended by the Web Ontology Language (OWL) with additional resources that can be used to build more expressive ontologies for the web.

OWL uses a predefined, reserved vocabulary to define classes and the relationships between them for specific areas of interest, At the same time, OWL introduces additional restrictions regarding the structure and contents of RDF documents in order to make processing, reasoning more computationally feasible. For example, OWL defines properties that correspond to the standard set operators: intersection, union, and complement to define Boolean combinations of classes.

Figure 8.4.1-4 shows an example of using intersection to define the concept of ConstrainedDevice is the intersection of several clasees: the Device class and three anonymous classes that put restrictions on specific paramaters such as memory, computing power and energy. At first glance, it might appear that the OWL is equivalent to saying that ConstrainedDevicer is rdfs:subClassOf Device as well as of each of the classes setting restrictions on memory, computing power and energy. However, subClassOf statements only state that all constrained devices are devices and have restricted values of memory, computing power and energy. They cannot be used to infer that an entity is a constrained device from only their being devices and their belonging to the three restricted classes , as can be done using owl:intersectionOf.

```
<owl:Class rdf:ID="ConstrainedDevice">
<owl:intersectionOf rdf:parseType="Collection">
<owl:Class rdf:about="#Device"/>
<owl:Restriction>
  <owl: onProperty ref:resource="#availableMemory"/>
  <owl:hasValue rdf:resource="#Amount_Limited_Memory"/>
<owl:Restriction>
<owl:Restriction>
  <owl: onProperty ref:resource="#computingPower"/>
  <owl:hasValue rdf:resource="#Amount_Limited_Computing"/>
<owl:Restriction>
<owl:Restriction>
  <owl: onProperty ref:resource="#availableEnergy"/>
  <owl:hasValue rdf:resource="#Amount_Limited_Energy"/>
<owl:Restriction>
</owl:intersectionOf>
</owl:Class>
```

Figure 8.4.1-4:Example of owl:intersection

OWL uses the RDF and RDFS, XML schema datatypes, and OWL namespaces. The OWL vocabulary itself is defined in the namespace http://www.w3.org/2002/07/owl# and is commonly referred to by the prefix owl.

OWL allows you to use a number of predefined datatypes, most of which are defined in the XML Schema.

RDF based vocabularies such as RDFS or OWL, make it easy to define inference possibilities and use expressive formats such as RDF/XML and Turtle. As such the use of RDF and OWL as common languages within oneM2M is recommended.

8.5 Architectural Aspects

8.5.1 Introduction

This clause presents architectural recommendations for, or potential constraints to, the oneM2M architectural design. This clause also highlights any restrictions that the oneM2M architecture potentially places on utilization of the analysed semantics technologies within oneM2M.

8.5.2 oneM2M architectural design considering semantics

oneM2M architecture for the Common Service Layer in the M2M system specifies a Common Service Entity (CSE) with Common Services Functions (CSFs). The CSFs provide services to the Application Entities (AEs) via the Mca reference point and to other CSEs via the Mcc reference point. CSEs interact with the underlying Network Service Entity (NSE) via the Mcn reference point.

For supporting semantics in oneM2M architecture, core functionalities of semantics (i.e. semantic engine) should reside inside the CSE for interactingthe AEs via the Mca reference point and provide semantic interworking with other CSEs via the Mcc reference point. Then, oneM2M system can handle ontologies and semantic information through interacting with various CSFs in support of semantics engine in the CSE.

Figure 8.5.2-1 illustrate a high-level architectural model for semantics from the existing oneM2M architecture.

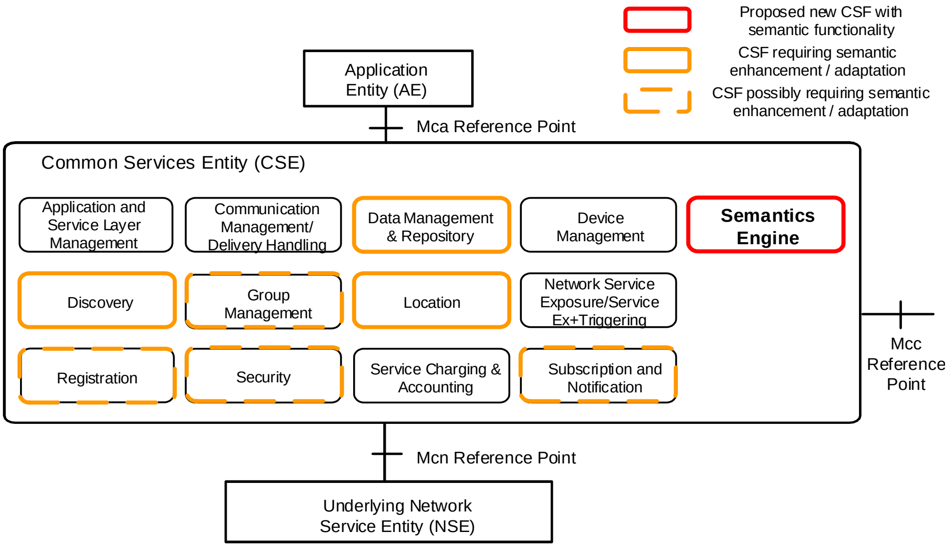


Figure 8.5.2-1: A high-level architectural model for oneM2M semantics

The semantic related CSFs can be logically divided into the following groups:

- Semantic engine: The semantic engine provides core functionalities in the generic functional model of figure 7.1.2.0-1to support semantics.
- Semantic enabled functions: Enhanced functionalities for semantics are required to existing CSFs through a direct link with semantic engine:
  - CSFs requiring semantic enhancement/adaptation: data management & repository, discovery, locatio.
  - CSFs possibly requiring semantic enhancement/adaption: group management, registration, security, subscription and notification.

8.5.3 Semantic Description of Resources

8.5.3.1 Overview

As identified in the requirements, Semantic Annotation within the oneM2M platform is of key importance for supporting semantic functionalities within the oneM2M platform. In the following clauses a semantic descriptor is introduced.

8.5.3.2 Semantic Descriptor

Resources of type <AE>, <container> and <contentInstance> optionally can have one or more semantic descriptor resources. The semantic descriptor resource is further described in clause 8.5.2.3. This is shown in figure 8.5.3.2-1, where <resource> only refers to resources of type <AE>, <container> and <contentInstance>. The case that there are multiple semantic descriptors can be used if the same resource is to be semantically described according to multiple different ontologies.

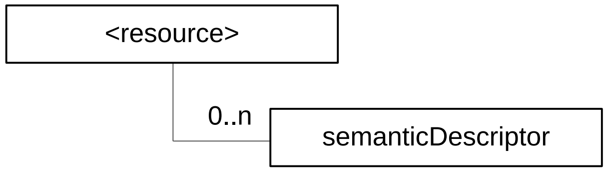


Figure 8.5.3.2-1: semanticDescriptor resource

8.5.3.3 Resource semanticDescriptor

The <*semanticDescriptor*> resource is used to store a semantic description pertaining to a resource and potentially sub-resources. Such a description can be provided according to ontologies. The semantic information is used by the semantic functionalities of the oneM2M system and is also available to applications or CSEs.

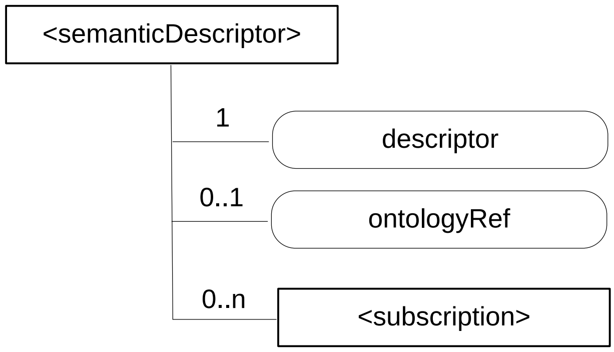


Figure 8.5.3.3-1: Structure of <*semanticDescriptor*> resource

The <*semanticDescriptor*> resource contains the child resources specified in table 8.5.3.3-1.

Table 8.5.3.3-1: Child resources of <*semanticDescriptor*> resource

Child Resources of <*semanticDescriptor*>	Child Resource Type	Multiplicity	Description
[variable]	<*subscription*>	0..n	See [i.39], clause 9.6.8 where the type of this resource is described.

The <*semanticDescriptor*> resource contains the attributes specified in table 8.5.3.3-1.

Table 8.5.3.3-2: Attributes of <*semanticDescriptor*> resource

Attributes of <*semanticDescriptor*>	Multiplicity	RW/ RO/ WO	Description
resourceType	1	RO	See [i.39], clause 9.6.1.3 where this common attribute is described
resourceID	1	RO	See [i.39], clause 9.6.1.3 where this common attribute is described.
resourceName	1	WO	See [i.39], clause 9.6.1.3 where this common attribute is described.
parentID	1	RO	See [i.39], clause 9.6.1.3 where this common attribute is described.
accessControlPolicyIDs	0..1 (L)	RW	See [i.39], clause 9.6.1.3 where this common attribute is described
creationTime	1	RO	See [i.39], clause 9.6.1.3 where this common attribute is described
expirationTime	1	RW	See [i.39], clause 9.6.1.3 where this common attribute is described
lastModifiedTime	1	RO	See [i.39], clause 9.6.1.3 where this common attribute is described
Labels	0..1 (L)	RW	See [i.39], clause 9.6.1.3 where this common attribute is described
Creator	0..1	RO	The *AE-ID* of the entity which created the resource. This can also be the CSE-ID of the IN-CSE if the IN-CSE created the resource.
descriptor	1	RW	The semantic description of the resource whose child resource the <semanticDescriptor> resource is.
ontologyRef	0..1	WO	A reference (URI) of the ontology used to represent the information that is stored in the *descriptor attribute*. If this attribute is not present, the *ontologyRef* from the parent resource is used if present.

8.5.3.4 Example showing the uses of the Semantic Descriptor resource

This clause gives an example of how semantic annotations based on the Smart Appliance REFerence Ontology (SAREF) [i.40] can be used to describe an AE representing a smart appliance.

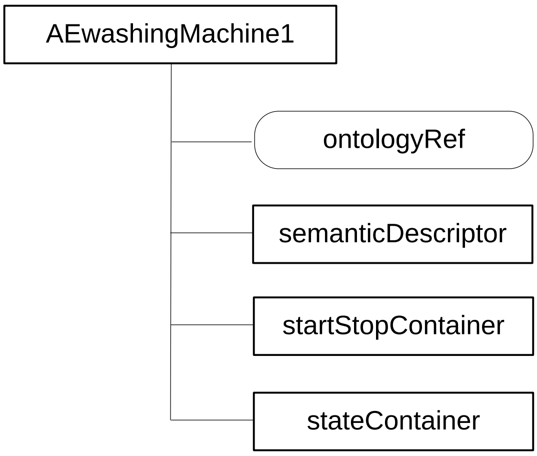


Figure 8.5.3.4-1: Resource structure of smart washing machine AE

Figure 8.5.3.4-1 shows the resource structure of an AE representing a smart washing machine.

NOTE: The assumption here is that the Washing Machine acts as an AE using the Mca interface. Here no distinction is made between the Washing Machine AE and the washing machine as a physical device. This can be done under the given assumption, but could lead to problems in other cases, e.g. if an interworking proxy is involved.

The resource includes an ontologyRef attribute, which contains the URI of the ontology concept of the smart washing machine, e.g. "http://ontology.tno.nl/saref#WashingMachine". The startStopContainer and the stateContainer represent the functional interface aspects of the washing machine, i.e. it can be started and stopped and the current state can be requested.

Table 8.5.3.4-1 shows the semantic annotation stored in the descriptor attribute of the semanticDescriptor resource. The information provides the link between the operations of the washing machines and the containers of the smart washing machine AE and describes the REST methods that can be executed on the containers. The washing operation can be started by executing a Create request on the startStopContainer whose URI is provided, the same for the state operation, where a Retrieve request on the *latest* contentInstance of the stateContainer will provide the current state of the washing machine.

Table 8.5.3.4-1: Semantic resource description of smart washing machine AE based on SAREF


```
<rdf:RDF
   <rdf:Description rdf:about="http://ontology.tno.nl/saref#WASH_LG_123">
      <rdf:type rdf:resource="http://ontology.tno.nl/saref#WashingMachine"/>
      <saref:hasManufacturer>LG</saref:hasManufacturer>
      <saref:hasDescription>Very cool Washing Machine</saref:hasDescription>
       <saref:hasLocation rdf:resource="http://ontology.tno.nl/saref#Bathroom"/>
      <msm:hasService rdf:resource="http://ontology.tno.nl/saref#WashingService_123"/>
      <msm:hasService rdf:resource="http://ontology.tno.nl/saref#StateService_123"/>
   </rdf:Description>

  <rdf:Description rdf:about="http://ontology.tno.nl/saref#WashingService_123">
      <rdf:type rdf:resource="http://ontology.tno.nl/saref#WashingService"/>
      <msm:hasOperation rdf:resource="http://ontology.tno.nl/saref#WashingOperation_123"/>
   </rdf:Description>

  <rdf:Description rdf:about="http://ontology.tno.nl/saref#WashingOperation_123">
  <rdf:type rdf:resource="http://ontology.tno.nl/saref#WashingOperation"/>
      <hr:hasMethod>Create</hr:hasMethod>
      <hr:hasURITemplate>/CSE1/WASH_LG_123/startStopContainer </hr:hasURITemplate>
      <msm:hasInput rdf:resource="http://ontology.tno.nl/saref#Action"/>
   </rdf:Description>

   <rdf:Description rdf:about="http://ontology.tno.nl/saref#StateService123">
 <rdf:type rdf:resource="http://ontology.tno.nl/saref#StateService"/>
      <msm:hasOperation rdf:resource=”http://ontology.tno.nl/saref#StateOperation123/>
   </rdf:Description>

   <rdf:Description rdf:about= http://ontology.tno.nl/saref#StateOperation123>
  <rdf:type rdf:resource="http://ontology.tno.nl/saref#StateOperation"/>
      <hr:hasMethod>Retrieve</hr:hasMethod>
      <hr:hasURITemplate>/CSE1/WASH_LG_123/state/stateContainer/latest</hr:hasURITemplate>
      <msm:hasOutput rdf:resource="http://ontology.tno.nl/saref#State"/>
   </rdf:Description>
</rdf:RDF>
```


8.5.3.5 Semantic Instance Management

8.5.3.5.1 Overview

The management of semantic instances stored in the *<semanticDescriptor>* resource is one of key semantic functionalities , including the create, update and delete operations against semantic instances. A simple way to deal with the update of semantic instances is to overwrite the whole *<semanticDescriptor>* resource which might lead to data redundancy problems, but a more efficient approach is needed to handle the management of semantic instances. An approach using functions e.g SPARQL HTTP POST, SPARQL HTTP PUT etc provided by SPARQL to update or create semantic instances is proposed.

8.5.3.5.2 Concrete Example of Managing Semantic Instance

In the oneM2M system, , in order to change a semantic instances in the *<semanticDescriptor>* resources, the system has to change the whole semantic descriptor.

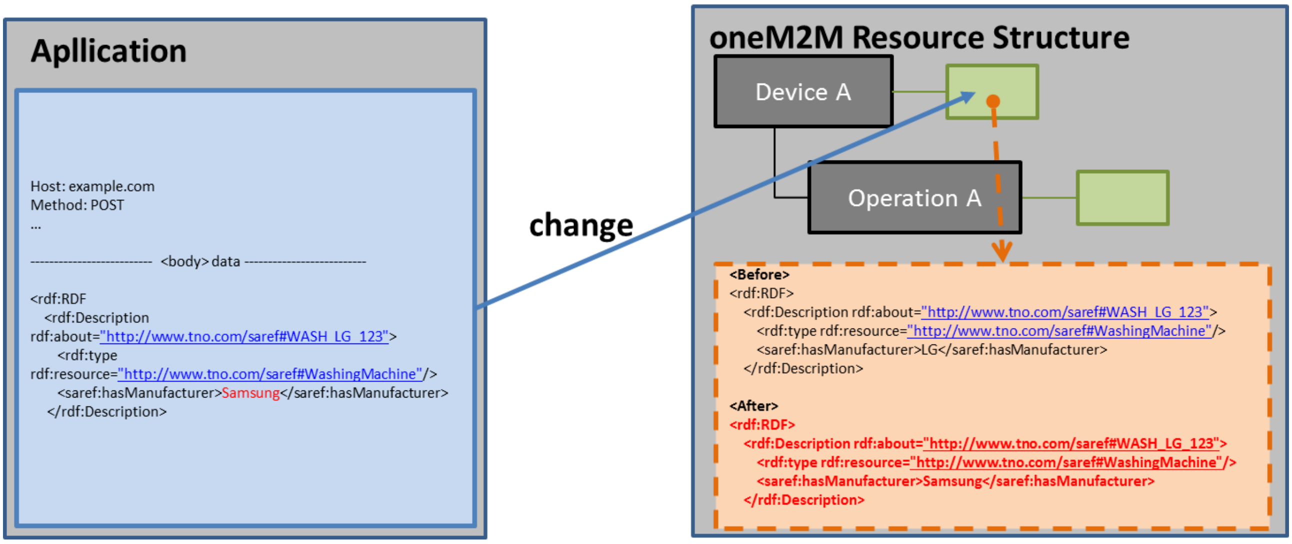


Figure 8.5.3.2-1: Concreate example of managing semantic instance in the oneM2M system

8.5.3.5.3 Managing Semantic Instances using SPARQL update operation

As mentioned in clause 8.5.3, a semantic description consisting of semantic instances is contained in the <*semanticDescriptor*> resource. In order to manage semantic instances using the SPARQL update operation, it is assumed that the semantic description is stored in the <*semanticDescriptor*> resource as depicted in figure 4.

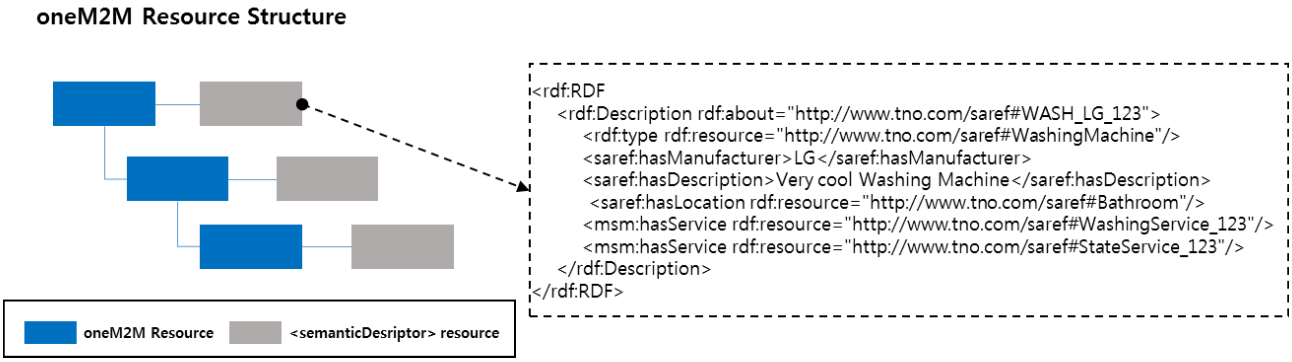


Figure 4: Representation of a semantic description in a <semanticDescripor> resource

To manage the semantic instances, the SPARQL update operation has to be mapped to the oneM2M UPDATE (U) procedure. The UPDATE procedure can be used by an AE Originator to manage the semantic instances stored in the <*semanticDescriptor*> resource on a Receiver CSE (also called the Hosting CSE).

Originator is responsible for sending requests to update semantic instances stored in the <*semanticDescriptor*> resource by using the UPDATE method.

Hosting CSE processes the update procedures against the requested semantic instances if the originator is allowed to do the update operation. Figure 5 shows the interaction between Originator and Receiver and the procedures are processed as follows.

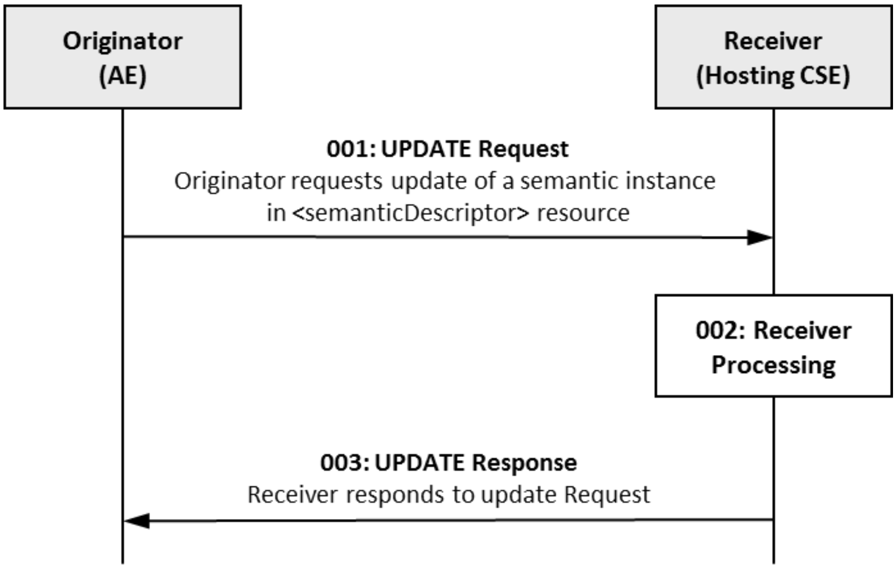


Figure 5: Procedure for managing the semantic instances using oneM2M UPDATE operation

Step 001: The Originator can include mandatory parameters and partial or whole optional parameters in the Request message for UPDATE operation. In **Step 001**, different SPARQL statements can be used in order to update (i.e. add, delete, and modify) an existing semantic description in a target <*semanticDescriptior*> resource at the Receiver as shown in the following cases.

- **Case 1:** This case is to add semantic instances (e.g., RDF, triples) to an existing semantic description in a target <*semanticDescriptor*> resource in the Receiver. In this case, the Originator can include `INSERT DATA` or `INSERT` SPARQL statements in a Request as shown in the block below. The `INSERT DATA` statement can add semantic instances using the `RDF PAYLOAD` to the semantic description in a target <*semanticDescriptor*> resource. Thus, `RDF PAYLOAD` in an `INSERT DATA` SPARQL statement disallows blank nodes or variables. On the other hand, an `INSERT` SPARQL statement can add semantic instances corresponding to a `template` by copying semantic instances from a source <*semanticDescriptor*> resource to a target <*semanticDescriptor*> resource based on a `pattern`. Accordingly, a `template` in the `INSERT` statement allows blank nodes or variables with conditional SPARQL statements. Examples 1 and 2 give examples using `INSERT DATA` and `INSERT` SPARQL statements, respectively.

```
======================== INSERT DATA statement =========================
INSERT DATA
{GRAPH <Target URI of <semanticDescriptor> resource> {RDF PAYLOAD} }
========================= INSERT statement ===========================
INSERT {
GRAPH <Target URI of <semanticDescriptor> resource 1> {template} }
WHERE {GRPAH <Target URI of <semanticDescriptor> resource 2> {pattern} }
```

EXAMPLE 1: Add semantic instance to a <*semanticDescriptor*> resource using INSERT DATA statement:

```
INSERT DATA
{
GRAPH <http://<Hosting CSE address>/<CSEBase>/<AE>/<semanticDescriptor>>
{saref:WASH_LG_123 msm:hasOperation saref:WashingOperation_123}
}
```

EXAMPLE 2: Add semantic instance to a <*semanticDescriptor*> resource using INSERT statement:

```
INSERT {
GRAPH <http://<Hosting CSE address>/<CSEBase>/<AE>/<semanticDescriptor1>>
{?a saref:hasManufacturer ?c}
}
WHERE {
GRAPH <http://<Hosting CSE address>/<CSEBase>/<AE>/<semanticDescriptor2>>
```

```
{?a saref:hasManufacturer ?c}
}
```

- **Case 2:** This case is to remove any of the semantic instances from an existing semantic description in a target <*semanticDescriptor*> resource in the Receiver. In this case, the Originator can include two different SPARQL statements including `DELETE DATA` statement or `DELETE` statement in the Request as shown in the block below.
  The `DELETE DATA` statement can remove specific semantic instances using `RDF PAYLOAD` from an existing semantic description in a target <*semanticDescriptor*> resource. Hence, the `RDF PAYLOAD` in the `DELETE DATA` SPARQL statement is not allowed to contain blank nodes or variables. However, the `DELETE` SPARQL statement can remove semantic instances corresponding to the `template` through matching semantic instances based on a `pattern`. Accordingly, the `template` in the `DELETE` SPARQL statement is allowed to contain blank nodes or variables with conditional SPARQL statements. Examples 3 and 4 give examples using `DELETE DATA` and `DELETE` SPARQL statements, respectively.

```
======================== DELETE DATA statement ========================
DELETE DATA
{GRAPH <Target URI of <semanticDescriptor> resource> {RDF PAYLOAD} }
========================== DELETE statement ============================
WITH <Target URI of <semanticDescriptor> resource>
DELETE {template }
WHERE {pattern}
```

EXAMPLE 3: Remove semantic instances in the <*semanticDescriptor*> resource using `DELETE DATA` statement:

```
DELETE DATA
{
GRAPH <http://<Hosting CSE address/<CSEBase>/<AE>/<semanticDescriptor>>
{saref:WASH_LG_123 msm:hasService saref:StateService_123}
}
```

EXAMPLE 4: Remove semantic instance in the <*semanticDescriptor*> resource using `DELETE` statement:

```
WITH <<http://<Hosting CSE address/<CSEBase>/<AE>/<semanticDescriptor>>
DELETE {?a msm:hasService saref:StateService_123}
WHERE {?a saref:hasManufacturer 'LG'}
```

- **Case 3:** This case is to modify any of the semantic instances from the semantic description in a target <*semanticDescriptor*> resource in the Receiver. In this case, the Originator can include `DELETE/INSERT SPARQL` statements with `template1` and `template2` in the Request as shown in the block below. At this time, blank nodes or variables are allowed in each `template`.

```
WITH <Target URI of <semanticDescriptor> resource>
DELETE {template1} INSERT {template2}
WHERE {pattern}
```

EXAMPLE 5: Modify semantic instances in a <*semanticDescriptor*> resource using DELETE/INSERT operation:

```
WITH http://<Hosting CSE address/<AE>/<semanticDescriptor>
DELETE {?a saref:hasManufacturer 'LG'}
INSERT {?a saref:hasManufacturer 'SAMSUNG'}
WHERE {?a saref:hasManufacturer 'LG'}
```

Step 002: The Receiver will verify the existence (including ***Filter Criteria*** checking, if it is given) of the requested resource first and whether the Originator has the appropriate privilege to update the requested resource. On successful validation, the Receiver can update the semantic instances according to the SPARQL statements in the Request message. The update procedures are processed as follows:

- **According to Case 1:** If the `INSERT DATA` SPARQL statement is included in Request message, the `RDF PAYLOAD` in the statement will be added to the target <*semanticDescriptor*> resource. However, if the `RDF PAYLOAD` already exists in the target <*semanticDescriptor*> resource, then the Receiver will return a failure request status with additional error information through **Step 003**. If the `INSERT` SPARQL statement included in the Request message, the Receiver adds semantic instances corresponding to `template` by copying semantic instances from the source <*semanticDescripor*> resource to the target <*semanticDescriptor*> resource based on the `pattern`. At this time, if there are no existing semantic instances corresponding to `template` or matched semantic instances based on `pattern`, the Receiver will return a failure request status with additional error information through **Step 003.** The blocks below show the processing result of Example 1 and Example 2 presented in **Case 1**.

Result of Examples 1 and 2 presented in Case 1 using semantic description in figure 4.

Description before:

```
<rdf:RDF
    <rdf:Description rdf:about="http://ontology.tno.nl/saref#WASH_LG_123">
        <rdf:type rdf:resource="http://ontology.tno.nl/saref#WashingMachine"/>
        <saref:hasManufacturer>LG</saref:hasManufacturer>
        <saref:hasDescription>Very cool Washing Machine</saref:hasDescription>
        <saref:hasLocation rdf:resource="http://ontology.tno.nl/saref#Bathroom"/>
        <msm:hasService rdf:resource="http://ontology.tno.nl/saref#WashingService_123"/>
        <msm:hasService rdf:resource="http://ontology.tno.nl/saref#StateService_123"/>
    </rdf:Description>
</rdf:RDF>
```

Description after:

```
<rdf:RDF
    <rdf:Description rdf:about="http://ontology.tno.nl/saref#WASH_LG_123">
        <rdf:type rdf:resource="http://ontology.tno.nl/saref#WashingMachine"/>
        <saref:hasManufacturer>LG</saref:hasManufacturer>
        <saref:hasDescription>Very cool Washing Machine</saref:hasDescription>
         <saref:hasLocation rdf:resource="http://ontology.tno.nl/saref#Bathroom"/>
        <msm:hasService rdf:resource="http://ontology.tno.nl/saref#WashingService_123"/>
        <msm:hasService rdf:resource="http://ontology.tno.nl/saref#StateService_123"/>
        <msm:hasOperationrdf:resource="http://ontology.tno.nl/saref#WashingOperation_123"/>
    </rdf:Description>
</rdf:RDF>
```

- **According to Case 2:** If the `DELETE DATA` SPARQL statement is included in Request message, then the `RDF PAYLOAD` in the statement will be removed from the target <*semanticDescriptor*> resource. At this time, if the `RDF PALOAD` does not exist in the resource, then the Receiver can return a failure request status with additional error information through **Step 003**. If the `DELETE` SPARQL statement is included in the Request message, the Receiver will remove all of semantic instances corresponding to the template based on a pattern. Accordingly, if there are no existing semantic instances corresponding to `template` or matched semantic instances base on `pattern`, then the Receiver will return a failure request status with additional error information through **Step 003**. The block below shows the processing result of Example 3 and Example 4 presented in **Case 2**.

Result of Example 2 presented in Case 2 using semantic description in figure 4.

Description before:

```
<rdf:RDF
    <rdf:Description rdf:about="http://ontology.tno.nl/saref#WASH_LG_123">
        <rdf:type rdf:resource="http://ontology.tno.nl/saref#WashingMachine"/>
        <saref:hasManufacturer>LG</saref:hasManufacturer>
        <saref:hasDescription>Very cool Washing Machine</saref:hasDescription>
         <saref:hasLocation rdf:resource="http://ontology.tno.nl/saref#Bathroom"/>
```

```
        <msm:hasService rdf:resource="http://ontology.tno.nl/saref#WashingService_123"/>
        <msm:hasService rdf:resource="http://ontology.tno.nl/saref#StateService_123"/>
    </rdf:Description>
</rdf:RDF>
```

Description after:

```
<rdf:RDF
    <rdf:Description rdf:about="http://ontology.tno.nl/saref#WASH_LG_123">
        <rdf:type rdf:resource="http://ontology.tno.nl/saref#WashingMachine"/>
        <saref:hasManufacturer>LG</saref:hasManufacturer>
        <saref:hasDescription>Very cool Washing Machine</saref:hasDescription>
         <saref:hasLocation rdf:resource="http://ontology.tno.nl/saref#Bathroom"/>
        <msm:hasService rdf:resource="http://ontology.tno.nl/saref#WashingService_123"/>
    </rdf:Description>
</rdf:RDF>
```

- **According to Case 3:** If the target semantic instance included in the Request message exists in the <*semanticDescriptor*> resource, then the Receiver will remove all semantic instances corresponding to `template1` from the <*semanticDescriptor*> resources, and then the Receiver will create new semantic instances corresponding to `template2` in the <*semanticDescriptor*> resource base on `pattern`. In case that either the target semantic instance corresponding to `template1` requested to be removed does not exist in the <*semanticDescriptor*> resource or the target semantic instance corresponding to `template2` requested to be added to the <*semanticDescriptor*> resource already exists, the Receiver will return a failure request status with additional error information through **Step 003**. The block below shows the processing result of Example 3 presented in **Case 3**.

Result of Example 3 presented in Case 3 using semantic description in figure 4.

Description before

```
<rdf:RDF
    <rdf:Description rdf:about="http://ontology.tno.nl/saref#WASH_LG_123">
        <rdf:type rdf:resource="http://ontology.tno.nl/saref#WashingMachine"/>
        <saref:hasManufacturer>LG</saref:hasManufacturer>
        <saref:hasDescription>Very cool Washing Machine</saref:hasDescription>
         <saref:hasLocation rdf:resource="http://ontology.tno.nl/saref#Bathroom"/>
        <msm:hasService rdf:resource="http://ontology.tno.nl/saref#WashingService_123"/>
        <msm:hasService rdf:resource="http://ontology.tno.nl/saref#StateService_123"/>
    </rdf:Description>
</rdf:RDF>
```

Description after:

```
<rdf:RDF
    <rdf:Description rdf:about="http://ontology.tno.nl/saref#WASH_LG_123">
        <rdf:type rdf:resource="http://ontology.tno.nl/saref#WashingMachine"/>
        <saref:hasManufacturer>SAMSUNG</saref:hasManufacturer>
        <saref:hasDescription>Very cool Washing Machine</saref:hasDescription>
         <saref:hasLocation rdf:resource="http://ontology.tno.nl/saref#Bathroom"/>
        <msm:hasService rdf:resource="http://ontology.tno.nl/saref#WashingService_123"/>
        <msm:hasService rdf:resource="http://ontology.tno.nl/saref#StateService_123"/>
    </rdf:Description>
</rdf:RDF>
```

Step 003: The Receiver will include all mandatory parameters and partial or whole optional parameters in the Response message for the UPDATE operation.

8.5.4 Semantic Filtering and Discovery

In clause 8.5.3, the <*semanticDescriptor*> resource has been introduced. One of the key functionalities generating value from the semantic descriptions contained in such resources is enabling semantic filtering. Semantic filtering is especially relevant if an application wants to discover resources and specifies the characteristics of the resources it is interested in.

Previously, filtering has been supported by having filter criteria on attributes (see [i.39], table 8.1.2-1). By adding support for *<semanticDescriptor>* child resources containing the semantic description, a filter criteria has to be added that pertains to this semantic descriptions. Table 8.5.4-1 shows the definition of the *semantics* filter criteria. There can be multiple instances, which according to the general semantics for evaluating filter criteria, means that an "OR" semantics applies, i.e. the overall result for the semantics filter criteria is true if one or more of the semantic filters matches the semantic description.

Table 8.5.4.1-1: *semantics* Filter Criteria

semantics	0..n	The semantic description contained in one of the <semanticDescriptor> child resources matches the specified semantic filter.

Since the representation of the semantic descriptions has not been specified in the architecture, the definition of the *semantics* filter criteria there remains on a high abstraction level as well. It has to be decided, whether the architecture description has to become more specific in this respect, or whether this is left for stage 3.

For the purpose of this technical report, the assumption is that the semantic descriptions are specified as RDF triples - in whatever representation, e.g. RDF/XML, Turtle. The *semantics* filter criteria can thus be specified as SPARQL requests that are executed on each of the semantic descriptions. The interpretation is that the semantic filter evaluates to *true* whenever the execution of the SPARQL request provides one or more results and *false* otherwise.

In the following two examples are shown. The semantic descriptors are given as triples. For readability the ontology concepts and instances are shown with namespaces, but the namespace and prefix definitions are omitted. The entries are shown as subject-predicate-object triples, or domain - object property-range as used in the definition of OWL object properties.

EXAMPLE 1: Filter for AE resources representing devices that measure temperature.

Semantic Descriptor of Device 1 AE

my:MyDevice1	rdf:type	base:Device
my:MyDevice 1	base:hasService	my:MyService1
my:MyService1	base:hasFunctionality	my:MyFunctionality1
my:MyFunctionality1	rdf:type	base:Measuring
my:MyFunctionality1	base:refersTo	my:MyAspect1
my:myAspect1	rdf:type	aspect:Temperature

Semantic Descriptor of Device 2 AE

my:MyDevice2	rdf:type	base:Device
my:MyDevice2	base:hasService	my:MyService2
my:MyService2	base:hasFunctionality	my:myFunctionality2
my:myFunctionality2	rdf:type	base:Controlling
my:myFunctionality2	base:refersTo	my:myAspect2
my:myAspect2	rdf:type	aspect:Temperature

SPARQL Request 1

```
SELECT ?device
    WHERE { ?device rdf:type base:Device .
                         ?device base:hasService ?service .
                         ?service base:hasFunctionality ?functionality .
                         ?functionality rdf:type base:Measuring .
                         ?functionality base:refersTo ?aspect .
                         ?aspect rdf:type instance:Temperature  }
```

SPARQL Execution Results

- (On Device1 semantic description) --> my:myDevice1.
- (On Device 2 semantic description) --> *empty*.

This means that the AE resource that is described by my:myDevice1 will be included in the result set, whereas the AE resource described by my:myDevice2 will not be included.

NOTE: The following SPARQL request would yield the same result in our case, but without checking whether the functionality is actually offered by a device (base:Device). The core concept identifying what is being represented by the resource (in this case a device) could be specified in the *ontologyRef* attribute.

SPARQL Request 2

```
SELECT ?functionality
    WHERE {  ?functionality rdf:type base:Measuring .
             ?functionality base:refersTo ?aspect .
             ?aspect rdf:type instance:Temperature  }
```

SPARQL Execution Results

- (On Device1 semantic description) --> my:myDevice1.
- (On Device 2 semantic description) --> *empty*.

EXAMPLE 2: Filter for contentInstance resources that contain a temperature value in Celsius with an accuracy of ±1 °C.

Semantic Descriptor of Temperature 1 contentInstance

my:myTemperature123	rdf:type	aspect:Temperature
my:myTemperature123	tempOnt:hasUnit	tempOnt:Celsius
my: myTemperature123	tempOnt:has Accuracy	0,9

Semantic Descriptor of Temperature 2 contentInstance

my:myTemperature234	rdf:type	aspect:Temperature
my:myTemperature234	tempOnt:hasUnit	tempOnt:Celsius
my: myTemperature234	tempOnt:has Accuracy	1,5

Semantic Descriptor of Temperature 3 contentInstance

my:myTemperature345	rdf:type	aspect:Temperature
my:myTemperature345	tempOnt:hasUnit	tempOnt:Fahrenheit
my: myTemperature345	tempOnt:has Accuracy	0,5

SPARQL Request

```
SELECT ?temperatureInstance
    WHERE {  ?temperatureInstance rdf:type tempOnt:Temperature  .
                         ?temperatureInstance tempOnt:hasUnit tempOnt:Celsius .
                         ?temperatureInstance tempOnt:hasAccuracy ?accuracy
                         FILTER (?accuracy <= 1.0) }
```

SPARQL Execution Results

- (On Temperature1 description) --> my:myTemperature123.
- (On Temperature 2 description) --> *empty* - not accurate enough.
- (On Temperature 3 description) --> *empty* - not in Celsius.

8.5.5 Semantic Filtering on Distributed Semantic Descriptors

8.5.5.1 Problem Description

In the previous clause, semantic filtering criteria have been introduced and it has been described how they are applied to the content of the <*semanticDescriptor*> child resource. However, in some cases the relevant semantic information is not contained in the <*semanticDescriptor*> child resource directly, but in a different <semanticDescriptor> resource. For example, this could be the case if looking for devices that can provide temperature output. This semantic information may not be directly attached to the resource representing the device, but there is another resource that represents a specific operation of the device and the semantic description can be found there. In order to correctly include the device in the result set, the semantic information attached to the operation has to be considered.

Figure 8.5.5.1-1 shows the general situation. In the lower part of the figure, a semantic graph representing subject-predicate-object relations is shown. In a oneM2M system, different parts of this graph may be stored in different <*semanticDescriptor*> resources. If semantic operations like the semantic filtering are to be applied to (parts of) the complete semantic graph, the different parts of the graph have to be linked and these links have to be followed when executing the semantic operation.

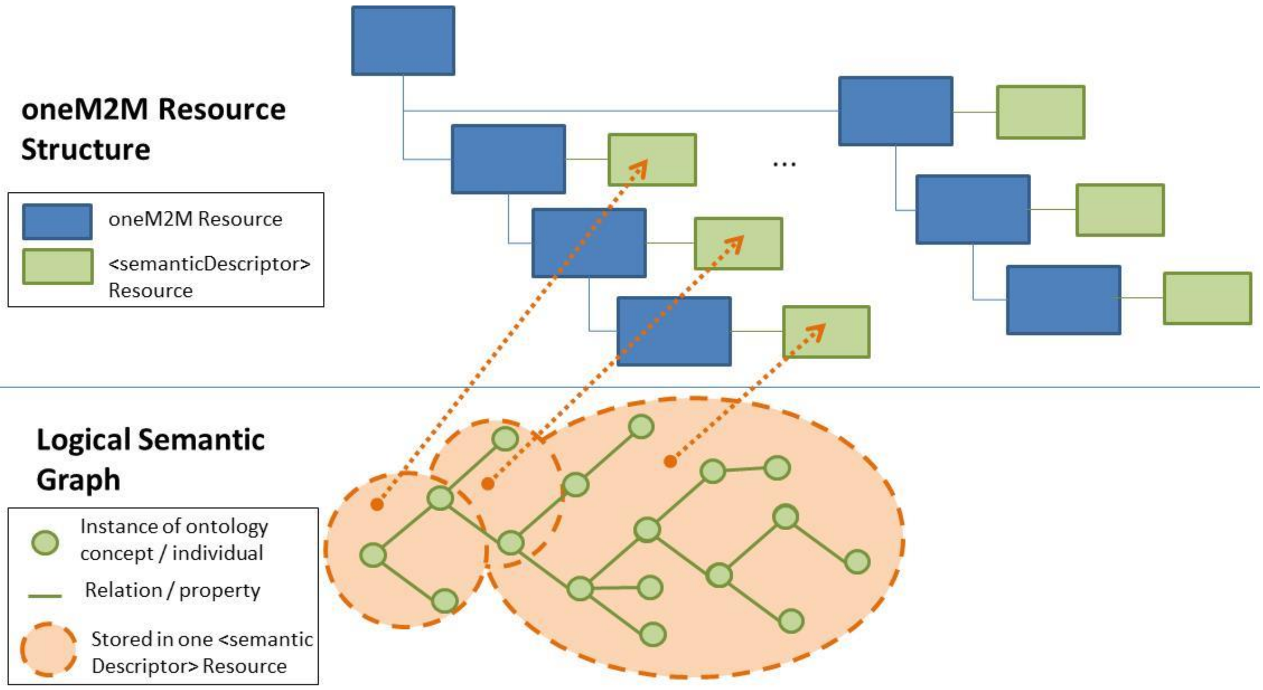


Figure 8.5.5.1-1: Mapping of logical semantic graph to oneM2M resource structure

Figure 8.5.5.1-2 shows the case of a semantic filter whose scope takes into account semantic information stored in different <*semanticDescriptor*> resources.

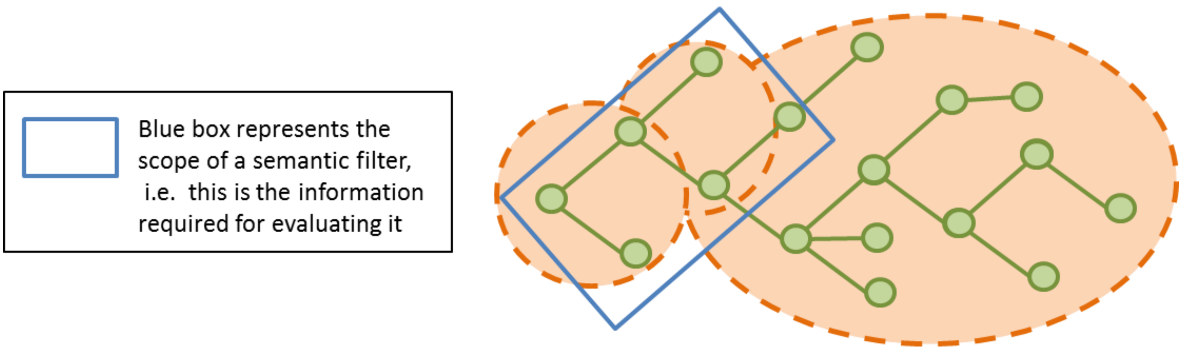


Figure 8.5.5.1-2: Scope of semantic filter across semantic information stored in different resources

8.5.5.2 Related Solutions

- Semantic Web:
  - In the semantic web, the URI identifying the class instance can be directly de-referenced, possibly resulting in some re-direction. This means that for each instance the related information can be found based on its URI. In the oneM2M case, the semantic instances are not first-class citizens, it is only resources that can be accessed and the information about instances is stored as content of the resources. This means that based on only the URI of the semantic information the related information cannot be found, so the approach is not applicable in the oneM2M case.
- Federated SPARQL queries:
  - SPARQL [i.28] supports federated queries using the SERVICE keyword, where the URL of a remote SPARQL endpoint can be specified.This approach would work only, if the requestor would a-priori know which semantic descriptors contain which information. This is not the case here as the resources are not known, it is the purpose of the filter to select them, therefore the approach is also not applicable.

8.5.5.3 Proposed Solution 1

8.5.5.3.0 Introduction

The underlying assumption is that the semantic description stored in the <semanticDescriptor> resources are represented as RDF triples [i. 26] in some kind of serialization. The semantic descriptions can be part of an overall semantic description that is distributed across the <semanticDescriptor> resources. The RDF triples are based on classes and properties defined in OWL.

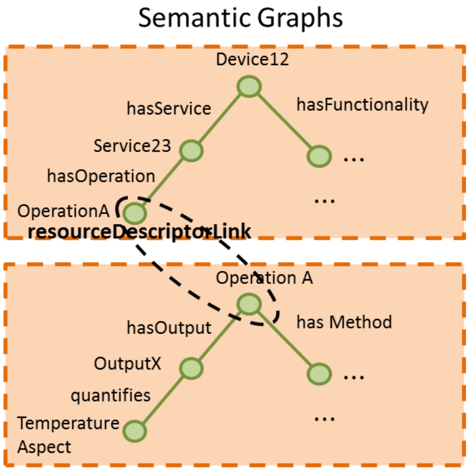


Figure 8.5.5.3.0-1: Parts of semantic descriptions stored in different <semanticDescriptor> resources

Figure 8.5.5.3.0-1 shows an example of two semantic descriptions in the form of RDF triples, visualized as semantic graphs, each stored in a <semanticDescriptor> resource. Logically the two graphs form a combined semantic graph, as the "OperationA" instance is part of both trees, both in the subject and object role of a triple.

To enable semantic operations across the overall logical tree, the proposal is to add an annotation link in the form of a *resourceDescriptorLink* OWL annotation property. This annotation property can be specified for any class instance and its value is the URL of a <semanticDescriptor> resource, where additional RDF triples for the given class instance can be found.

For enabling semantic filtering on semantic descriptions stored across <semanticDescriptor> resources connected by *resourceDescriptorLink* properties, the SPARQL-based semantic filtering engine has to be modified in the following way:

- The semantic filter formulated as a SPARQL request is executed on the content of the semantic descriptor resource of the candidate resource.

- If in the course of the execution a class instance with one or more *resourceDescriptorLink* annotations is encountered, the execution is halted.
- The content of each of the <semanticDescriptor> resources the *semanticDescriptorLink* references is added to the content on which the SPARQL request is being executed (lazy evaluation, alternative: fetch everything before execution, but may result in fetching unnecessary information).
- The execution of the SPARQL request is continued on the enlarged content.

Advantages of the proposed solution 1:

- The required semantic information can be found.
- The application adding the resourceDescriptorLinks only has to be aware of directly related instances in other <semanticDescriptor> resources.
- Resource-based access control can be easily enforced as the information is being accessed at execution time of the operation and the access privileges of the requester can be applied.

Disadvantages of the proposed solution 1:

- The execution of the SPARQL request has to be changed to allow retrieving semantic descriptor information whenever a *resourceDescriptorLink* has to be followed for which the information was not already previously retrieved.

8.5.5.3.1 Examples for Solution 1

Figure 8.5.5.3.1-1, 8.5.5.3.1-2 and 8.5.5.3.1-3 show three different examples, where semantic descriptions are stored across <semanticDescriptor> resources as they describe aspects more closely related to different oneM2M resources. Nevertheless, the filter request needs to be applied across the distributed semantic description.

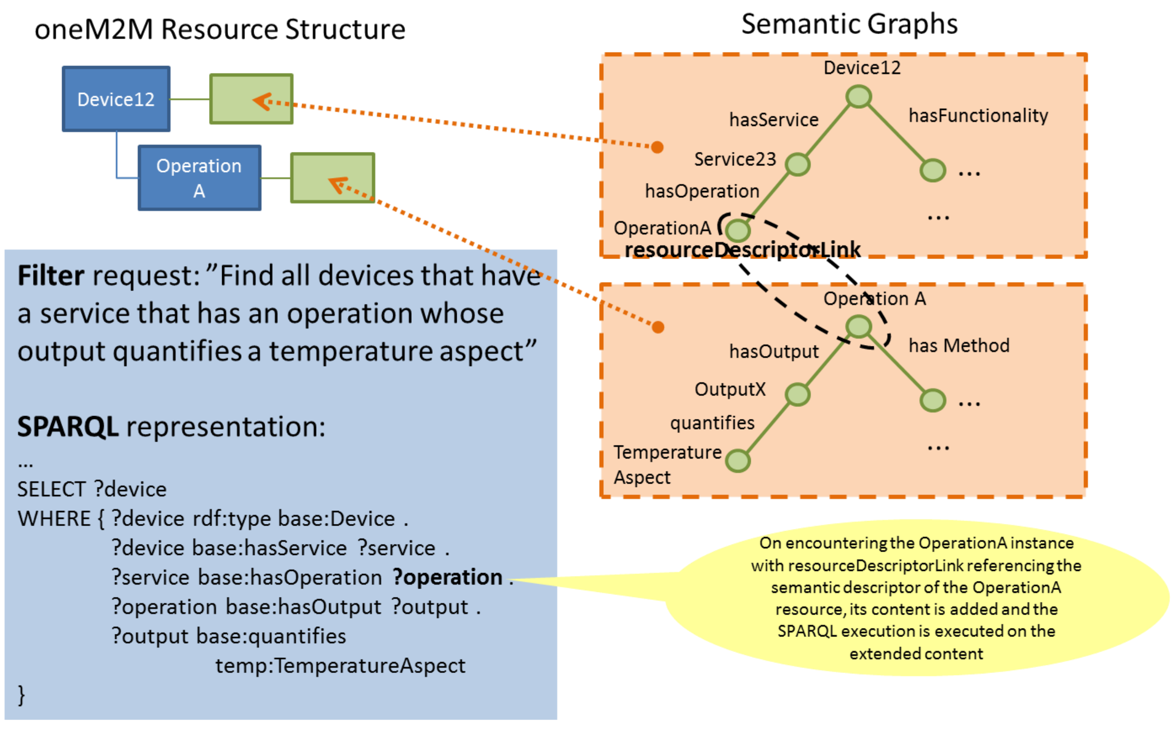


Figure 8.5.5.3.1-1: Semantic filter for devices that produce temperature output

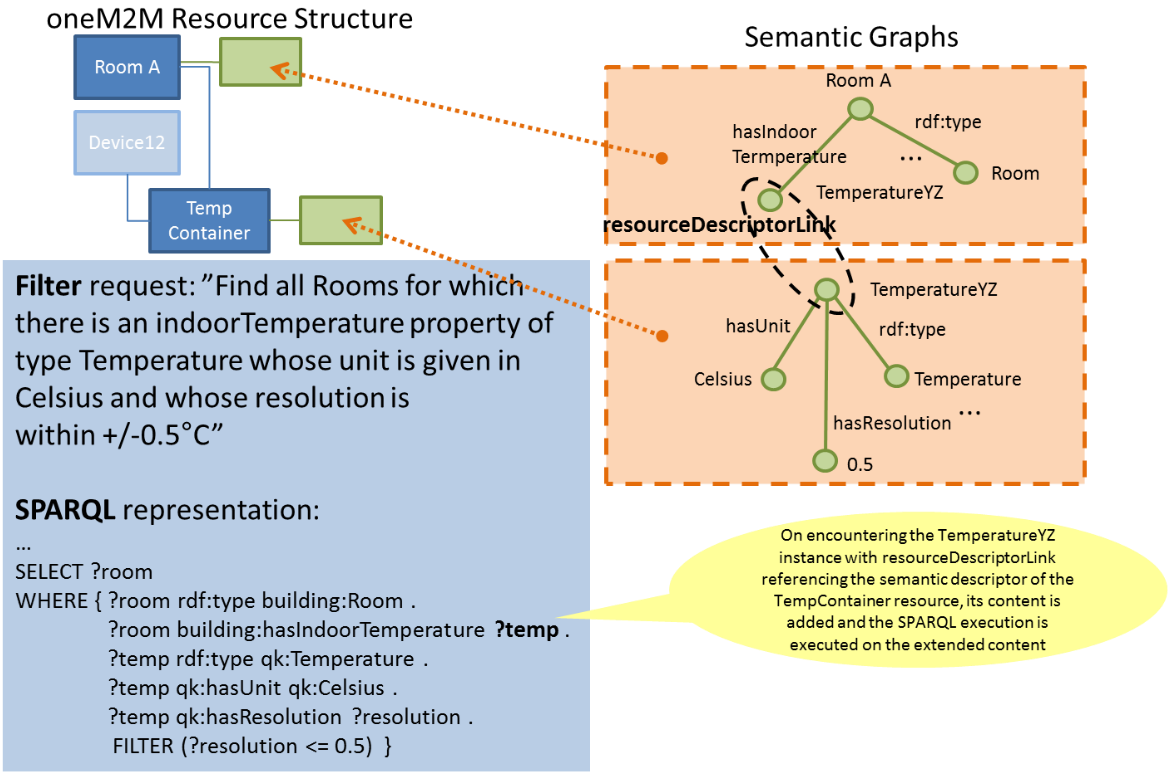


Figure 8.5.5.3.1-2: Semantic filter for rooms with indoor temperature in Celsius

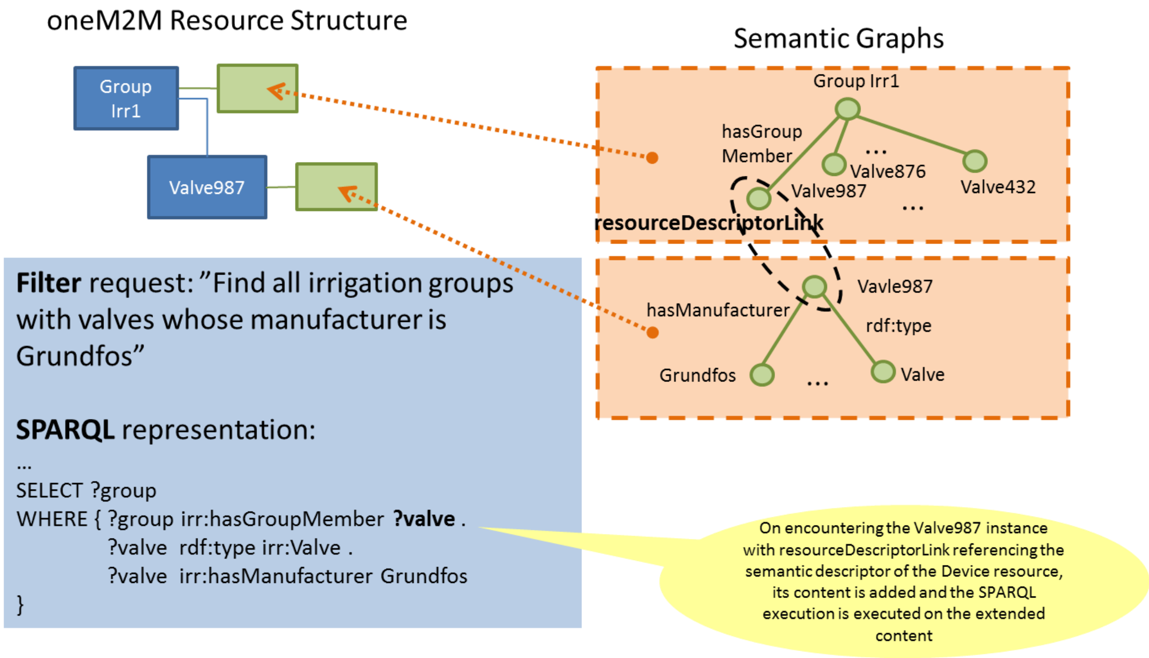


Figure 8.5.5.3.1-3: Semantic filter for groups of valves that include a valve manufactured by Grundfos

8.5.5.4 Proposed Solution 2

8.5.5.4.0 Introduction

The same underlying assumptions as in the previous solution are made:

- The semantic descriptions stored in the <*semanticDescriptor*> resources are represented as RDF triples [i.26] in some kind of serialization.

- The semantic descriptions can be part of an overall semantic description that is distributed across the <*semanticDescriptor*> resources.

- The RDF triples are based on classes and properties defined in OWL.

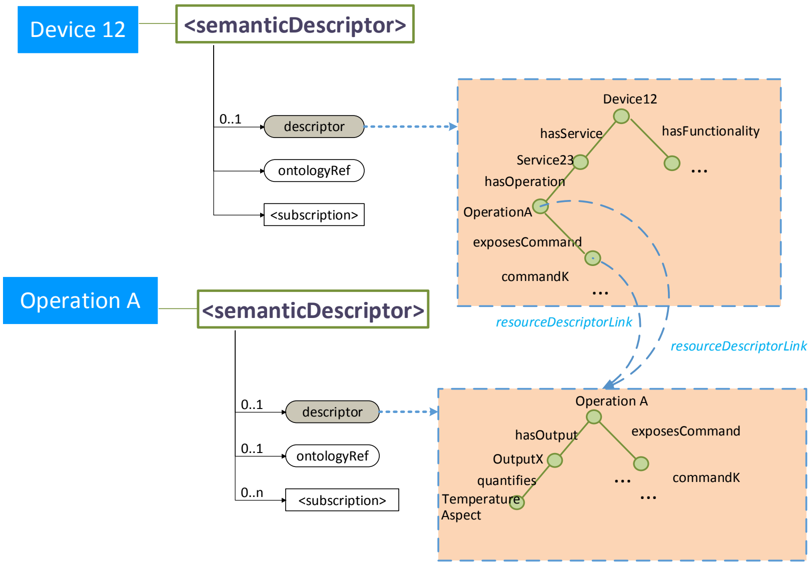


Figure 8.5.5.4.0-1: Distributed concepts across different <semanticDescriptor> resources

Figure 8.5.5.4.0-1 shows the example of two semantic descriptions in the form of RDF triples, visualized as semantic graphs, each stored in a separate <*semanticDescriptor*> resource. The two descriptions are logically related, as "OperationA" and "CommandK" instances are part of both trees. Other concept instances might be common between the two trees, so several resourceDescriptorLinks might be established between the same two descriptors.

In this proposal an attribute *relatedSemantics* is added to the <*semanticDescriptor*> resource to indicate all the resources with semantics descriptors related to the current one.

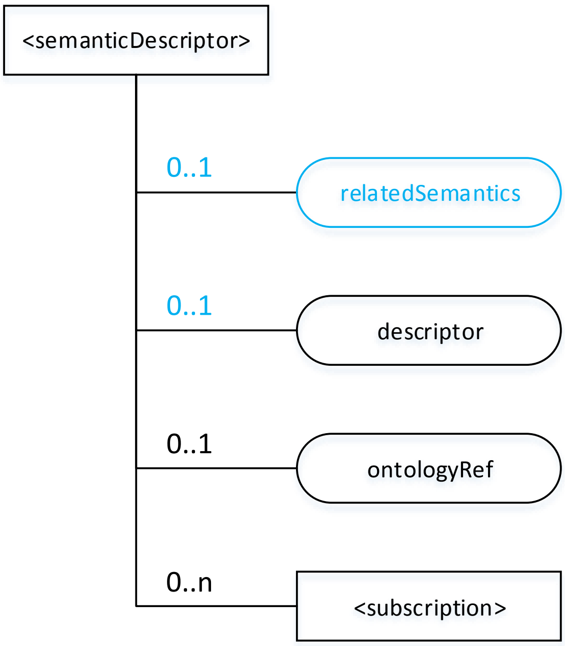


Figure 8.5.5.4.0-2: *relatedSemantics* attribute

Two representations of the *relatedSemantics* information are envisioned, and may be used in the same implementation:

- List of links: In this case the *relatedSemantics* attribute contains a list of links pointing to other <sematicDescriptor> resources which should be used together to perform semantic queries as described in clause 8.5.5.4.1.
- Group of links: In this case the *relatedSemantics* attribute points to either:
  - a <*group*> resource with the functionality described in clause 8.5.5.4.2;
  - a newly proposed <*semanticGroup*> resource as described in clause 8.5.5.4.3.

The way the SPARQL query results are produced is detailed with the previous example of graphs containing information relevant to each other, but which are stored in two independent resources <Device12>, and <OperationA>. Semantically the *OperationA* concept from the first graph is further described in the descriptor of <Device12>, while the relationship *exposesCommand* and object *commandK* are contained in both.

The goal is to enable the creation of a larger resultant graph to be submitted for evaluation of the SPARQL query, as shown infigure 8.5.5.4.0-3.

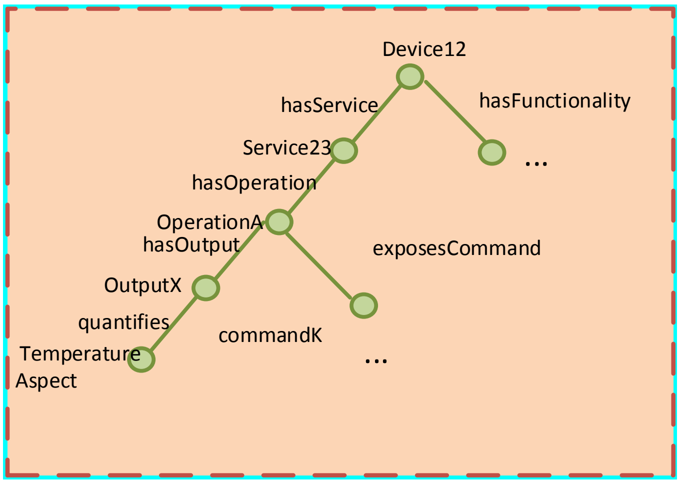


Figure 8.5.5.4.0-3: Composite graph from sub-graphs distributed in separate descriptors

8.5.5.4.1 "List of links" use

In this representation the *relatedSemantics* attribute contains a list of links pointing to descriptors associated with other resources, which should be used together with the descriptor in the given <*semanticDescriptor*> resource in order to perform semantic queries. In our case the list points to the semantic descriptor of <operationA>.

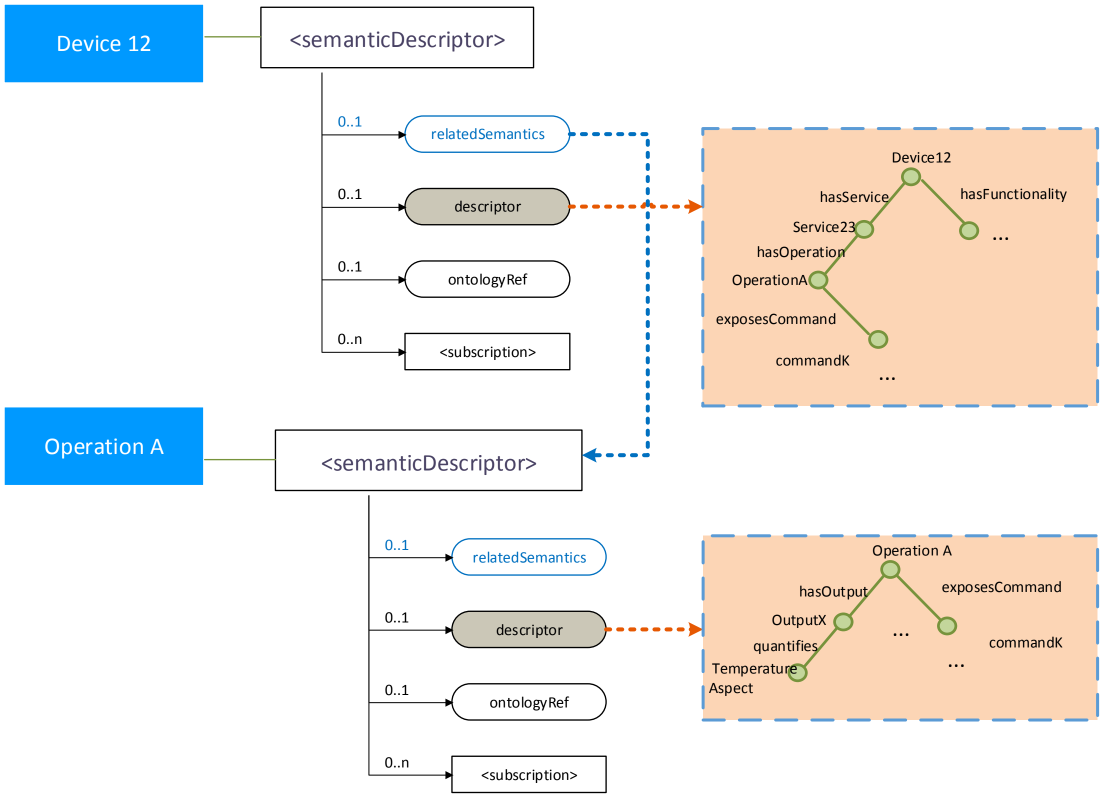


Figure 8.5.5.4.1-1: Use of *relatedSemantics* attribute with a list of links

This representation is useful in cases like the one presented, when a more limited number of semantic descriptors are related, making the link list short.

8.5.5.4.2 <group> Resource use

In this representation the *relatedSemantics* attribute points to a <group> resource which includes <Device12> and <operationA> resources.

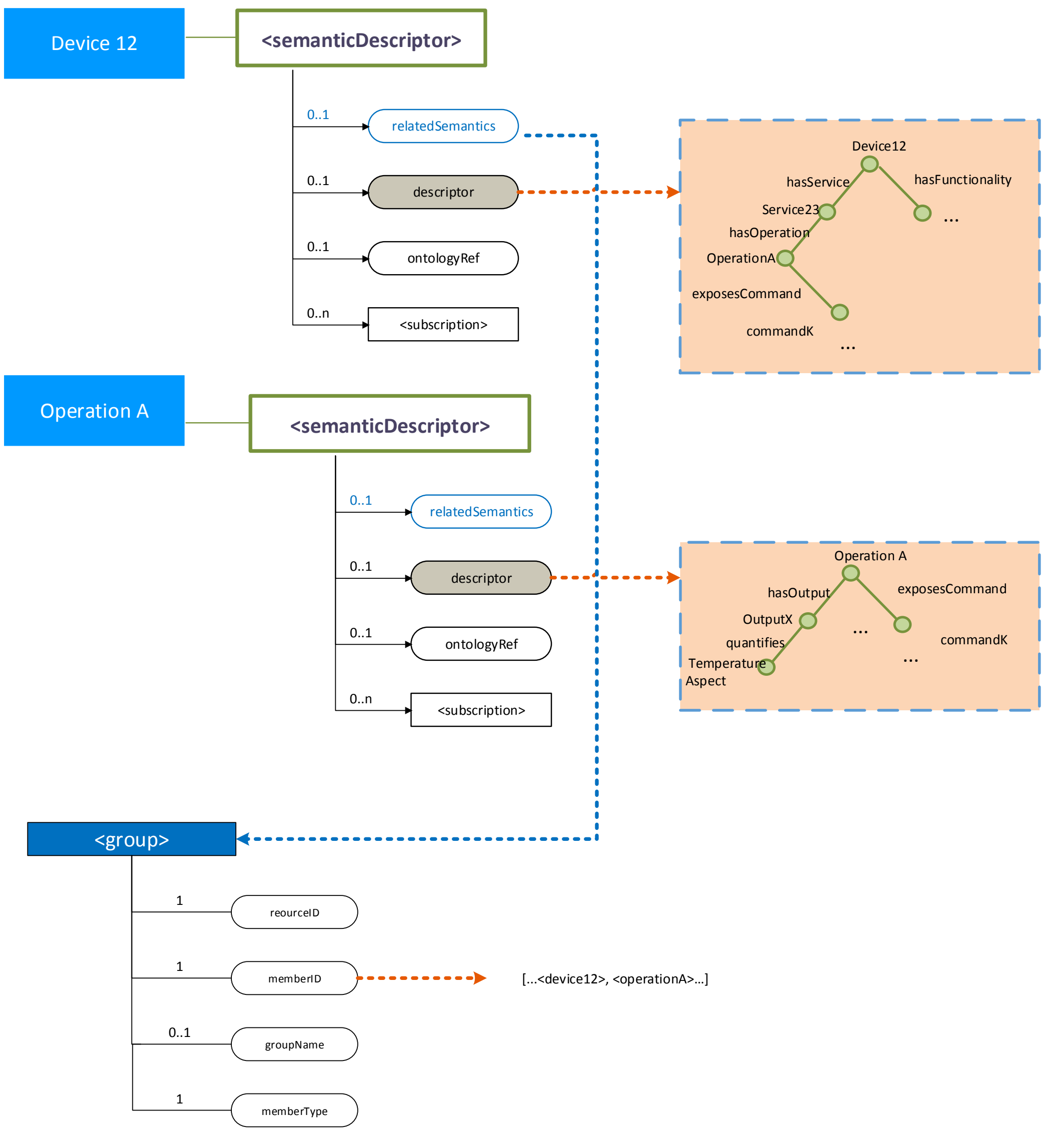


Figure 8.5.5.4.2-1: Use of *relatedSemantics* attribute with a <group> resource

By using the *memberID* attribute, all the descriptors to be used together when performing semantic queries are identified. The SPARQL query engine would use all the links to retrieve the needed descriptors and perform the Query on the overall graph.

For descriptors distributed across CSEs, this is useful when there is only one SPARQL query engine which will process all the descriptors, including on the other CSEs. This means that the process of fetching descriptors from other CSEs is left for implementation and the process of retrieving descriptors from other CSEs is not specified.

When there are individual SPARQL query engines in each CSE, the <*group*> resource may be used as well to target queries to resources belonging to different CSEs by using the <*fanOutPoint*> virtual resource. If the <group> resource residing on a first CSE includes member resources on other CSEs, the <*group*> hosting CSE will forward the RETRIEVE request including the SPARQL query to each CSE containing a group member resource. The individual SPARQL engines on each CSE can process the SPARQL request individually. This means that the SPARQL engine on the <*group*> hosting CSE needs to be able to merge the results before the returning the final query result.

8.5.5.4.3 <semanticGroup> Resource use

In this representation the *relatedSemantics* attribute points to a new <*semanticGroup*> resource shown below, which also includes <Device12> and <operationA> resources.

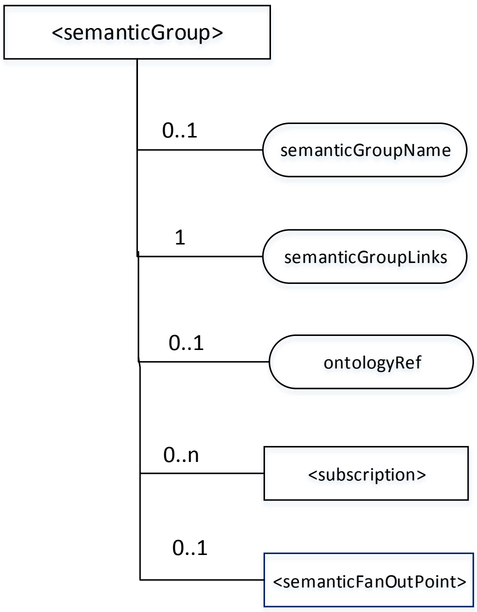


Figure 8.5.5.4.3-1: <semanticGroup> resource

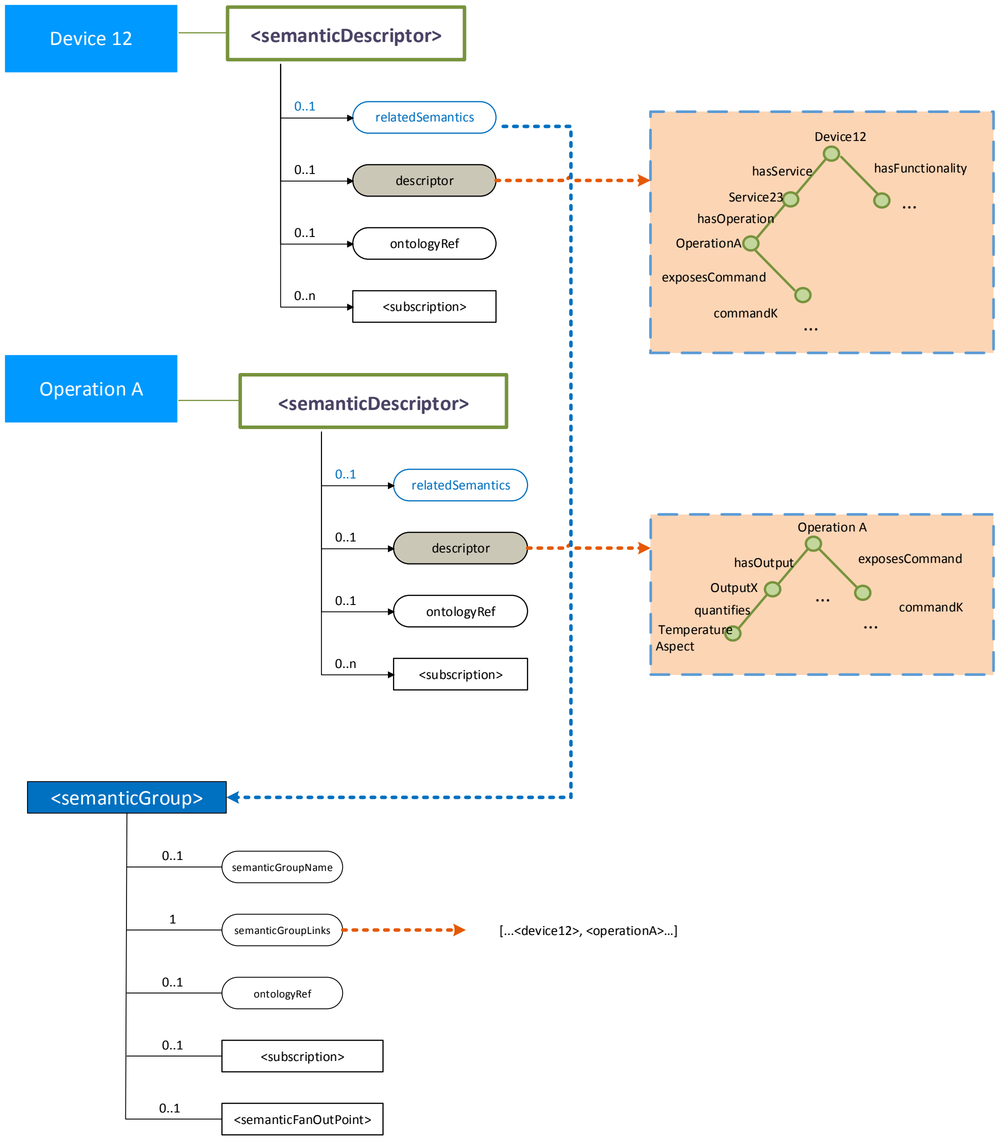


Figure 8.5.5.4.3-2: Use of *relatedSemantics* attribute with a <semanticGroup> resource

The *semanticGroupLinks* attribute of the <*sematicGroup*> resource has the same role as the *memberID* of <*group*>, containing the descriptors to be used together when performing semantic queries are identified. The SPARQL query engine would use all the links to retrieve the needed descriptors and perform the query on the overall graph.

The difference in using the <*semanticGroup*> resource, compared to the <*group*> resource, is in targeting queries to resources belonging to different CSEs using <*semanticFanOutPoint*>. In this case the <*semanticGroup*> hosting CSE will transform the RETRIEVE request including the SPARQL query into RETRIEVE requests for the semantic descriptors on each CSE. Upon return of the results, the SPARQL engine at the <*semanticGroup*> hosting CSE will be

For enabling semantic filtering on semantic descriptions stored across several resources connected by *relatedSemantics* attributes, in either representation, the semantic engine has to be modified in the following way:

When a resource other than <*semanticFanOutPoint*> is targeted:

- The receiver begins processing the request by retrieving the <*semanticDescriptor*> resource of the request target.
- Based on the *relatedSemantics* attribute of the <*semanticDescriptor*> resource targeted, all the related descriptors are discovered, as follows:
  - If the *relatedSemantics* attribute includes a list of links, each of the linked Descriptors are accessed based on the respective access control policies.
  - If the *relatedSemantics* points to a <*group*> resource, the group members from the *memberID* attribute are used and each of their <*semanticDescriptor*>(s) are accessed based on the respective access control policies.
  - If the *relatedSemantics* points to a <*semanticGroup*> resource, the group members from the *semanticGroupLinks* attribute are used and each of their <*semanticDescriptor*>(s) are accessed based on the respective access control policies.
- Once all of the related <*semanticDescriptor*>(s) have been accessed, the content of each of the *descriptor* attributes is added to the content on which the SPARQL request is being executed.
- The full/enlarged content subject to the SPARQL request is provided to the SPARQL engine for processing.

When a <*semanticFanOutPoint*> resource is targeted:

- Based on the *semanticGroupLinks* attribute targeted all the related Descriptors are discovered, and those on the <*semanticGroup*> hosting CSE are retrieved together.
- If there are descriptors stored on a different CSE, individual RETRIEVE requests are sent to each CSE for retrieving the external descriptors.
- All semantic descriptors are accessed based on the respective access control policies.
- Once all of the related <*semanticDescriptor*>(s) have been accessed, the content of each of the *descriptor* attributes is added to the content on which the SPARQL request is being executed.
- The full/enlarged content subject to the SPARQL request is provided to the SPARQL engine for processing.

It may be left to implementation or local policies if, in case the discovered related descriptors specify further *relatedSemantics***,** these are added to the original list and how many levels of indirection may be accommodated. This would enable different implementations and applications to specifically target their goals towards either: expansive and thorough query results with potential runtime costs, or narrower query results optimized for time and/or memory. Other rules which may be specified may include for example handling of the descriptors of child resources e.g. by default all the descriptors attached to children may be considered related (or not, or for a certain number of levels only, etc.).

Advantages of the proposed solution 2:

- The required semantic information can be found.
- Resource-based access control can be easily enforced as the information is being accessed at execution time of the operation and the access privileges of the requester can be applied.

- The content to be processed by the SPARQL engine is collected prior to processing, allowing the use of external, non-oneM2M specific engines.
- Descriptors containing more than one common concept are linked by only one link, allowing for easier avoidance of duplicate content being considered for request processing.

Disadvantages of the proposed solution 2:

- All linked semantic information has to be fetched, even if it is not needed for the execution of the SPARQL query.

Editor's note: It is FFS if/how the processing described using <*semanticGroup*> resource may be implemented using the <*group*> resource.

8.5.5.4.4 SPARQL query result examples for Solution 2

Consider the example in figure 8.5.5.4-1 where semantic descriptions are stored across resources <operationA> and <device12> with the following filter request:

- "Find all devices that have a service that has an operation whose output quantifies a temperature aspect, and filter it for those with the output = OutputX and command = commandK".

The corresponding SPARQL representation of the request is:

```
…
SELECT ?device
WHERE {     ?device rdf:type base:Device .
        ?device   base:hasService   ?service .
        ?service   base:hasOperation   ?operation .
        ?operation   base:hasOutput   ?output .
        ?output   base:quantifies   temp:TemperatureAspect .
?device   base:exposesCommand  ?command .
FILTER (?output == OutputX && ? command == commandK)
}
```

The SPARQL request will be applied across the resultant graph in figure 8.5.5.4-2. The RDF representation below reflects the resultant description, in red for the subgraph retrieved from <Device12>, in blue for the subgraph retrieved from <operationA>.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix oneM2M: <http://oneM2M.org/owl#> .
@prefix ex: <http://example.com/cseBase/> .

<rdf:Description rdf:about ex: Device12>
        <oneM2M:hasService> ex:Service23 </oneM2M:hasService>.
        <oneM2M:hasFunctionality> ex:FuncF </oneM2M:hasFunctionality>
</rdf:Description>
<rdf:Description rdf:about ex: Service23 >
        <oneM2M:hasOperation> ex:OperationA </oneM2M:hasOperation>
</rdf:Description>

<rdf:Description rdf:about ex: OperationA >
        <oneM2M:exposesCommand> ex:CommandK </oneM2M:exposesCommand>
        <oneM2M:hasOutput> ex:OutputX </oneM2M:hasOutput>
</rdf:Description>

<rdf:Description rdf:about ex: OutputX >
        <oneM2M:quantifies> ex:TemperatureAspect </oneM2M:quantifies>
</rdf:Description>
```

Figure 8.5.5.4.4-1: RDF representation of resultant graph

The SPARQL query execution result on <Device12>, using this solution is ex:Device12. The query execution without the ability to link in the <operationA> semantic descriptor content would be empty as the output filter would not be matched.

8.5.6 Semantic Queries and the Use of Semantic Repositories

8.5.6.1 Introduction

Semantic Repositories as introduced in clause 7 are supported by existing technologies and dedicated to the management of semantic annotations. Semantic Repositories may also store new semantic information resulting from reasoning and may support optimized query tools.

Also as discussed in clause 5, Semantic Query is a function that is required in the oneM2M System to support annotation, reasoning, as well as the discovery of resources. Semantic Query uses linked Semantic Triples contained in Semantic Graph Stores (i.e. Semantic Repositories).

In order to support the use of Semantic Repositories for Semantic Query in the oneM2M System, several architectural options are considered in clauses 8.5.6.2, 8.5.6.3 and 8.5.6.4.

8.5.6.2 Semantic Queries with Semantic Descriptors Distributed in Resource Tree(s)

Semantic Filtering and Discovery have been described in the previous section which uses the distributed Semantic Descriptors without Semantic Repositories.

For the purpose of performing Semantic Queries, local and temporary Semantic Graph Stores (i.e. temporary Semantic Repositories) may be employed, as illustrated in figure 8.5.6.2-1.

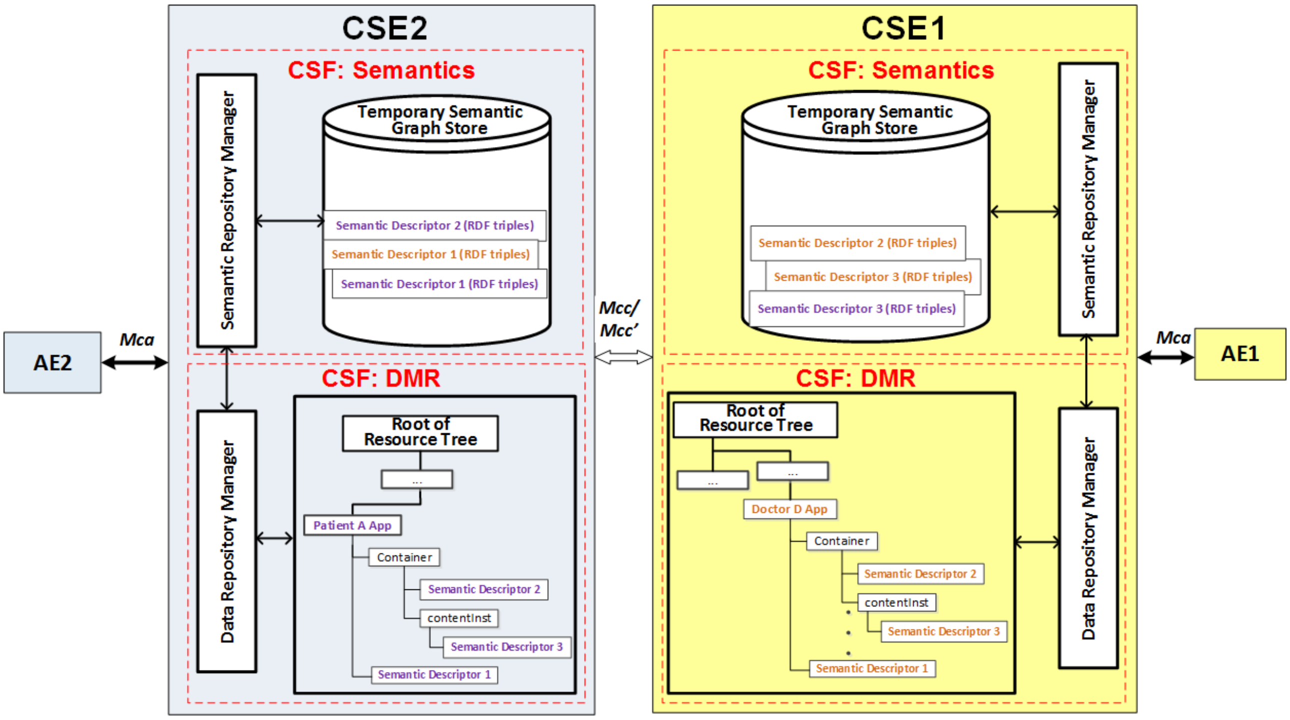
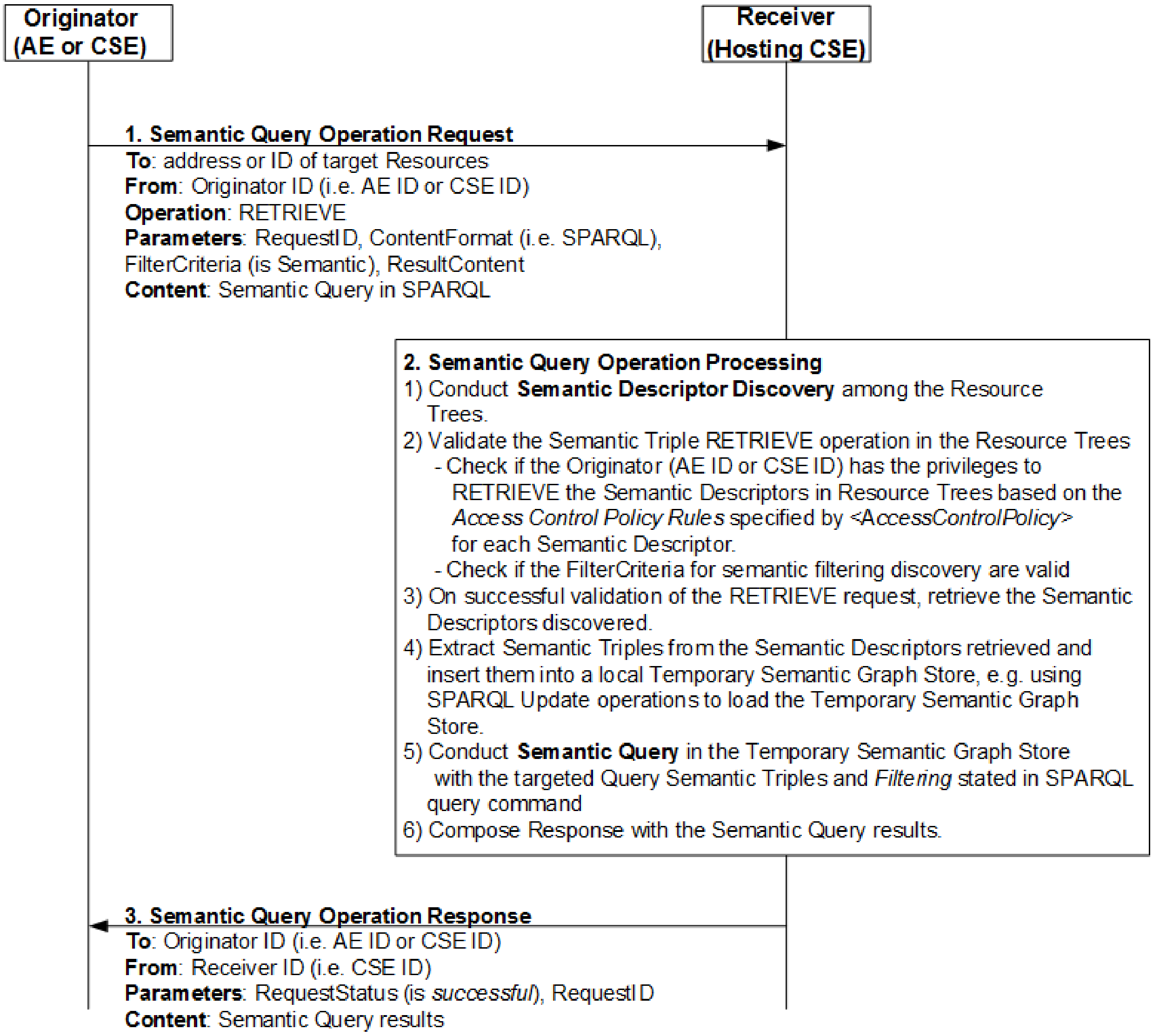


Figure 8.5.6.2-1: Architecture for Distributed Semantic Descriptors with Temporary Semantic Graph Stores

After an initial discovery of Semantic Descriptors in hierarchical Resource Tree(s), the Semantic Triples may be extracted from the Semantic Descriptors discovered and deposited into a local temporary Semantic Graph Store (i.e. a Temporary Semantic Repository). The Semantic Query is then performed on the resultant graph, using the tools supported by the graph store. An example is illustrated in figure 8.5.6.2-2.



NOTE: A Temporary Semantic Graph Store is used to store all the Semantic Triples extracted from the Semantic Descriptors discovered from the Resource Tree(s), i.e. to form the relationship graph for each Semantic Query. It may be cleared for each query or periodically based on local policies.

Figure 8.5.6.2-2: Semantic Query with Distributed Semantic Descriptors and a Temporary Semantic Graph Store

8.5.6.3 Semantic Queries with all Semantic Triples Contained in a Semantic Repository

Editor's Note: Operations related to this approach have not been addressed in the currentTS-0001. The filtering semantic discovery is not applicable to this approach.

The architecture supporting Semantic Queries with all Semantic Triples contained in a Semantic Graph Store is exemplified by figure 8.5.6.3-1. It illustrates the implementation of Semantic Repositories as a linked-data databases available for semantic operations.

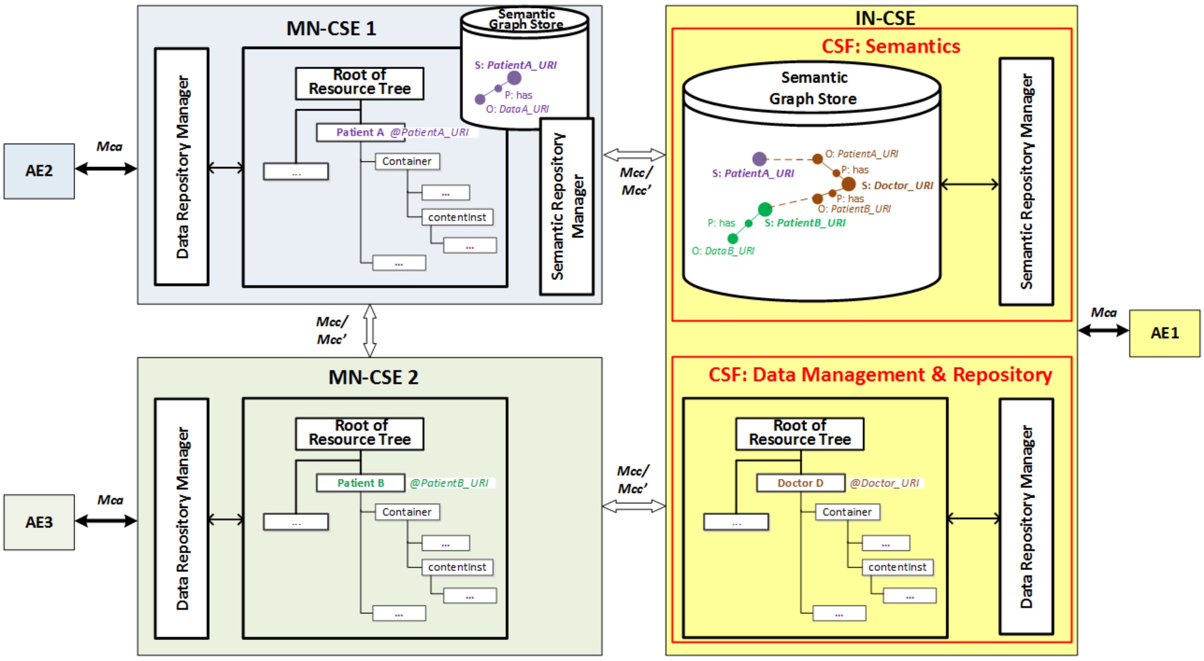


Figure 8.5.6.3-1: Architecture with all Semantic Triples Contained in a Semantic Graph Store

A Semantic Query in the architecture shown in figure 8.5.6.3-1 is illustrated in figure 8.5.6.3-2.

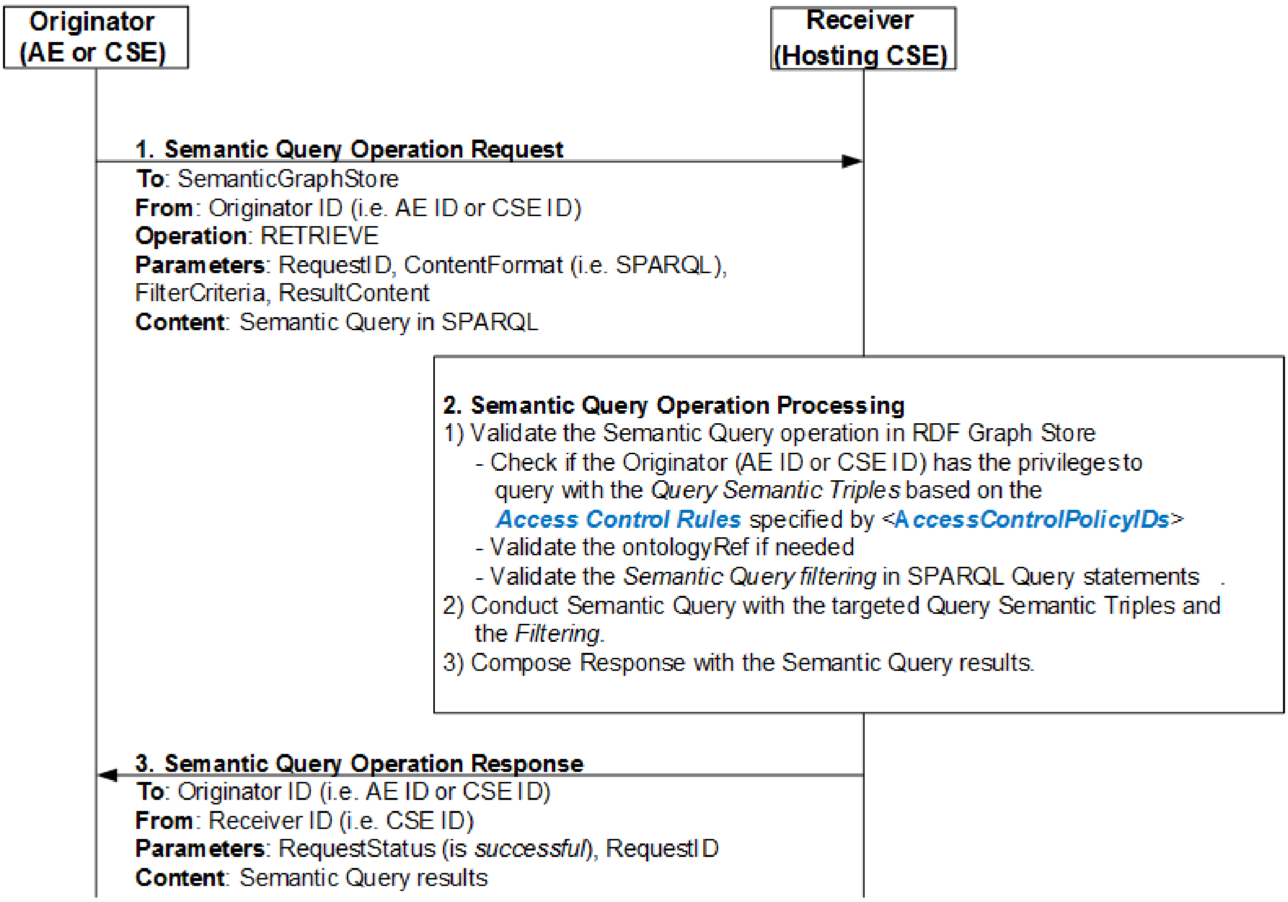


Figure 8.5.6.3-2: Semantic Query with all Semantic Triples Contained in a Semantic Graph Store

8.5.6.4 Semantic Queries with Semantic Descriptors in Resource Tree(s) and Semantic Triples in a Semantic Repository

A hybrid approach is shown in figure 8.5.6.4-1 with the following two variants:

1) Hybrid I - in which all Semantic Descriptors are distributed in Resource Tree(s) for Semantic Resource Discovery and all the Semantic Triples are contained in a Semantic Graph Store for Semantic Query.

2) Hybrid II - in which all the Semantic Descriptors are "virtual resources" as distributed in Resource Tree(s) and any operation to them will trigger the corresponding operation in the Semantic Graph Store which contains all the Semantic Triples.

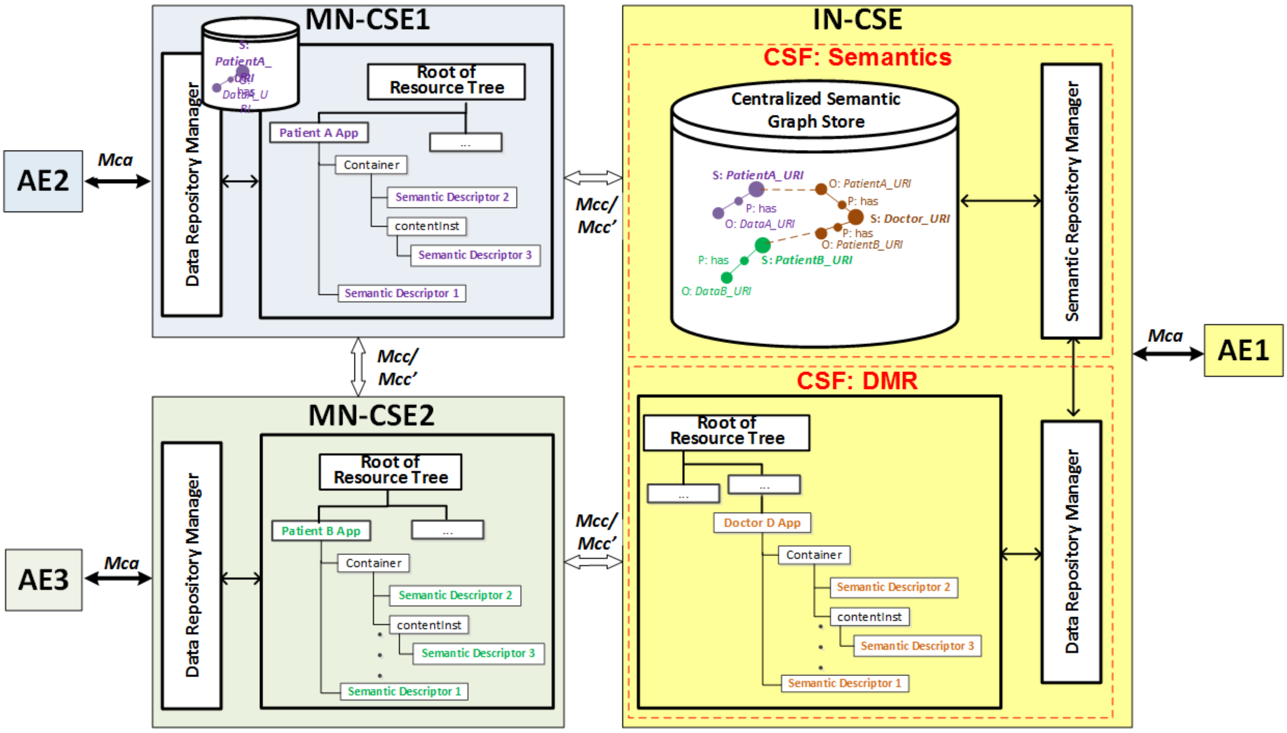


Figure 8.5.6.4-1: Architecture with Semantic Descriptors in Resource Tree(s) and Semantic Triples in a Semantic Repository

An example of a semantic query processing in the hybrid architecture is illustrated in figure 8.5.6.4-2.

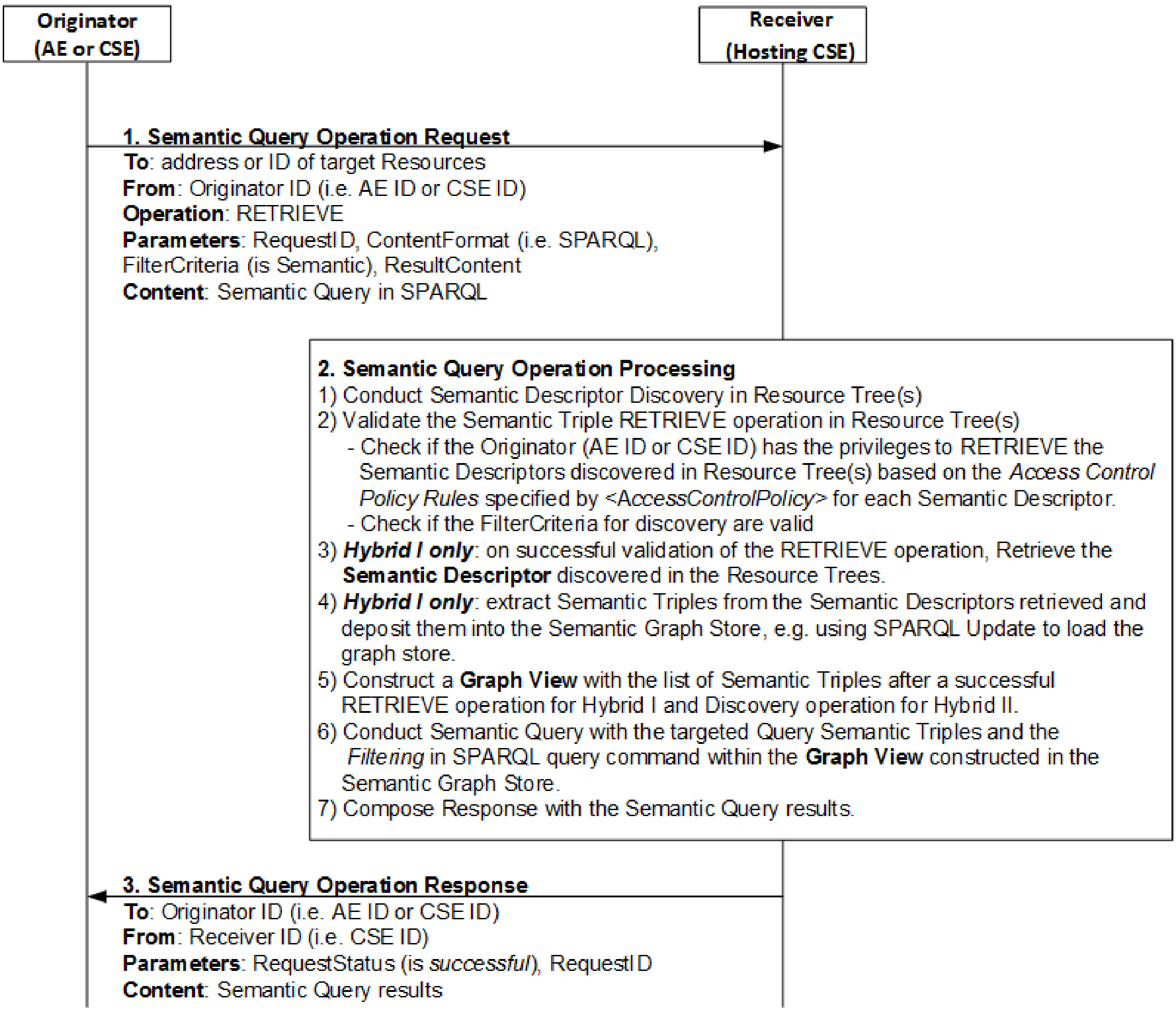


Figure 8.5.6.4-2: Semantic Query with Semantic Descriptors in Resource Tree(s) and Semantic Triples in a Semantic Repository

Editor's note:

1) Hybrid I: why need a permanent Semantic Graph Store? i) May be needed for advanced semantic features such as reasoning, mash-up etc. ii) May be needed for interworking with W3C Semantic Web based applications.
2) Hybrid II: the list of Semantic Triples discovered may not be retrievable due to access control.

8.5.7 Access Control for Semantic Information

8.5.7.1 Access Control for Data Resources and Semantic Triples

8.5.7.1.0 Introduction

Access control for semantic information provided as semantic triples is discussed in this clause. As the efficiency of access control depends also on the underlying implementation, architectural and implementation perspectives are presented.

8.5.7.1.1 Access Control in a Hierarchically Layered Architecture

While the current oneM2M architectural view is exclusively based on resources, these can be implemented in different ways. Especially for managing semantic triples, triple stores are the straightforward choice. This has implications for how the access control to resources and triples is handled. In this clause different implementation options are discussed assuming a hierarchical structuring into a semantic layer and a data layer.

As shown in figures 8.5.7.1.1-1 and 8.5.7.1.1-2, data resources and semantic triples may be integrated in a hierarchically layered architecture with a Data Layer containing the data resources and related functions and a Semantic Layer containing semantic triples and related semantic functions.

The upper layer, i.e. the Data Layer in figure 8.5.7.1.1-1 and the Semantic Layer in figure 8.5.7.1.1-2, controls and manages the Access Control Policies (ACPs). The lower layer, i.e. the Semantic Layer in figure 8.5.7.1.1-1 and the Data Layer in figure 8.5.7.1.1-2, supports the upper layer with semantic graphs or raw data respectively.

The layers may reside on different CSEs, but integration on the same CSE may be more performance efficient.

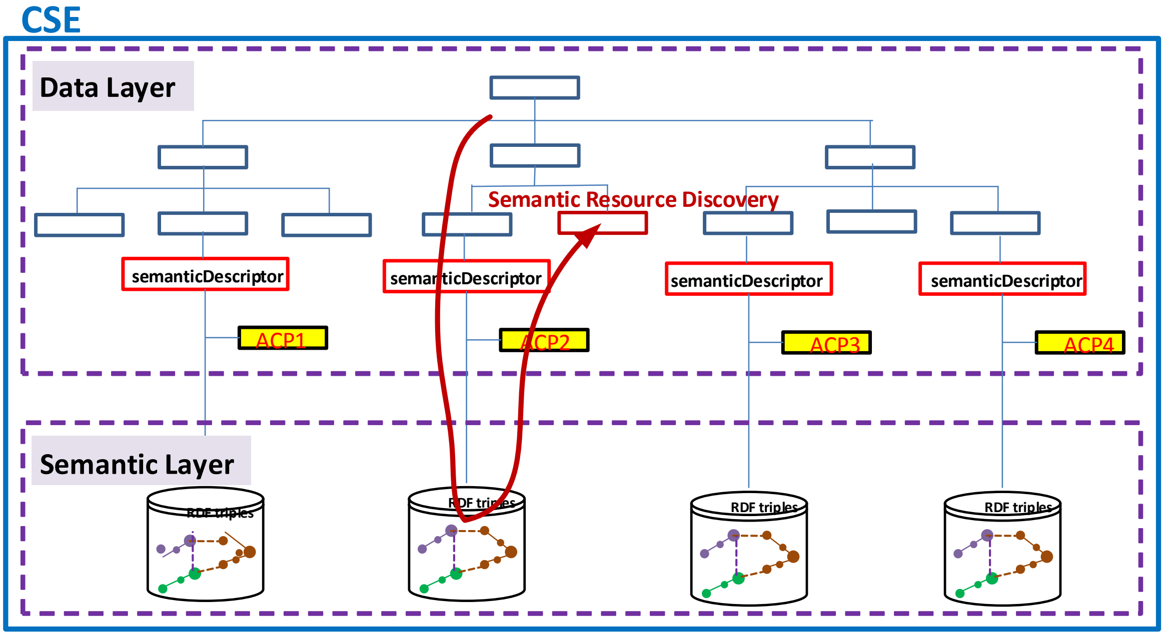


Figure 8.5.7.1.1-1: Access Control in a Hierarchically Layered Structure - Controlled by the Data Layer

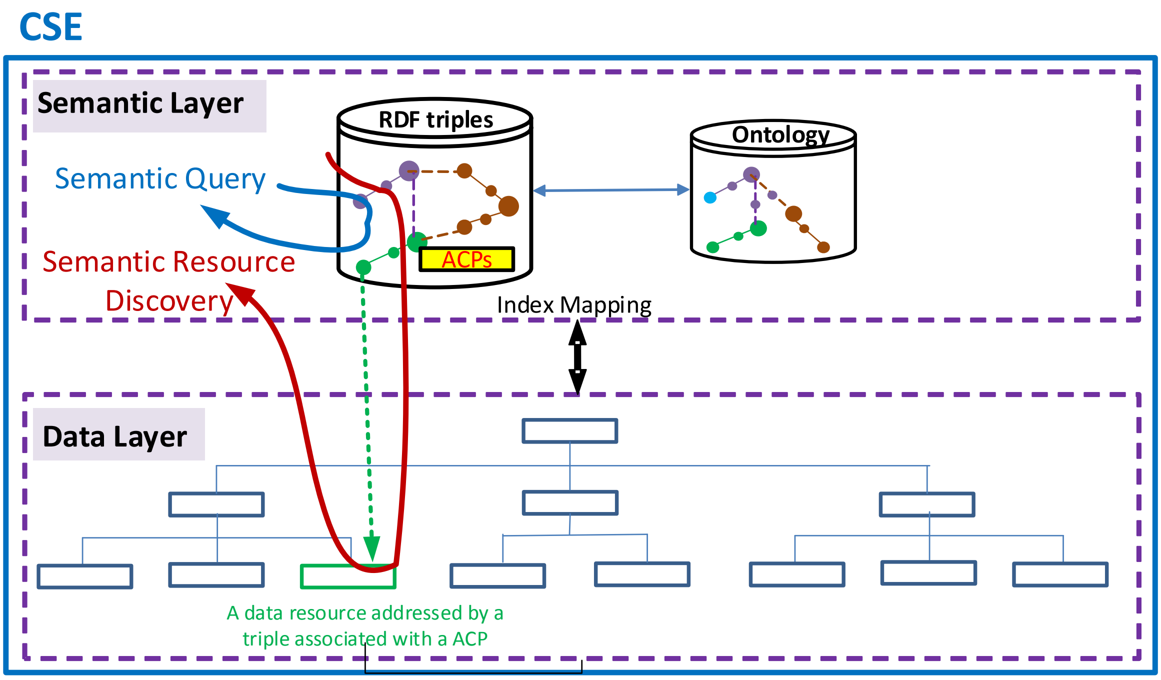


Figure 8.5.7.1.1-2: Access Control in a Hierarchically Layered Structure - Controlled by the Semantic Layer

Figure 8.5.7.1.1-1 shows a typical data resource driven scheme for M2M scenarios. Resource Discovery through the data resource tree may be supported by the semantic leaves (i.e. distributed graph stores) in the Semantic Layer. The ACPs are maintained under <*semanticDescriptor*> resources in the Data Layer.

Figure 8.5.7.1.1-2 shows a typical semantics driven scheme for Semantic Web scenarios. Semantic Query may be conducted in the Semantic Layer with the return of the URI or URL of the data resources in the Data Layer. Semantic Resource Discovery may also be realized with the return of the data resources in the Data Layer via proper mapping between these two layers. The triples in the Semantic Layer are associated with their specific ACPs. A data resource in the Data Layer is addressed by a triple (e.g. via its URI or URL) associated with a ACP.

8.5.7.1.2 Access Control in a Parallel Architecture

Looking beyond the currently supported semantic functionality, a more advanced architecture may be needed. In the following the concepts of Data Entity and Semantic Entity are introduced that may be used in different configurations, supporting more advanced semantic functionality like semantic mash-up and the possible interaction with other semantic platforms like the semantic web.

Figure 8.5.7.1.2-1 shows an exemplary scheme for intelligent IoT scenarios, which has more advanced data and semantics features or functions. As illustrated in figure 8.5.7.1.2-1, a parallel architecture may have a Data Entity and a Semantic Entity.

A Data Entity and/or a Semantic Entity may each have its own Access Control Policies (ACPs) for managing the access control within its scope.

The semantic triples or data resources in the Data Entity may be exposed to the Semantic Entity with specific ACPs associated. For example:

- a Semantic Publication function in the Data Entity may expose semantic triples to the Central Graph Store in Semantic Entity with the corresponding ACPs associated with the triples; or
- a Data Annotation function in the Data Entity may expose data and associated ACPs to the Central Graph Store via the local temporary or caching Relational Data Base Management System (RDBMS) and Semantic Reasoning and Mapping functions in Semantic Entity.

The data resources or semantic triples in the Semantic Entity may also be exposed to the Data Entity with specific ACPs associated. For example:

- a Semantic Mash-up function in the Semantic Entity may expose new data resources and related ACPs from the semantic mash-up to the Data Entity; or
- a Semantic Annotation function in the Semantic Entity may expose semantic triples and related ACPs to the <*semanticDescriptor*> resources in the Data Entity.

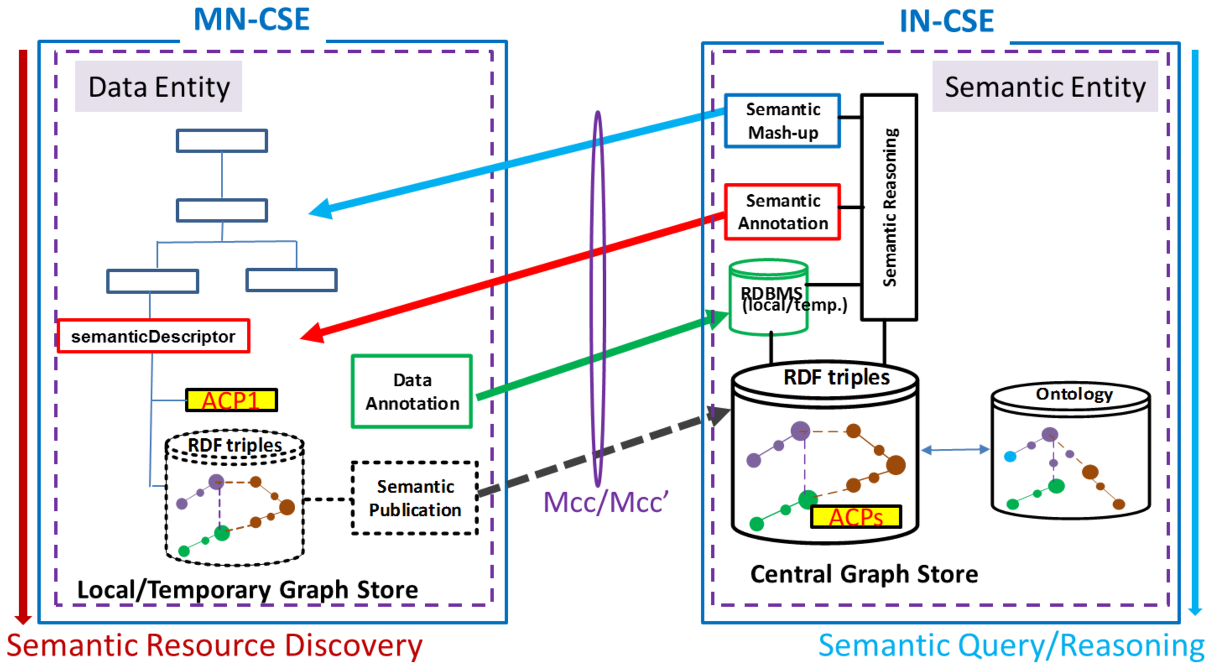


Figure 8.5.7.1.2-1: Access Control in a Parallel Structure

Figure 8.5.7.1.2-1 shows that a Data Entity and a Semantic Entity may reside on different CSEs. But a Data Entity and a Semantic Entity may also reside on the same CSE. Figure 8.5.7.1.2-2 shows a logical resource tree with both a Data Entity and a Semantic Entity.

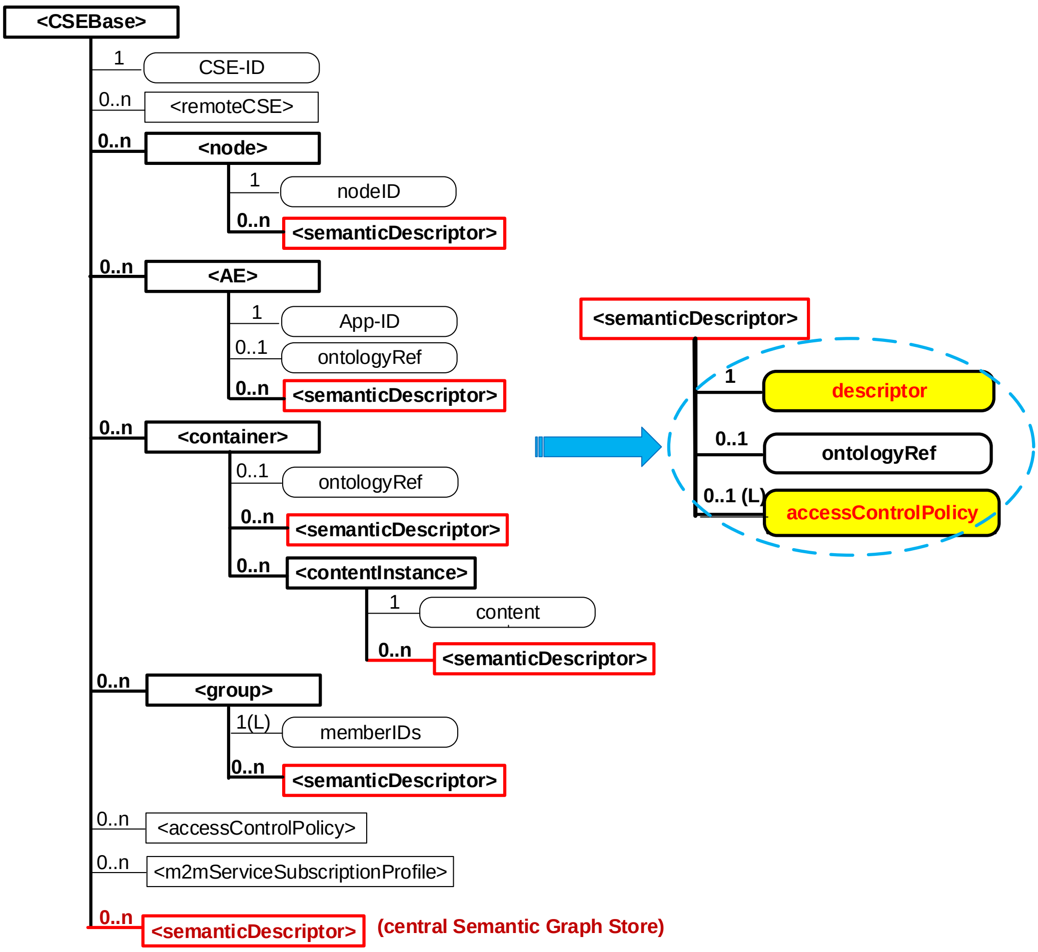


Figure 8.5.7.1.2-2 Logic Resource Tree with both a Data Entity and a Semantic Entity

8.5.7.2 Access control solutions for semantic discovery

8.5.7.2.1 Access Control based on the ACPs in resource tree and triples in the graph store

8.5.7.2.1.0 Introduction

Implementing access control based on the ACPs in resource tree can be seen as an attractive approach since the compatiblity with existing resource tree ACPs will be kept.

8.5.7.2.1.1 Solution A ---- Semantic filtering based on graph store

In the solution descriptions below, some assumptions are considered.

1) There is a centralized graph store to store the triples in all <semanticDescriptor>s.

2) Based on the query request with a target URL in the resource tree, the scope of the query is limited to the triples in both the <semanticDescriptor>s as child resources in the sub-tree under the target URL and the relevant <semanticDescriptor>s linked to the <semanticDescriptor>s under the target URL.

Editor'note : the extensions of this solution for other cases (e.g. distributed graph stores) may be further considered.

In solution A, the triples in the <semanticDescriptor>s will be stored in one graph of a graph store. To retain the effect of ACPs in resource tree, the <semanticDescriptor>is used as the anchor to link the ACPs in resource tree and the access control during the query on semantic repository.The procedure of the solution A are described as follows.

➢ Pre-steps before semantic query process:

01) The CSE hosting the Graph store creates an internal ontology with class *SemanticDescriptor* and *atomDescription*, and the property *describedIn*, *hasSubject* *hasObject* and *hasProperty*

02) For each <semanticDescriptor>with ACP in resource tree(s), the CSE hosting the graph store creates corresponding semantic descriptor instances in the semantic graph store using IRI/URL of the respective <semantic Descriptor>. The semantic descriptor instances are the instances of the predefined class *SemanticDescriptor*.

03) The CSE hosting the graph store adds triples in semantic graph sotre to associate the semantic triples in the <semanticDescriptor>s in resource tree with the created semantic descriptor instances. The triples in the <semanticDescriptor>s in the resource tree(s) of other CSEs should be notified to the CSE hosting the graph store.

Considering that one subject can be described in multiple <semanticDescriptor>s with different ACPs, the association should be implemented with each triple for classification, and the association triples are added based on each triple described in the <semanticDescriptor>. Figure 8.5.7.2.1.1-1 shows the association between the triple and the semantic descriptor instance.

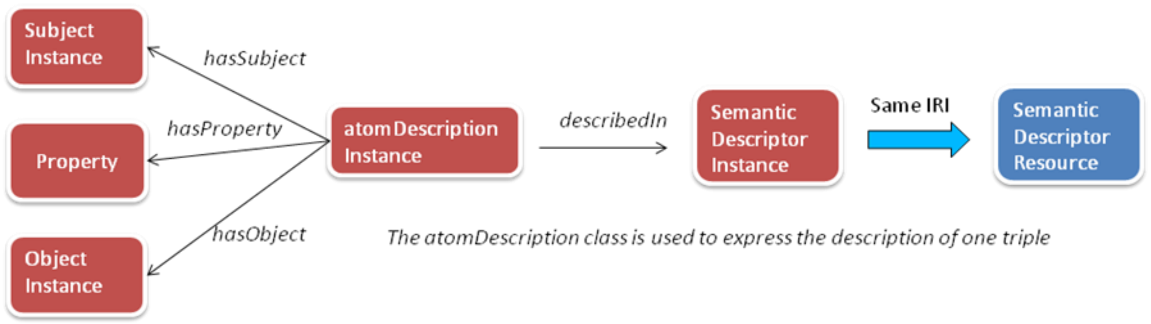


Figure 8.5.7.2.1.1-1. the association between the triple and the *SemanticDescriptor* instance

For example, for the triple in <semanticDescriptor> A as *classX property Y classZ*, the following association triples are needed to be added.

atomDescriptionA hasSubject classX

atomDescriptionA hasObject classX

atomDescriptionA hasproperty classZ

atomDescriptionA describedIn SemanticDescriptorA

➢ The process after receiving the semantic query request with SPARQL statement.

1) The receiver CSE finds the <semanticDescriptor>s where the Originator (AE ID or CSE ID) is allowed to use for querying based on the ACPs in resource tree and the target URL in the request,.

2) The receiver CSE identifies the corresponding *SemanticDescriptor* instances (same IRI/URL with the <semanticDescriptor>) in the semantic graph store.

3) In the received original SPAQRL semantic query statements, the receiver CSE adds new sentences to indicate that the target variable triples are associated with the identified *SemanticDescriptor* instances as follows..

   a) find the variables and theire relevant triples in the SPARQL query

   b) create *atomdescription* variables for each triple with variables in the query,

   c) associate the *atomdescription* variables with each triple with variables in the query

   d) add the sentence to associate the *atomdescription* variables and the identified *SemanticDescriptor* instances.

Figure 8.5.7.2.1.1-2 shows the association between the triple with variables and the *SemanticDescriptor* instances.

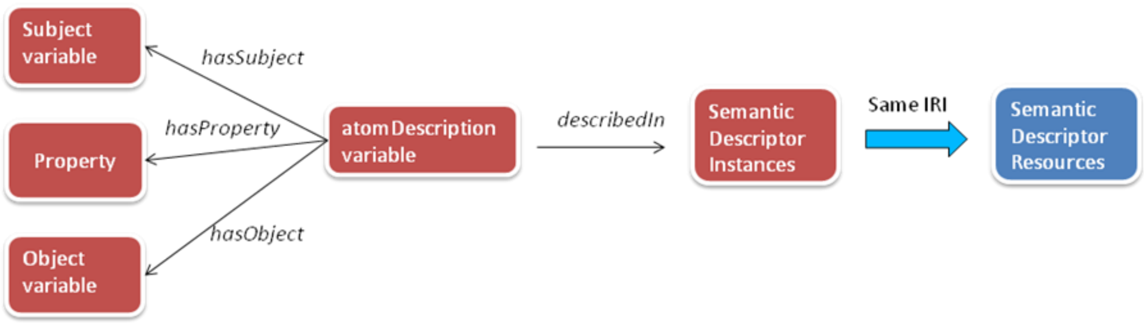


Figure 8.5.7.2.1.1-2 association between the triple with variables in SPARQL query and the *SemanticDescriptor* instances

For example, for the original SPAQRL query

```
SELECT ?device ?operation
WHERE {
?device rdf:type m2m:WashMachine.
?device m2m:hasOperation ?operation.
}
```

If the allowed *SemanticDescriptor* instance is *SemanticDescriptorA*, then the modified SPAQRL query is given as

```
SELECT ?device ?operation
WHERE {
?device rdf:type m2m:WashMachine.
?device m2m:hasOperation ?operation.
?atom1 temp:hasSubject ?device.
?atom1 temp:hasObject ?operation
?atom1 temp:hasProperty m2m:hasOperation
?atom2 temp:hasSubject ?device
?atom2 temp:hasObject m2m:WashMachine
?atom2 temp:hasProperty rdf:type
?atom1  temp:desribedIn SemanticDescriptorA(IRI/URL).
?atom2  temp:desribedIn SemanticDescriptorA(IRI/URL).
}
```

4) The receiver CSE sends the modified SPARQL semantic query statement to the CSE hosting the graph store for querying the graph store.

5) The receiver CSE compose Response according to the semantic query results feedback from the CSE hosting the graph store..

In the following part, we give a complete example to explain solution A. The considered resource tree is shown in Figure 8.5.7.2.1.1-3 where three <semanticDescriptor> (SD) resources are involved.

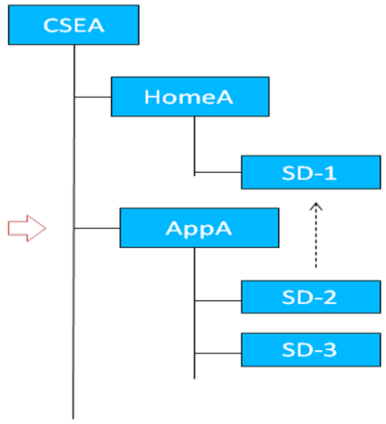


Figure 8.5.7.2.1.1-3. considered resource tree in the example

The triples in SD-1, SD-2 and SD-3 are assumed as follows.

SD-1

HomeA rdf:type ex:Home

HomeA ex:hasLocation LocationA

LocationA ex:hasLatitude "300"

LocationA ex:hasLongitude "200"

SD-2

DeviceA rdf:type m2m:TemperatureSensor

DeviceA ex:hasLocation LocationA

There is a link to SD-1 in SD-2 that indicates Location(class) related information can be trieved from SD-1.

SD-3

DeviceB rdf;type ex:DoorLock

DerviceB ex:hasLocation LocationA

The CSEA hosts the graph store. The triples in the graph store will include all the tripes in the<semanticDescriptor>, and there will be 4 additional triples for each triple in the<semanticDescriptor>. The total additional triples are given as follows.

For SD-1

atom1 temp:hasSubject HomeA

atom1 temp:hasObject ex:Home

atom1 temp:hasProperty rdf:type

atom1 temp:describedIn SD-1

atom2 temp:hasSubject HomeA

atom2 temp:hasObject LocationA

atom2 temp:hasProperty ex:hasLocation

atom2 temp:describedIn SD-1

atom3 temp:hasSubject LocationA

atom3 temp:hasObject “300”

atom3 temp:hasProperty ex:Latitude

atom3 temp:describedIn SD-1

atom4 temp:hasSubject LocationA

atom4 temp:hasObject “200”

atom4 temp:hasProperty ex:hasLongtitude

atom4 temp:describedIn SD-1

For SD-2

atom5 temp:hasSubject DeviceA

atom5 temp:hasObject m2m:TemperatureSensor

atom5 temp:hasProperty rdf:type

atom5 temp:describedIn SD-2

atom6 temp:hasSubject DeviceA

atom6 temp:hasObject LocationA

atom6 temp:hasProperty ex:hasLocation

atom6 temp:describedIn SD-2

For SD-3

atom7 temp:hasSubject DeviceB

atom7 temp:hasObject m2m:DoorLock

atom7 temp:hasProperty rdf:type

atom7 temp:describedIn SD-3

atom8 temp:hasSubject DeviceB

atom8 temp:hasObject LocationA

atom8 temp:hasProperty ex:hasLocation

atom8 temp:describedIn SD-3

It is assumed that an originator sends the query request in which the target URL is the URL of AppA and SPARQL query filter is

SELECT ?device

WHERE {

?device ex:hasLocation ?Location.

?Location ex:hasLatitude ?val1.

?Location ex:hasLagitude ?val2.

FILTER(?val1==”300” && ?val2 ==”200”)

}

When receiving this query request, the CSEA will first identify the scope of the <semanticDescriptor>s related to the query as follows.

1) find that there are two SDs, i.e. SD-2 and SD-3, under the target URL, and then check the ACPs linked to these SDs and find that only SD-2 is allowed to be used by the originator in the query.

2) find that there is a link (e.g. relatedSemantics attribute of <semanticDescriptor>) in the SD-2 to SD-1 for the class Location, and then check the original SPARQL query and find that class Location is involved.

3) check the ACP linked to SD-1 and find that SD-1 is not allowed to be used by the originator in the query.

After identifying the scope of the <semanticDescriptor>s (SD_2) related to the query, the CSEA revises the original SPARQL query as

SELECT ?device

WHERE {

?device ex:hasLocation ?Location.

?Location ex:hasLatitude ?val1.

?Location ex:hasLagitude ?val2.

?agtom1 temp:hasSubject ?device.

?atom1 temp:hasObject ?Location.

?atom1 temp:hasProperty ?hasLocation.

?atom2 temp:hasSubject ?Location.

?atom2 temp:hasObject ?val1.

?atom2 temp:hasProperty ex:hasLatitude.

?atom3 temp:hasSubject ?Location.

?atom3 temp:hasObject ?val2.

?atom3 temp:hasProperty ex:hasLongitude

?atom1 temp:describedIn SD-2

?atom2 temp:describedIn SD-2

?atom3 temp:describedIn SD-2

FILTER(?val1==”300” && ?val2 ==”200”)

}

The CSEA applies the revised SPARQL query to the graph store, and return the result.

The returned result is none device.

8.5.7.2.1.2 Solution B ----- Graph division based semantic filtering

In the solution descriptions below, some assumptions are considered.

1) There is a centralized graph store to store the triples in all <semanticDescriptor>s.

2) The returned information for query request is in the scope of the triples in all possible <semanticDescriptor>s. The scope may contain more triples (i.e. from <semanticDescriptor>s not explicitely linked to the <semanticDescriptor>s under the original target URL) compared with solution A.

Editor'note : the extensions of this solution for other cases (e.g. distributed graph stores) may be further considered.

In solution B, the triples in the <semanticDescriptor>s will be stored in separated graphs in the graph store. There are two options for the graph division.

- Option 1

  Store the triples in the same <semanticDescriptor> in a graph.

- Option 2

  Store the triples in the semantic descriptors linked to the same ACP in a graph

For Option 1, there will be many graphs in the graph store and the query speed will be slow when the query across the union of a lot of graphs, but each graph of Option 1 is not necessary to be updated with the update of ACPs. For Option 2, the number of graphs will be small but it needs the synchronization between the graphs and the ACPs.

The procedure of solution B is simple.

➢ Pre-step

  01) The CSE hosting the graph store stores the triples in the separated graphs of graph store as Option 1 or Option 2.

  02) The ACP information related to <semanticDescriptor>s in the resource tree(s) of other CSEs should be notified to the CSE hosting the graph store.

➢ Query-step

The query request will be forwarded from the receiver CSE to the CSE hosting the graph store for query.

For Option 1:

After receiving the forwarded request, the CSE hosting the graph store will

1) Find the <semanticDescriptor>which is allowed to be used in the query according to the ACP.
2) Identify the graphs that corresponds to the found <semanticDescriptor> in the previous step.
3) Apply the SPARQL query on the union of the identified graphs
4) Return the query result

For Option 2:

After receiving the forwarded request, the CSE hosting the graph store will

1) Identify the access permissions of the Originator of the original request before forwarding – according to ACPs, and find the relevant ACPs that includes the Originator to have discovery permissions.
2) Identify the graphs that corresponds to the found ACPs in the previous step.
3) Apply the SPARQL query on the union of the identified graphs.
4) Return the query results

8.5.7.3 Direct Access Control of Semantic Graph Store for Semantic Queries

➢ Approach Description

As described in previous sub-clauses 8.5.7.1 and 8.5.7.2, <accessControlPolicy> specified by the accessControlPolicyIDs attribute of <semanticDescriptor> may be used for access control in the Semantic Graph Store when executing SPARQL operations as a part of semantic queries. One approach is to implement access control policies directly in the Semantic Graph Store, which makes it more efficient and scalable to control the access to a centralized Semantic Graph Store -. This approach may contain the following main steps with the following assumptions: 1) There is a centralized Semantic Graph Store; 2) The proposed approach will be used for semantic query over this centralized Semantic Graph Store; 3) Each semantic query matches all semanticDescriptors but not cross multiple semanticDescriptors; 4) There is a need for synchronization of access control policies between the CSE and the Semantic Graph Store.

1. Construct *Access Control Rules* specified by <*accessControlPolicy*> in the Semantic Graph Store. Note that <*accessControlPolicy*> is specified by the accessControlPolicyIDs attribute of <semanticDescriptor> resource.
2. Associate targeted *Semantic Triples* (i.e. RDF triples as described by the descriptor attribute of <semanticDescriptor> but stored in the Semantic Graph Store) with their a*ccessControlPolicyIDs* or <*accessControlPolicy*> with related *Access Control Rules*.
3. Semantic triple operations are conducted with the selected semantic triples which are associated with the *Access Control Rules* allowing the Originator to operate.

Figure 8.5.7.3-1 below gives an example of access control policy for two <semanticDescriptor> resources, where there are two access control policies (i.e. <accessControlPolicy1> and <accessControlPolicy2>). The access to <semanticDescriptor1> is controlled by <accessControlPolicy1> and <accessControlPolicy2>, while the access to <semanticDescriptor2> is only controlled by <accessControlPolicy2>.

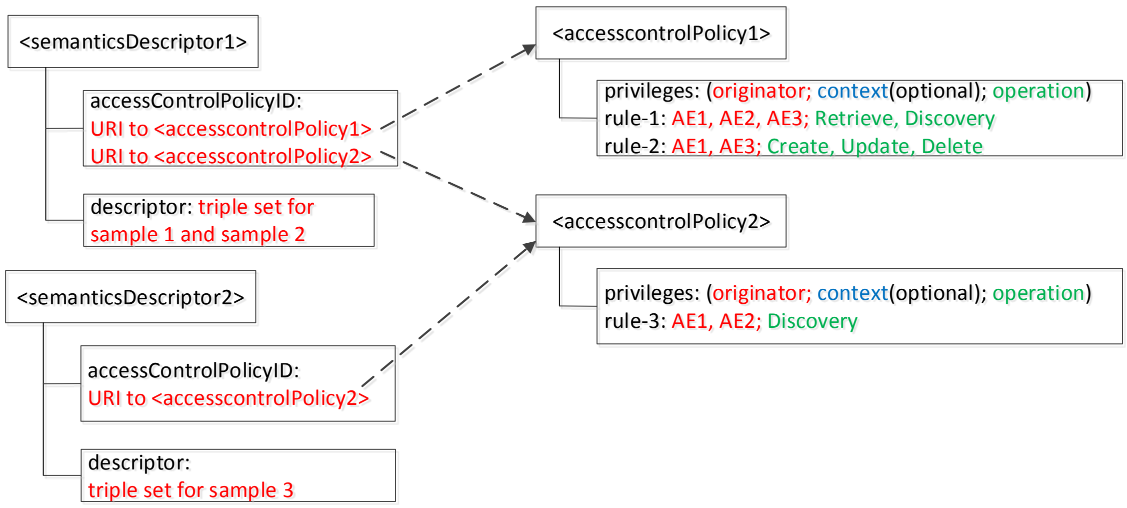


Figure 8.5.7.3-1 Example of Access Control Policy for <semanticDescriptor>

- Access Control Modelling in Semantic Graph Store

Examples for access control modelling in the Semantic Graph Store are shown in figure 8.5.7.3-2, figure 8.5.7.3-3, figure 8.5.7.3-4, and figure 8.5.7.3-5.

The **Access Control Ontology** shown in figure 8.5.7.3-2 defines two new classes: accessControlPolicy and accessControlRule. In addition, five new properties (i.e. hasACPRule, hasACOriginator, hasACOperations, hasACContexts and appliedTo) are defined. hasACPRule is used to link an accessControlPolicy instance with an accessControlRule instance. Properties hasACOriginator, hasACOperations and hasACContexts (optional) basically describe an accessControlRule instance and are used to specify who can issue what operations under which conditions. Property appliedTo is used to describe which <semanticDescriptor> resource an accessControlPolicy instance can be applied to (i.e., bind <accessControlPolicy> and <semanticDescriptor>). Note that this ontology is defined by following how oneM2M <accessControlPolicy> resource is specified in oneM2M TS-0001, where an access-control-rule-tuple consists of parameters such as accessControlOriginators, accessControlOperations, and accessControlContexts.

```
Access Control Ontology

@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd:  <http://www.w3.org/2001/XMLSchema#> .
@prefix acp: <http://accessControlPolicy.org/>.
@prefix ex: <http://example.org/> .
@prefix m2m: <http://oneM2M.org/> .

acp:accessControlPolicy rdf:type rdfs:Class .
acp:accessControlRule rdf:type rdfs:Class .

acp:hasACPRule          rdf:type          rdf:Property ;
                        rdfs:domain       acp:accessControlPolicy ;
                        rdfs:range        acp:accessControlRule .

acp:hasACOriginator     rdf:type          rdf:Property ;            Access Control Triples
                        rdfs:domain       acp:accessControlRule ;
                        rdfs:range        m2m:AE_ID, m2m:CSE_ID, xsd:anyURI .

acp:hasACContexts       rdf:type          rdf:Property ;
                        rdfs:domain       acp:accessControlRule ;
                        rdfs:range        m2m:ipv4, m2m:ipv6, m2m:contryCode, rdfs:Literal.

acp:hasACOperations     rdf:type          rdf:Property ;
                        rdfs:domain       acp:accessControlRule ;
                        rdfs:range        m2m:accessControlOperations, rdfs:Literal .

acp:appliedTo           rdf:type          rdf:property ;
                        rdfs:domain       acp:accessControlPolicy ;
                        rdfs:range        xsd:anyURI, rdfs:Literal, m2m:ID, ex:resourceGroup .
```

Figure 8.5.7.3-2 Access Control Ontology Model

Figure 8.5.7.3-3 show an example of eHealth Ontology Reference Model, which will be used to develop the Semantic Graph Store example in figure 8.5.7.3-4 and SPARQL example in figure 8.5.7.3-5.

```
eHealthcare Ontology Reference Model

@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd:  <http://www.w3.org/2001/XMLSchema#> .
@prefix ex: <http://example.org/> .
@prefix acp: <http://accessControlPolicy.org/>.

ex:Person         a rdfs:Class .

ex:dateOfBirth    a rdf:Property ; rdfs:domain ex:Person ; rdfs:range xsd:date ; rdfs:comment "Date of Birth" .

ex:name           a rdf:Property ; rdfs:domain ex:Person ; rdfs:range rdfs:Literal ; rdfs:comment "name of the person" .

ex:Patient  rdfs:subClassOf ex:Person .
ex:Doctor   rdfs:subClassOf ex:Person .

ex:takeCareOf     a rdf:Property ; rdfs:domain ex:Doctor ; rdfs:range rdfs:Patient ;
                  rdfs:comment "doctor take care of (relation) patient" .

ex:MeasurementSample a rdfs:Class .

ex:measureOn      a rdf:Property ;  rdfs:domain ex:MeasurementSample ; rdfs:range xsd:date ;
                  rdfs:comment "the date of measurement" .

ex:measureFor     a rdf:Property ;  rdfs:domain ex:MeasurementSample ; rdfs:range ex:Patient ;
                  rdfs:comment "sample is measure for which patient" .

ex:unit           a rdf:Property ;  rdfs:domain ex:MeasurementSample ; rdfs:range rdfs:Literal ;
                  rdfs:comment "unit of the value" .

ex:BPMeasurementSample rdfs:subClassOf ex:MeasurementSample .

ex:dValue         a rdf:Property ;  rdfs:domain ex:BPMeasurementSample ; rdfs:range xsd:integer ;
                  rdfs:comment "value of the diastolic" .

ex:sValue         a rdf:Property ;  rdfs:domain ex:BPMeasurementSample ; rdfs:range xsd:integer ;
                  rdfs:comment "value of the systolic" .

ex:resourceGroup  a rdf:Class ;
                  rdfs:comment "contain a list of resources in resource tree" .

ex:containMeasurement   a rdf:Property ; rdfs:domain ex:resourceGroup ; rdfs:range ex:MeasurementSample ;
                        rdfs:comment "resourceGroup contains one or more measurement samples" .
```

Figure 8.5.7.3-3 eHealth Ontology Reference Model

Figure 8.5.7.3-4 describes an example of RDF triples in the Semantic Graph Store based on the example shown in figure 8.5.7.3-1 and the Access Control Ontology defined in figure 8.5.7.3-2. In this use case, there are two patients Jack and Alice; their doctors are John and Steve, respectively. There are three blood pressure meansurement samples (i.e. Sample1 for Jack, Sample2 and Sample3 for another patient3). Corresponding triples are shown in black text in figure 8.5.7.3-4, which are generated based on the eHealth Ontology Reference Model in figure 8.5.7.3-3.

The triples in red text in figure 8.5.7.3-4 are added for access control purpose according to the proposed Access Control Ontology model in figure 8.5.7.3-2, when new ACPs are created or updated. In this example, it is assumed two access control polices be created. First, two <semanticDescriptor> are described (i.e. semanticDescriptor1 contains Sample1 and Sample2, while semanticDescriptor2 contains Sample3. Then, two access control policies are defined (i.e. accessControlPolicy1 is applied to semanticDescriptor1, while accessControlPolicy2 is applied to both semanticDescriptor1 and semanticDescriptor2). Next, the detail Access Control Rules for accessControlPolicy1 and accessControlPolicy2 are described.

- accessControlPolicy1 has two accessControlRules, which states that 1) AE-ID-1, AE-ID-2, and AE-ID-3 can RETRIEVE and DISCOVER triples in the semanticDescriptor which accessControlPolicy1 is applied to (i.e. semanticDescriptor1); 2) AE-ID-1 and AE-ID-3 can CREATE, UPDATE, or DELETE triples in the semanticDescriptor which accessControlPolicy1 is applied to (i.e. semanticDescriptor1).
- For accessControlPolicy2, only one accessControlRule is defined; this accessControlRule states that AE-ID-1 and AE-ID-2 can DISCOVER triples in the semanticDescriptor which accessControlPolicy2 is applied to (i.e. semanticDescriptor1 and semanticDescriptor2).

```
eHealth Semantic Graph Store

@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd:  <http://www.w3.org/2001/XMLSchema#> .
@prefix ex: <http://example.org/> .
@prefix acp: <http://accessControlPolicy.org/>.

ex:Patient1 a ex:Patient ; ex:name "Jack" ; ex:dateOfBirth  "2000-08-03"^^xsd:date .
ex:Patient2 a ex:Patient ; ex:name "Alice" ; ex:dateOfBirth  "1998-06-03"^^xsd:date .

ex:Doctor1 a ex:Doctor ; ex:name "John" ; ex:dateOfBirth  "1944-08-21"^^xsd:date ; ex:takeCareOf
ex:Patient1 .
ex:Doctor2 a ex:Doctor ; ex:name "Steve" ; ex:dateOfBirth  "1947-02-11"^^xsd:date ; ex:takeCareOf
ex:Patient2 .

ex:Sample1 a ex:BPMeasurementSample ;
      ex:measureOn "2014-08-21"^^xsd:date ; ex:measureFor ex:Patient1 ;
      ex:unit "mmHg" ; ex:sValue "150"^^xsd:integer ; ex:dValue "100"^^xsd:integer .
ex:Sample2 a ex:BPMeasurementSample ;
      ex:measureOn "2014-07-24"^^xsd:date ; ex:measureFor ex:Patient3 ;
      ex:unit "mmHg" ; ex:sValue "140"^^xsd:integer ; ex:dValue "96"^^xsd:integer .
ex:Sample3 a ex:BPMeasurementSample ;
      ex:measureOn "2012-11-24"^^xsd:date ; ex:measureFor ex:Patient3 ;
      ex:unit "mmHg" ; ex:sValue "130"^^xsd:integer ; ex:dValue "57"^^xsd:integer .

## Below are triples associating measurement samples with corresponding access control policy
ex:semanticsDescriptor1 a ex:resourceGroup ; ex:containMeasurement ex:Sample1, ex:Sample2 .
ex:semanticsDescriptor2 a ex:resourceGroup ; ex:containMeasurement ex:Sample3 .

acp:accessControlPolicy1 acp:appliedTo ex:semanticsDescriptor1 .
acp:accessControlPolicy2 acp:appliedTo ex:semanticsDescriptor1 .
acp:accessControlPolicy2 acp:appliedTo ex:semanticsDescriptor2 .

## Below are triples created for access control policy 1 resource based on access control ontology
acp:accessControlRule1_1 rdf:type acp:accessControlRule.
acp:accessControlRule1_2 rdf:type acp:accessControlRule.
acp:accessControlPolicy1 rdf:type acp:accessControlPolicy.
acp:accessControlPolicy1 acp:hasACPRule acp:accessControlRule1_1, acp:accessControlRule1_2 .

acp:accessControlRule1_1 acp:hasACOriginator "AE-ID-1", "AE-ID-2", "AE-ID-3" .
acp:accessControlRule1_1 acp:hasACOperations "RETRIEVE", "DISCOVERY" .

acp:accessControlRule1_2 acp:hasACOriginator "AE-ID-1", "AE-ID-3" .
acp:accessControlRule1_2 acp:hasACOperations "CREATE", "UPDATE", "DELETE" .

## Below are triples created for access control policy 2 resource based on access control ontology
acp:accessControlRule2_1 rdf:type acp:accessControlRule.
acp:accessControlPolicy2 rdf:type acp:accessControlPolicy.
acp:accessControlPolicy2 acp:hasACPRule acp:accessControlRule2_1 .

acp:accessControlRule2_1 acp:hasACOriginator "AE-ID-1", "AE-ID-2" .
acp:accessControlRule2_1 acp:hasACOperations "DISCOVERY" .
```

Figure 8.5.7.3-4 eHealth Triples in Semantic Graph Store

- Examples of SPARQL Query Procedure

When the CSE receives SPARQL query from Originator, it will add the access control related patterns according to the ID of the Originator and the request operation of the query into the received SPARQL statement, and use the revised SPARQL statement to make query on the semantic graph store.

For example, in the scenario of the example in figure 8.5.7.3-4, when AE-ID-3 sends the following SPARQL query request to the CSE,

```
select distinct ?sample ?sValue ?dValue
where
{
  ?sample rdf:type ex:BPMeasurementSample .
```

```
  ?sample ex:sValue ?sValue .

  ?sample ex:dValue ?dValue .

}
```

The CSE will add some access control related statements according to the ID (i.e. AE-ID-3) of the Originator and the request operation (i.e. DISCOVERY) of the query, the revised SPARQL query can be given as:

```
select distinct ?sample ?sValue ?dValue

where

{

  ?accessControlRule acp:hasACOriginator "AE-ID-3" .

  ?accessControlRule acp:hasACOperations "DISCOVERY" .

  ?accessControlPolicy acp:hasACPRule ?accessControlRule .

  ?accessControlPolicy acp:appliedTo ?semanticDescriptor .

  ?semanticDescriptor ex:containResource ?sample .

 ?sample rdf:type ex:BPMeasurementSample .

  ?sample ex:sValue ?sValue .

  ?sample ex:dValue ?dValue .

}
```

Alternatively, using the approach which is described in the sub-clause of 8.5.7.2 for the association between the semantic descriptor and semantic triples, the revised SPARQL query can be given as:

```
select distinct ?sample ?sValue ?dValue

where

{

  ?accessControlRule acp:hasACOriginator "AE-ID-3" .

  ?accessControlRule acp:hasACOperations "DISCOVERY" .

  ?accessControlPolicy acp:hasACPRule ?accessControlRule .

  ?accessControlPolicy acp:appliedTo ?semanticDescriptor .

  ?atom1 temp:describedIn ?semanticDescriptor.

  ?atom1 temp:hasSubject ?sample.

  ?atom1 temp:hasObject ?sValue.

  ?atom1 temp:hasProperty ex:sValue.

  ?atom2 temp:describedIn  ?semanticDescriptor.

  ?atom2 temp:hasSubject ?sample.

  ?atom2 temp:hasObject ?dValue.

  ?atom2 temp:hasProperty ex:dValue.

  ?atom3 temp:describedIn  ?semanticDescriptor.

  ?atom3 temp:hasSubject ?sample.
```

?atom3 temp:hasObject ex:BPMeasurementSample.

?atom3 temp:hasProperty rdf:type.

}

Figure 8.5.7.3-5 shows the SPARQL query result in the above example over the eHealth Semantic Graph Store in figure 8.5.7.3-4. According to the access control triples added to the Semantic Graph Store (i.e. red text in figure 8.5.7.3-4), AE-ID-3 is only allowed to DISCOVER samples included in semanticDescriptor1 (i.e. Sample1 and Sample2). As a result, the returned result for SPAQRL query in figure 8.5.7.3-5 presents the selected content of Sample1 and Sample2.

Figure 8.5.7.3-5 Example for eHealth Semantic Query Result with Access Control

8.5.7.4 Access Control Using Temporary Semantic Graph Stores

This section describes an implementation approach that can be used for implementing the semantic features supported in Release 2 of oneM2M. <semanticDescriptor> resources have been introduced to be able to semantically annotate oneM2M resources of certain resource types, including AE, Container, ContentInstance, FlexContainer and more. Semantic functionalities have been added for filtering and selectively updating the semantic information stored in the descriptor attribute of the <semanticDescriptor> , which is represented in the form of RDF triples. The SPARQL language has been selected as the most suitable for specifying the filter and selective update operations.

For implementing the SPARQL-based semantic functionality, a SPARQL-engine is needed and such engines are typically provided on top of semantic graph stores. Thus, it is a straight-forward implementation choice to use such a semantic graph store.

<semanticDescriptor> resources as any other oneM2M resources have associated access policies which determine whether a requester is allowed to access the content of the resource. These access policies also apply for the semantic description stored in a <semanticDescriptor> resource and have to be adhered to when executing SPARQL requests on and across semantic information contained in these <semanticDescriptor> resources. So even when storing the semantic information in semantic graph stores and accessing them using SPARQL requests, the access control policies have to be applied. Approaches for translating and applying access control policies within the semantic graph store are described in other Subsections of Section 8.5.7.

In this subsection, we show how the SPARQL-based functionality needed for oneM2M Release 2 can be implemented using temporary semantic graph stores. For an incoming SPARQL request, the access control policies are applied when accessing the relevant <semanticDescriptor> resource(s) for populating the temporary semantic graph store. Once this has happened, the SPARQL query can be executed without further access control checks.

Figure 8.5.7.4-1 shows the different steps in the case of a single <semanticDescriptor> resource being accessed to enable the execution of a SPARQL request on its semantic content.

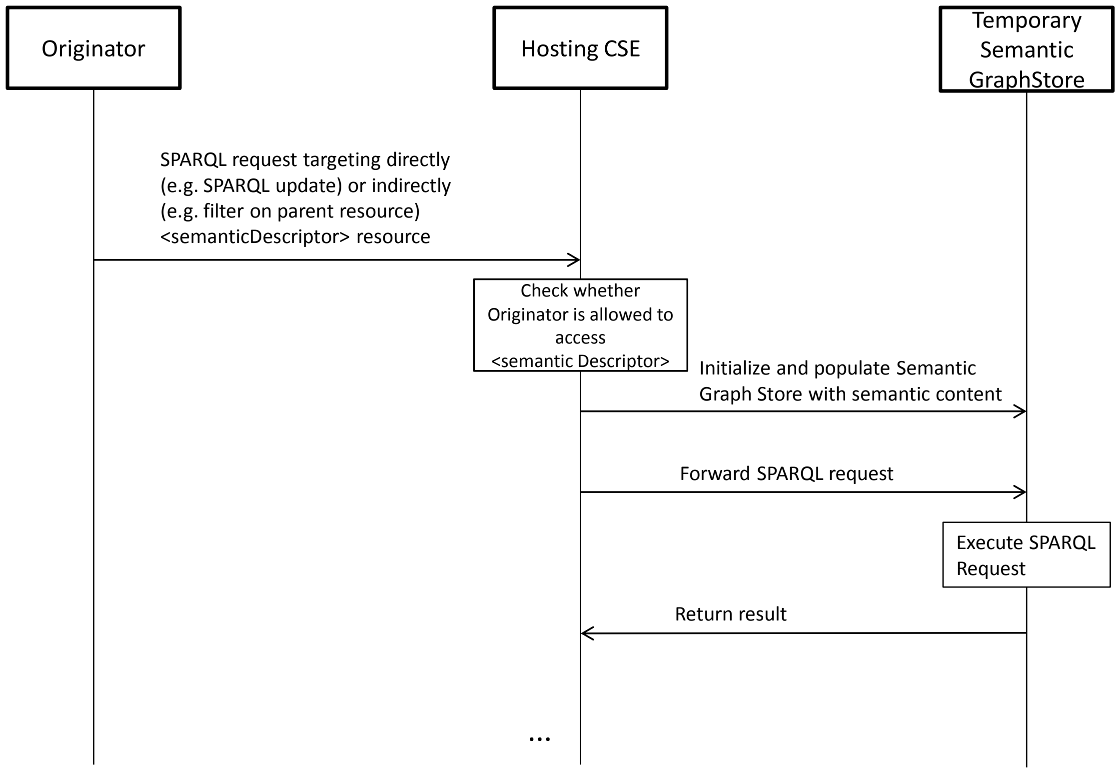
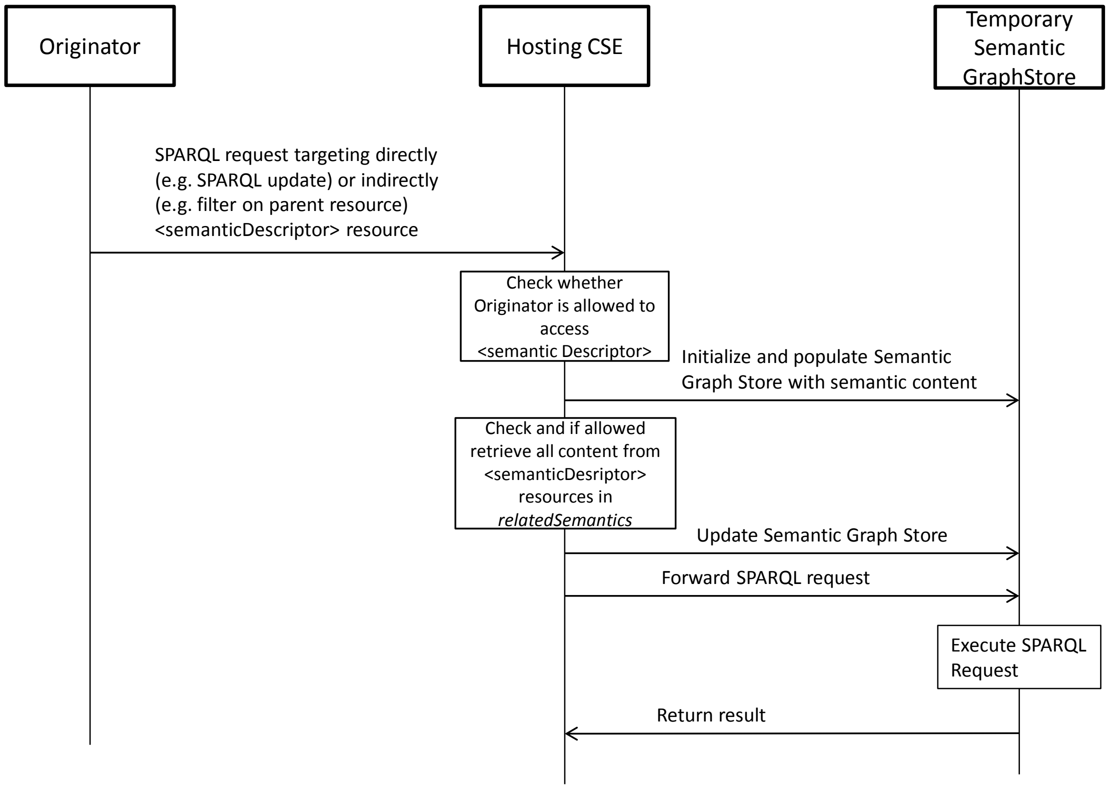


Figure 8.5.7.4-1 Execute SPARQL request on single <semanticDescriptor>

When receiving the request with the SPARQL content, the Hosting CSE checks the access control policy applying to the <semanticDescriptor> resource. If the originator of the request is allowed to access it, it retrieves the semantic information from its *descriptor* attribute, initializes the Temporary Semantic Graph Store, and populates it with the semantic information. Then it forwards the SPARQL request to the Temporary Semantic Graph Store to be executed. Note that the Temporary Semantic Graph Store in this context is seen as an implementation component and not an architectural component according to the oneM2M architecture.

Figure 8.5.7.4-2 shows the case that, in addition to the targeted <semanticDescriptor> resource, a set of related <semanticDescriptor> resources needs to be included before executing the SPARQL request. This is described in Section 8.5.5.4 and TS-0001 [i.39] Section 10.2.35.2.2. The relevant <semanticDescriptor> resources are identified through the *relatedSemantics* attribute.



…

Figure 8.5.7.4-2 Execute SPARQL request on <semanticDescriptor> plus those identified through *relatedSemantics* attribute

In addition to the steps shown in figure 8.5.7.4-1, the Hosting CSE attempts to retrieve the semantic content of the <semanticDescriptor> resources identified by the *relatedSemantics* attribute and updates the Temporary Semantic Graph Store accordingly. The respective access control policies are checked as part of the retrieval as in any other resource access. Only once all the identified (and accessible) content has been added is complete, the SPARQL request is forwarded for execution.

Figure 8.5.7.4-3 shows the case that the semantic content of the targeted <semanticDescriptor> resource contains links to further <semanticDescriptor> resoruces identified by *resourceDescriptorLink* properties. This annotation-based approach is described in Section 8.5.5.3 and in TS-0001 [i.39], Section 10.2.35.2.1.

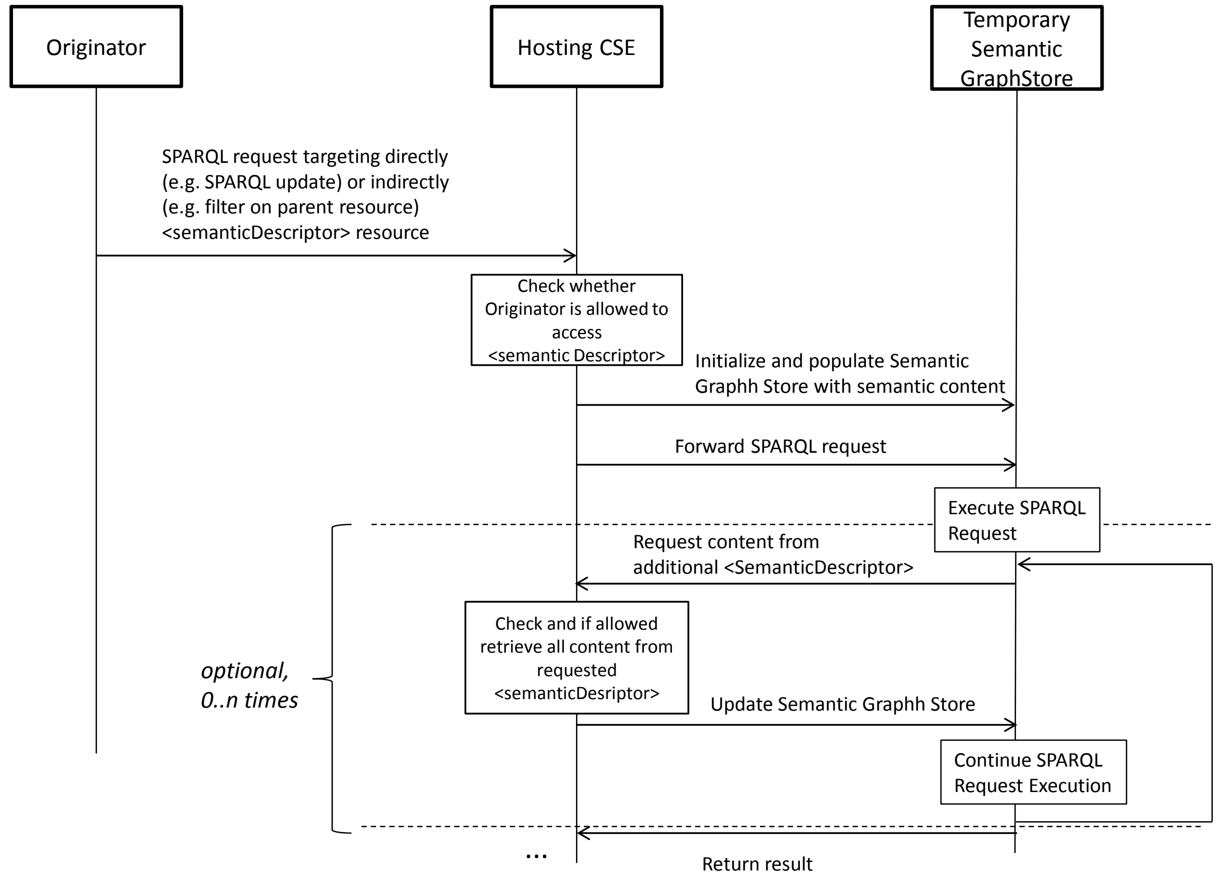


Figure 8.5.7.4-2 Execute SPARQL request on <semanticDescriptor> plus those encountered during the execution in form of *resourceDescriptorLink* properties

In this case, the modified SPARQL engine of the Temporary Semantic Graph Store may encounter elements with *resourceDescriptorLink* properties when matching variables as part of the execution of the SPARQL request. In this case the execution will be halted and the engine requests the content of the <semanticDescriptor> resource identified by the *resourceDescriptorLink* property before continuing execution on the merged content. This can happen 0 to n times during the execution. Note that the Temporary Semantic Graph Store is not an architectural oneM2M component and logically should be seen as part of the Hosting CSE. The semantic content is requested on behalf of the original Originator with its access rights.The advantage of the implementation with the Temporary Semantic Graph Store is that no special care has to be taken with respect to access control policies. Access control policies are evaluated in the usual way when accessing the semantic content on which the SPARQL request is to be executed. The disadvantage is of course that a Temporary Semantic Graph store has to be initialized and populated each time a SPARQL request is to be executed.

8.5.8 Ontology Support Resources

8.5.8.1 Overview

As identified in the requirements, storage, discovery and management of ontologies within the oneM2M platform are key for supporting basic and advanced semantic functionalities within the oneM2M platform. In general, the M2M system needs to represent knowledge as a hierarchy of concepts (ontologies), either external or internal to the M2M domain, using a shared vocabulary to denote the classes, properties and interrelationships of those concepts.

In the clauses 8.5.8.2 and 8.5.8.3 architectural support for Ontology-related functionality is introduced.

8.5.8.2 Ontology Repository

An Ontology Repository is capable of storing multiple ontologies in the unified languages adopted by the M2M system, e.g. RDFS/OWL. For easy illustration of the examples, in this clause its is assumed that the M2M system adopts RDFS/OWL in describing ontologies. Figure 8.5.8.2-1 provides an example of Ontology Repository with the oneM2M Base Ontology, SSN and Saref ontologies represented by individual resources.

This structure provides support for re-use of existing ontologies, the ability to access both internal and external ones and for ontology import into the system. It also allows to fulfil the requirements for ontology discovery, as well as addition and updates, via CRUD operations.

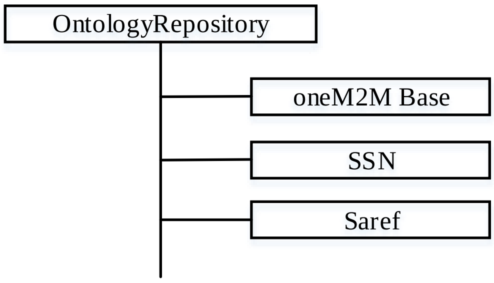


Figure 8.5.8.2-1: Ontology Repository example

Resources of type <CSEBase> and <AE> optionally can have one Ontology Repository resource. The ontology resource is further described in clause 8.5.8.3.

8.5.8.3 Ontology Resource

8.5.8.3.0 Introduction

The *<ontology>* resource is used to store the representation of an ontology. This representation may contain ontology descriptions in a variety of formats, given the requirements for re-use of existing ontologies, for support for ontologies available only externally and for support of ontology import into the system. The ontology description is made available to the semantic-related functions of the oneM2M system provided by applications or CSEs.

Various approaches to defining an ontology resource are identified in the following clauses, with the goal of providing an architecture to support all the identified requirements, as well as the flexibility needed for advanced features.

The "Unstructured" approach seeks to provide the oneM2M system access to any ontology document in a format supported by the system. The "Structured" approach aims to provide a oneM2M resource structure suitable for representing ontology information within the system. Another method seeks to provide flexibility in using both of these concepts through a dual use approach.

8.5.8.3.1 Use of Ontologies

In the following, a number of examples are given what applications, but also the semantic functionalities supported by the oneM2M platform itself, may need from an ontology:

1) get all classes of an ontology;

2) get all object | data properties of ontology;

3) get direct subclasses of class A;

4) get also transitive subclasses class A;
e.g. if information from instances of class A is requested, all subclasses of class A also need to be included as they are also instances of class A;

5) get all the superclasses of class A;
e.g. if for derived ontologies the class of the base ontology needs to be found from which the class is derived, for example to apply rules defined for the base ontology, e.g. for creating a resource structure;

6) get all object | data properties where class A is in the domain;
e.g. to find out what properties an instance of class A can possibly have;

7) get all object | data properties where class A is in the range;

8) get all sub-properties of a property A;
e.g. if information concerning property A is requested all sub-properties of A also need to be included;

9) get classes that are equivalent to class A.

8.5.8.3.2 Unstructured approach

8.5.8.3.2.0 Introduction

Using OWL 2.0 as an ontology format example to be supported by the oneM2M system and based on W3C specifications (http://www.w3.org/TR/owl2-syntax/#IRIs) the following apply:

- *"Ontologies and their elements are identified using Internationalized Resource Identifiers (IRIs) [RFC3987]; thus, OWL 2 extends OWL 1, which uses Uniform Resource Identifiers (URIs). Each IRI MUST be absolute (i.e. not relative). In the structural specification, IRIs are represented by the IRI UML class. Two IRIs are structurally equivalent if and only if their string representations are identical."*

And

- *"Ontology documents are not represented in the structural specification of OWL 2, and the specification of OWL 2 makes only the following two assumptions about their nature:*
  - *Each ontology document can be accessed via an IRI by means of an appropriate protocol.*
  - *Each ontology document can be converted in some well-defined way into an ontology (i.e. into an instance of the Ontology UML class from the structural specification)."*

Therefore current methods of accessing and importing ontologies requires access to the respective ontology document via an IRI (Internationalized Resource Identifiers) as specified in IETF RFC 3987 [i.42]. Given that access to the ontology document has been obtained, this approach also provides for local storage of the document in a *content* attribute which is available to the platform based on access control rules.

Given the possible need to have access to multiple versions of an ontology, and to different formats, a specialized attribute *contentFormat* provides information necessary for the system to interpret the information available in the *content* attribute.

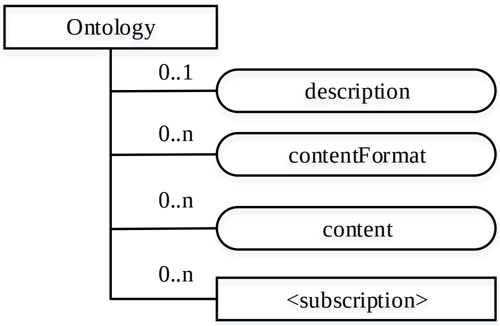


Figure 8.5.8.3.2.0-1: <*ontology*> resource for ontology document access

The <*ontology*> resource above contains the child resources specified in table 8.5.8.3.2.0-1.

Table 8.5.8.3.2.0-1: Child resources of <*ontology*> resource in the unstructured approach

Child Resources of <semanticDescriptor>	Child Resource Type	Multiplicity	Description
[variable]	<*subscription*>	0..n	See [i.39], clause 9.6.8 where the type of this resource is described.

The <*ontology*> resource above contains the attributes specified in table 8.5.8.3.2.0-2.

Table 8.5.8.3.2.0-2: Attributes of <*ontology*> resource in the unstructured approach

Attribute Name	Multiplicity	RW/RO/WO	Description
Description	0..1	RW	Text description of the ontology
contentFormat	1..n	RW	Attribute providing information about the format of the *content* attribute. It may indicate the content as: IRI - for an ontology to be accessed via the IRI provided in the *content* attribute OR File format - for an ontology for which the document is stored in the *content* attribute. In this case *contentFormat* also provides a description of the ontology format, e.g. OWL, Turtle, etc.
Content	0..n	RW	Depending on the *contentFormat* attribute, it may be interpreted either as: The IRI of the corresponding ontology document OR The content of the corresponding ontology document Editor's Note:It is FFS to determine how to handle the case when the content is the IRI of the corresponding ontology document, but it is not dereferenceable.

8.5.8.3.2.1 SPARQL request on ontology procedure via Retrieve Operation

This procedure shall be used for SPARQL requests to <ontology> resources. A ***Semantic Request*** parameter, defined as follows has to be provided:

- ***Semantic Request***: Contains a SPARQL request to be executed on the ontology content.

The ***contentFormat*** attribute of the <ontology> has to represent a file format. The result corresponds to the result of the execution of the SPARQL request on the ***content*** attribute of the <ontology> resource and shall be returned to the Originator using a successful Response message.

Editor's note: The table detailing the *SPARQL request on ontology procedure* needs to be provided.

In the following, the SPARQL content for the ***Semantic Request*** parameter for all the examples defined in clause 8.5.8.3.1 is given:

1) get all classes of an ontology
SELECT ?subject WHERE { ?subject rdfs:subClassOf+ owl:Thing }

2) get all object | data properties of ontology
SELECT ?subject WHERE { {?subject rdf:type+ owl:ObjectProperty } UNION {?subject rdf:type+ owl:DatatypeProperty } }

3) get direct subclasses of class A
SELECT ?subject WHERE { ?subject rdfs:subClassOf saref:Command }

4) get also transitive subclasses class A
SELECT ?subject WHERE { ?subject rdfs:subClassOf + saref:Command }

5) get all the superclasses of class A
SELECT ?object WHERE { saref:SetAbsoluteLevelCommand rdfs:subClassOf + ?object }

6) get all object | data properties where class A is in the domain
SELECT ?subject ?object WHERE { ?subject rdfs:domain saref:Service }

7) get all object | data properties where class A is in the range
SELECT ?subject ?object WHERE { ?subject rdfs:range saref:Command }

8) get all sub-properties of a property A
SELECT ?subject WHERE { ?subject rdfs:subPropertyOf om:singular_unit

9) get classes that are equivalent to class A
SELECT ?class WHERE {{ saref:Device owl:equivalentClass ?class} UNION {?class owl:equivalentClass saref:Device}}

8.5.8.3.3 Structured approach

The structured approach provides a oneM2M resource structure suitable for representing ontology information within the system.

In this case, the resource structure seeks to represent and maintain all the class and relationship information provided by the ontology definition. As such individual <class> and <relationship> sub-resources are defined, with attributes providing the corresponding mapping.

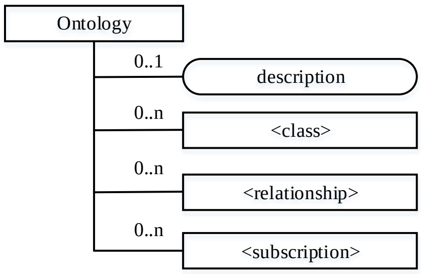


Figure 8.5.8.3.3-1: *<ontology>* resource for ontology document access

The *<ontology>* resource above contains the child resources specified in table 8.5.8.3.3-1.

Table 8.5.8.3.3-1: Child resources of *<ontology>* resource in the structured approach

Child Resources of *<semanticDescriptor>*	Child Resource Type	Multiplicity	Description
[variable]	*<class>*	0..n	
[variable]	*<relationship>*	0..n	
[variable]	*<subscription>*	0..n	See [i.39], clause 9.6.8 where the type of this resource is described.

The *description* attribute provides high-level description of the ontology. The attributes of two sub-resources are detailed in tables 8.5.8.3.3-2 and8.5.8.3.3-3, providing also examples based on the oneM2M Base Ontology. Additional attributes may be envisioned and added for to further enable new functionality.

Table 8.5.8.3.3-2: Attributes of <*class*> resource

Attribute Name	Multiplicity	RW/RO/WO	Description
isSubjectOf	1..n	RW	URI(s) of a <relationship> resource for which the class is a subject E.g. for oneM2M Base class <Service> this attribute may be the URI of: <hasOperation>, <consistsOf>
isObjectOf	1..n	RW	URI(s) of a <relationship> resource for which the class is an object E.g. for oneM2M Base class <Service> this attribute may be the URI of: <hasService>, <isExposedBy>
hasSubclass	0..n	RW	URI to another class which is a subclass of the one being defined E.g. for oneM2M Base class <thing> this attribute may be the URI of <device>
isSubclassOf	0..n	RW	URI to another class resource which is a superclass of the one being defined E.g. for oneM2M Base class <device> this attribute may be the URI of <thing>
equivalentTo	0..n	RW	URI to another class resource which is the equivalent of this class

Table 8.5.8.3.3-3: Attributes of <*relationship*> resource

Attribute Name	Multiplicity	RW/RO/WO	Description
relationshipCategory	0..n	RW	Optional, describes the relationship type, e.g. Synonymy, Antonymy, Hyponymy, Meronymy, Holonymy
hasSubject	1..n	RW	URI(s) of a <class> resource who is a subject for this relationship
hasObject	1.. n	RW	For Object relationships/properties, a URI(s) to a class which is the object for this relationship. For Data properties it would contain a data type
restriction	0..n	RW	Restrictions posed by this relationship, as they map to the OWL use of restriction

Figures 8.5.8.3.3-1 and 8.5.8.3.3-2 further detail how the oneM2M Base Ontology would be represented using the structured approach (not all of the currently defined classes and relationships have been depicted).

NOTE: Notations like <serviceClass> and<hasOperationRel> are meant to convey that these represent the <service> resource of <class> type and <hasOperation> resource of <relationship> type. The notation isSubjectOf (hasOperation) denotes an attribute of isSubject of type with the value hasOperation.

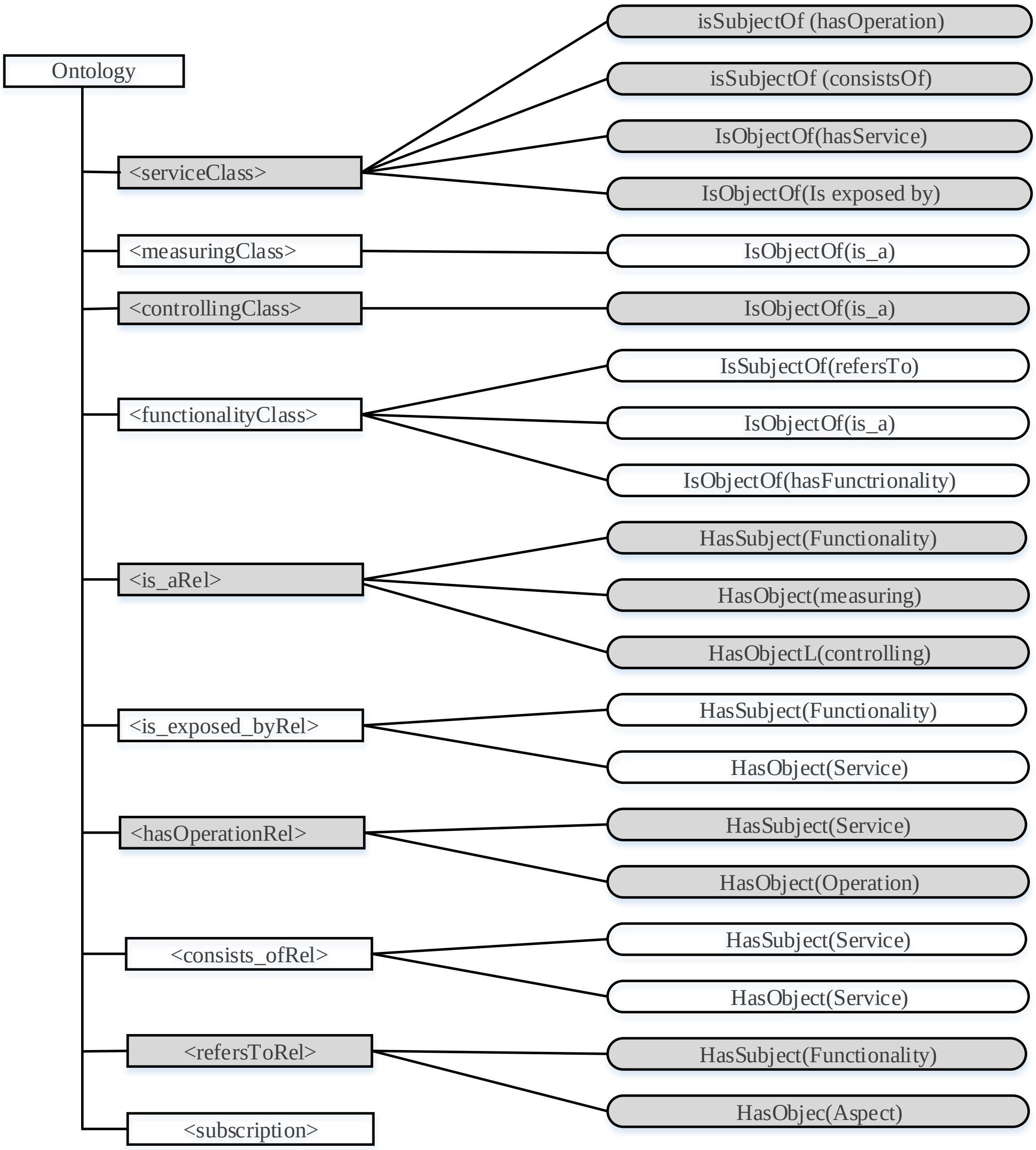


Figure 8.5.8.3.3-2: Example of <ontology> resource as a representation of the oneM2M Base Ontology

NOTE: Not all aspects of ontologies are represented as resources yet, e.g. individuals and implicitly defined classes.

8.5.8.3.4 Dual approach

The unstructured approach presented above has the advantage of providing direct access to the ontology file which then can be locally stored, cached, etc. It represents a simple method of accessing a broad range of ontologies in a straightforward manner. Using semantic requests on <ontology> resources, the aspects of the ontology relevant to the requester can be retrieved, making use of the full expressiveness of SPARQL.

NOTE: SPARQL can also be used for partial updates, which has not been described yet.

The structured approach, in turn creates a resource representation of the ontology within the oneM2M resource tree with addressable entities for several of aspects of the ontology. The resource structure enables inter-ontology mapping within the platform, as well as providing ways for representing ontology extensions, especially to externally defined ontologies. It also enables use of ontology sections and easier to identify partial updates of the ontology, which may trigger semantic annotation updates. It is also envisioned to enable reasoning-related features, as each class/relationship is discoverable and addressable.

Figure 8.5.8.3.4-1 presents a dual mode approach to the ontology resource definition which allows both representations to be used by the system, for maximum flexibility and increased system capabilities.

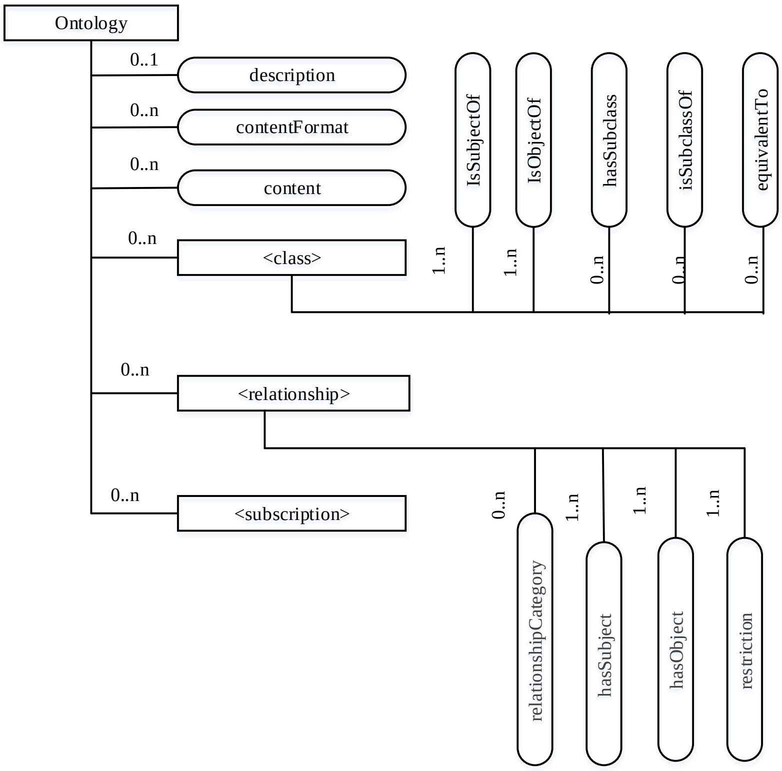


Figure 8.5.8.3.4-1: Ontology resource structure in the Dual Approach

8.6 Interworking with non oneM2M Devices and Area Networks in Rel-1

8.6.1 Semantic support in oneM2M Release 1

oneM2M Release 1 does not yet provide full semantic functions. However some limited aspects of semantics can already be used in Rel-1 to support interworking with Area networks and non-oneM2M devices. For that purpose only the structure information (concepts, relations), that is contained in ontologies, is proposed to be used in Rel-1.

A mechanism (a "cook book") is described that shows how that structure information can be utilized to create oneM2M resources (AEs and Containers) that are needed for interworking with non-oneM2M devices and area networks.

For that purpose the "ontologyRef" attribute of AEs and Containers as described in oneM2M TS 0001 [i.39] is used.

It should be noted that this approach for Rel-1 does not yet allow for more dynamic usage of semantic information like discovery, reasoning or mash-ups.

8.6.2 Basic functional principles

Annex F of oneM2M TS 0001 [i.39] describes Interworking/Integration of non-oneM2M solutions and protocols with the oneM2M System. Interworking is accomplished through specialized M2M Applications, called "Interworking Proxy Application Entities" (IPEs).

An Interworking Proxy Application Entity interfaces:

1) a CSE of the oneM2M System (IN-CSE, MN-CSE or ASN-CSE);

2) one or more Area Networks and the Non-oneM2M devices in these Area Networks.

The Interworking Proxy maps the native data model and communication primitives of the Non-oneM2M devices into oneM2M resources that can be accessed by oneM2M procedures (Create, Retrieve, Update, Delete). Interworked non-oneM2M devices communicate over Area Networks with the Nodes (IN, MN or ASN) of oneM2M and these Nodes therefore need to contain communication capabilites to for the Area Networks. Note, that Area Networks for legacy technologies (e.g. Mbus/COSEM) often are not based on IP.

Figure 8.6.2-1 illustrates how a M2M Application (e.g. a Utility Application) can access non-oneM2M devices (e.g. Sensors/Meters implementing MBus/COSEM or ZigBee technology) that interwork via Interworking Proxies with the oneM2M System.

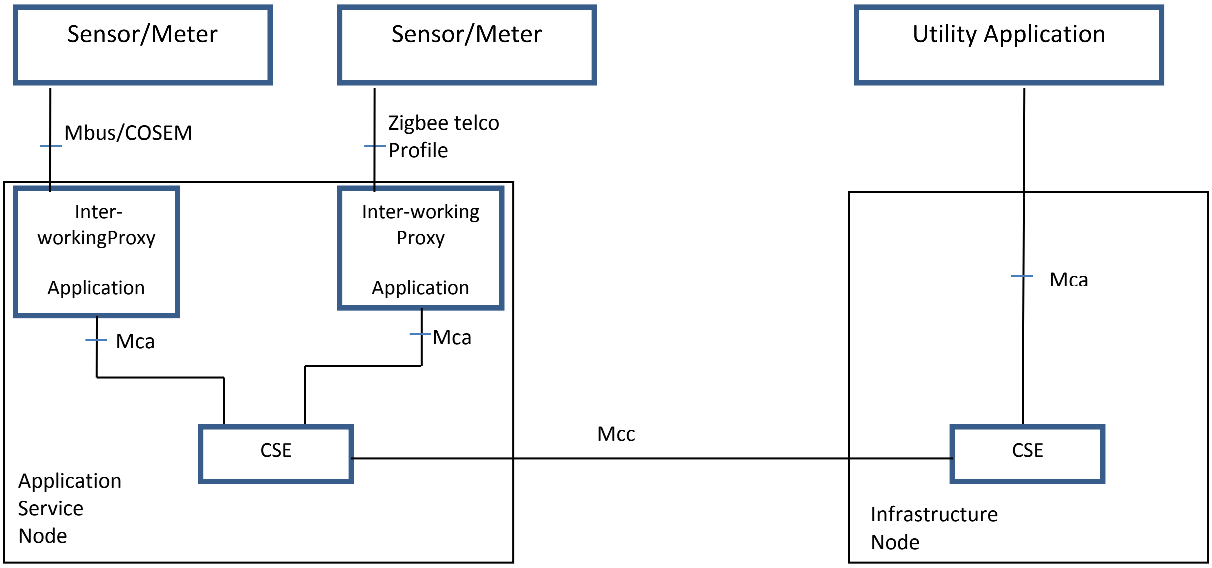


Figure 8.6.2-1

In this example, the scenario has been modelled with two IPEs within a node. In a different scenario there could also just be a single IPE responsible for both networks.

8.6.3 Generic topology for interworking with non-oneM2M devices and Area Networks

Figure 8.6.3-1 shows the concept model (following the Approach described in clause 8.2.1) representing an Area Network, its non-oneM2M devices as well as its relationship to an Interworking Proxy Application Entity (IPE). In clause 8.2 the modelling approach was presented, using a simpler model consisting of devices, operations and parameters. In this clause a solution is proposed that also takes into account M2M networks and has a more fine-grained modelling differentiating between non-oneM2M devices and FunctionBlocks. As clause 8.4 deals with interworking with other technologies, we explicitly model *non-oneM2M devices.* oneM2M devices are already modelled using the oneM2M resource structure and therefore are not considered here.

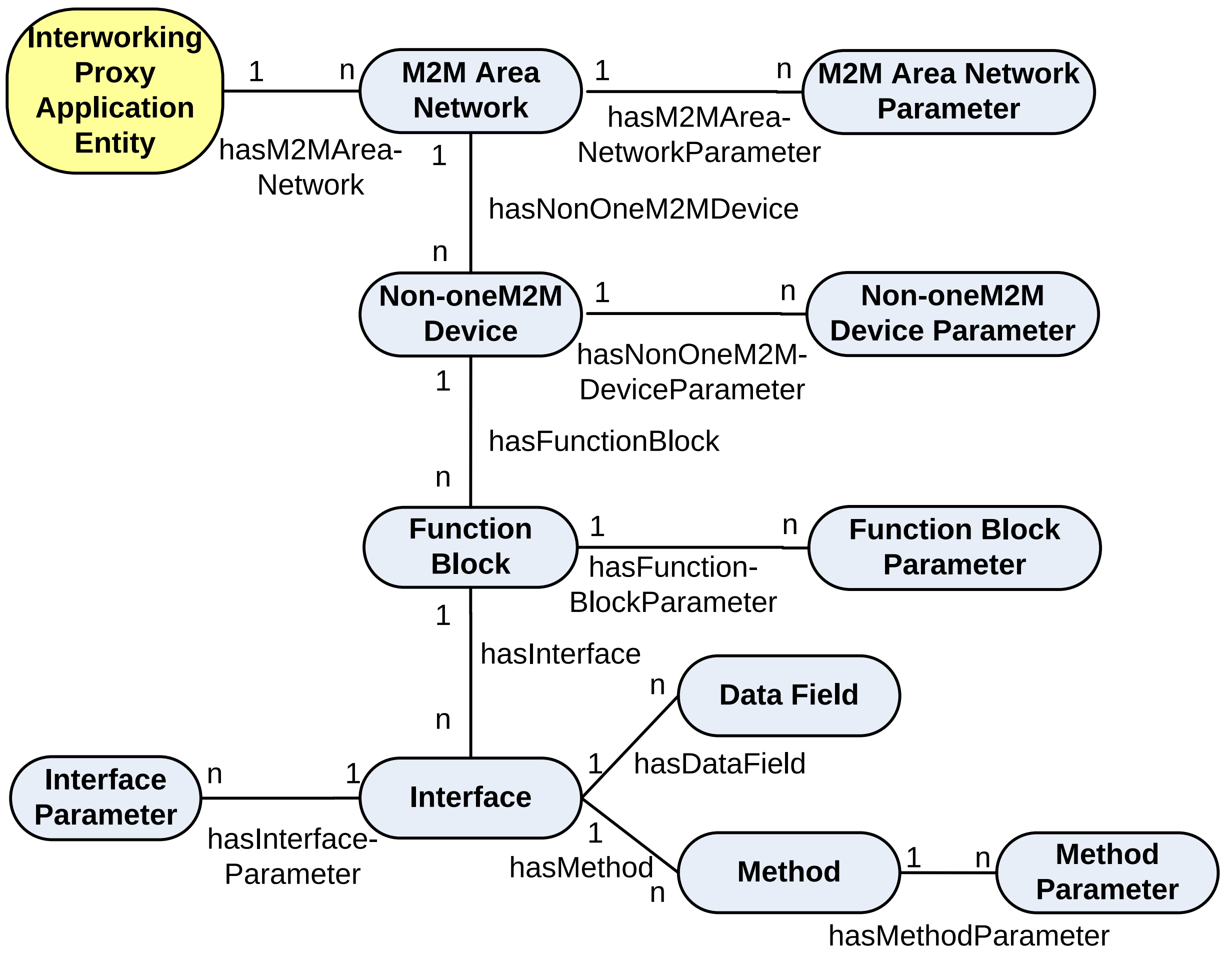


Figure 8.6.3-1

An **Interworking Proxy Application Entity** is an M2M Application that communicates with one or more M2M Area Networks.

The **M2M Area Network** contains a number of non-oneM2M Devices.

EXAMPLE 1: ZigBee network with Home Automation Profile.

Each **non-oneM2M Device** is an entity of the M2M Area Network that contains one or more Function Blocks.

EXAMPLE 2: ZigBee device object.

A **Function Block** is a sub-unit of a non-oneM2M Device that performs a some services of the non-oneM2M Device.

EXAMPLE 3: ZigBee Thermostat, a ZigBee Temperature Sensor.

Each Function Block has **Interfaces** for communication of data to/from the Function Block. Each Interface consists of combinations of:

- **Data Fields** that can be read and/or updated.
- **Methods** that can be executed.

Each Method has its **Method Parameters** that can (a) be input to the device or (b) output from the device.

8.6.4 Semantic modelling of interworking

In clause 8.4.3 the high-level semantic concepts and their relations have been defined. These high-level concepts are used for defining the oneM2M resource structure for interworking. According to the type modelling described in clause 8.2.2.2 the specific type related to the respective high-level concepts can be modelled and the ontologyRef attribute can be used to refer to it.

EXAMPLE: A ZigBeeTemperatureSensor can be modelled as a specific device, so the ontologyRef could be http://[ZigBeeOntology]#TemperatureSensor, indicating that the modelled device AE actually represents a temperature sensor.

For modelling the high-level semantic concepts *AE* resources and *container* resources are used. *AE* resources are used for those concepts where functionality may be handled by independent applications. Other concepts, as well as relations between concepts, are modelled by containers and subcontainers respectively, using the respective *ontologyRef* attribute to reference the ontology concept (e.g. OWL class) or the relation (e.g. OWL object properties). The values in the final subcontainer instance contain the URL of the oneM2M resource providing the content of the referenced instance.

EXAMPLE 2: The M2M Area Network may be modelled as an *AE* Resource. The relation *hasNon-oneM2M Device* between the concepts *M2M Area Network* and *Non-oneM2M Device* to describe that an M2M Area Network contains non-oneM2M Devices is mapped to a container for "hasNon-oneM2M Device", which has a subcontainer for each Non-oneM2MDevice, which is part of the network. The instance of the subcontainer resource contains the resource URI pointing to the *AE* Resource of the Non-oneM2MDevice.

In this way, any oneM2M application can utilize the fixed resource structure to access data and semantic-aware oneM2M applications can use the *ontologyRef* attribute to find out about the more specific semantic content of the resources, e.g. that the Non-oneM2M device is actually a ZigBee temperature sensor.

8.6.5 Mapping into oneM2M resources

8.6.5.0 Introduction

This clause describes the mapping principles that are used to map a generic M2M Area Network into a structured tree of oneM2M resources.

Naming convention: Following the convention in clause 9.5 of [i.39].

- A string delimited with '<' and '>' e.g. <resourceType> is a placeholder for the type of a resource.
- A string delimited with '[' and ']' e.g. [resourceName] is a placeholder for the name of a resource or an attribute.

Similarly, italics are used for use of ontologies (e.g. *oneM2M_ontology)* For example, the expression: *http://[oneM2M_ontology]#Area_Network* would indicate a concept *Area_Network* in some ontology. *[oneM2M_ontology]* is the placeholder for the name of that ontology, http://*[oneM2M_ontology]* is the URI of that ontology.

As explained before, the IPE is an M2M Application.

The basic principle followed here is to map Non-oneM2M entities (M2M Area Networks, non-oneM2M Devices) that are accessible via the IPE application are mapped into resources of type <AE> that are linked to the IPE application resource. The entities Function Blocks, Interfaces, Data Fields, Methods and Method Parameters are mapped into *container* resources as sub resources of a non-oneM2M Device AE. Figure 8.6.5.0-1 shows this high-level mapping.

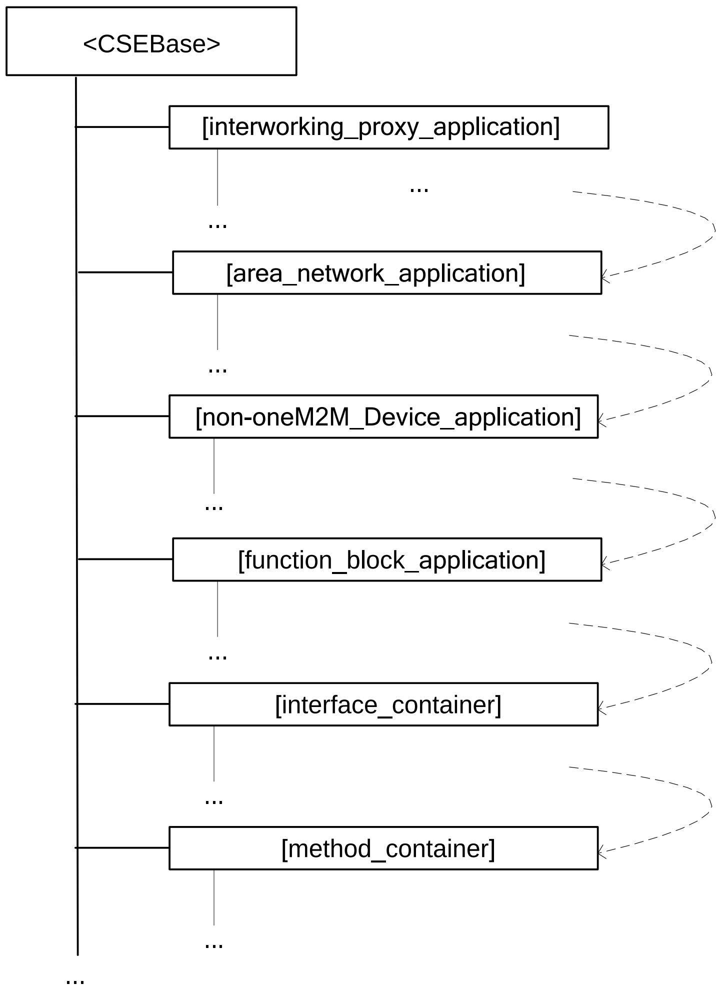


Figure 8.6.5.0-1: High-level resource structure for interworking

8.6.5.1 Interworking Proxy Application Entity

8.6.5.1.0 Introduction

The Interworking Proxy Application Entity (IPE) is modelled as a resource [interworking_proxy_application] of Resource Type <AE>. The URI used to access this resource has the following format:

[CSEBase]/[interworking_proxy_application]

e.g.: IN-CSEID.m2m.myoperator.org/CSERoot/myIPE

NOTE: While the addressing in the example above and in the rest of this clause follows the Hierarchical URI Method convention ([i.39], clause 9.3.1) all three different methods for addressing a resource within the oneM2M resource structure as described in [i.39], clause 9.3.1 are possible.

[CSEBase] is the Hosting CSE (IN-CSE, MN-CSE or ASN-CSE) where the IPE had been created by administrative means.

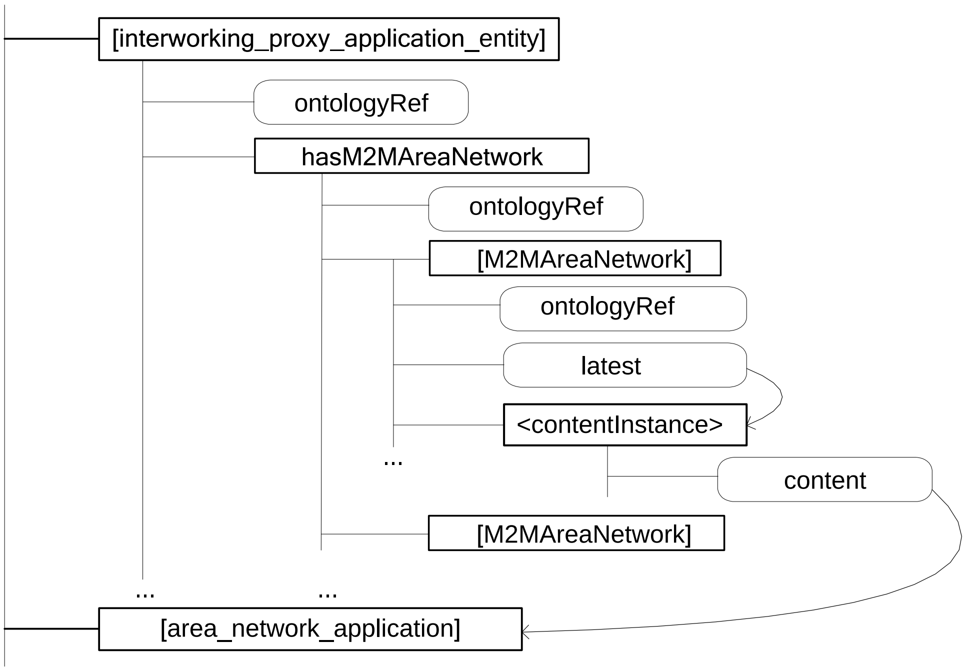


Figure 8.6.5.1.0-1

The ontologyRef attribute of the [interworking_proxy_application] resource has a value of:

http://[oneM2M_ontology] #IPE

or a more specific variant (e.g. a specific IPE for home automation by some industry group) that is a sub-class of the IPE class in the oneM2M_ontology.

8.6.5.1.1 Linking the Interworking Proxy Application Entity to its M2M Area Networks

The Interworking Proxy Application Entity resource contains a sub resource of type <container> with the fixed name *hasM2MAreaNetwork* that mirrors the relation *hasM2MAreaNetwork* between the concepts *InterworkingProxyApplication* and *M2M Area Network*. It contains subcontainers with references to each M2M Area Network the IPE supports.

The URI used to access this <container> resource has the following format:

[CSEBase]/[interworking_proxy_application]/hasM2MAreaNetwork

e.g.: IN-CSEID.m2m.myoperator.org/CSERoot/myIPE/hasM2MAreaNetwork

The hasM2MAreaNetwork resource of type <container> contains sub-containers of type <container>. Each sub-container of the hasM2MAreaNetwork container holds a reference to an M2M Application that represents one M2M Area network that is supported by the IPE.

When a new M2M Area network is supported by the IPE then the IPE:

1) creates a new M2M Area Network Application;
2) creates a new sub-container in the hasM2MAreaNetwork container whose contentInstance contains a reference to the M2M Area Network Application.

8.6.5.2 M2M Area Network Application

8.6.5.2.0 Introduction

The M2M Area Network Application is modelled as a resource [area_network_application] of Resource Type <AE>. An M2M Area Network Application is created by an IPE.

The URI used to access this resource has the following format:

[CSEBase]/[area_network_application]

e.g.: IN-CSEID.m2m.myoperator.org/CSERoot/myIPE_ZigBeeNW

The M2M Area Network Application is responsible for:

- Managing the one M2M Area Network, containing technology dependent logic.
  The management of the one M2M Area Network may be supported by using oneM2M Management Functions (e.g. by utilizing a related *areaNwkInfo* resource).
- Discovering the M2M Area Network structure (devices, etc.) and keeping track of changes of the structure.

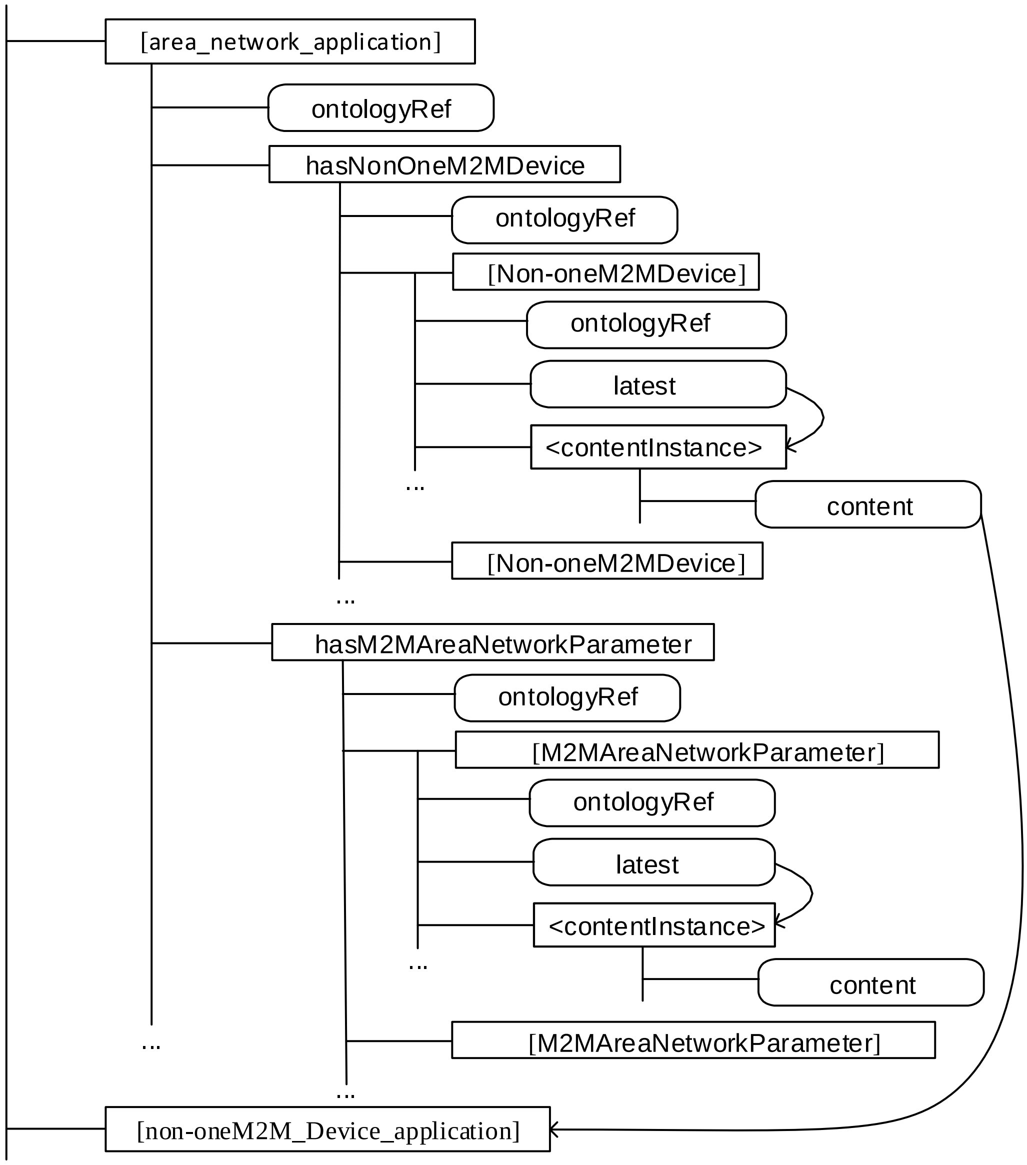


Figure 8.6.5.2.0-1

The M2M Area Network Application is dependent on the technology of the M2M Area Network and the hardware (drivers, etc.) of the area network communication module.

The ontologyRef attribute of the [area_network_application] resource has a value of:

http://[oneM2M_ontology] #Area_Network

or a technology specific variant that is a sub-class of the oneM2M_ontology, e.g.:

http://[ZigBee_ontology] #ZigBee_Area_Network

8.6.5.2.1 Linking the M2M Area Network to its non-oneM2M devices

The M2M Area Network Application resource contains a sub resource of type container with the fixed name *hasNonOneM2MDevice* that mirrors the relation *hasNon-oneM2M Device* between the concepts *M2M Area Network* and *Non-oneM2M Device*.

The URI used to access this <container> resource has the following format:

[CSEBase]/[area_network_application]/hasNonOneM2MDevice

e.g.: IN-CSEID.m2m.myoperator.org/CSERoot/myIPE/hasNonOneM2MDevice

The ontologyRef attribute of the hasNonOneM2MDevice container has a value of:

http://[oneM2M_ontology] #hasNonOneM2MDevice

or a more specific variant that is a sub-property of the *hasNonOneM2MDevice* property in the oneM2M_ontology.

This hasNonOneM2MDevice container in turn has sub-containers [Non-oneM2M_Device] whose contenInstance each holds a reference to an M2M Application that represents one non-oneM2M Device in the M2M Area Network.

The URI used to access a [Non-oneM2M_Device] <container> resource whose contentInstance contains the reference to a Non-oneM2M Device Application has the following format:

[CSEBase]/[area_network_application]/hasNonOneM2MDevice/[Non-oneM2M_Device]

e.g.: IN-CSEID.m2m.myoperator.org/CSERoot/myIPE/hasNonOneM2MDevice/myNon-oneM2M_Device_1

8.6.5.2.2 Parameters of the M2M Area Network

The M2M Area Network Application resource contains a sub resource of type <container> with the fixed name hasM2MAreaNetworkParameter that contains the parameters for that M2M Area Network.

The URI used to access this <container> resource has the following format:

[CSEBase]/[area_network_application]/hasM2MAreaNetworkParameter

e.g.: IN-CSEID.m2m.myoperator.org/CSERoot/myIPE/hasM2MAreaNetworkParameter

The ontologyRef attribute of the hasM2MAreaNetworkParameter container has a value of:

http://[oneM2M_ontology]#hasM2MAreaNetworkParameter

or a more specific variant that is a sub-property of the *hasM2MAreaNetworkParameter* property in the oneM2M ontology.

This *hasNonOneM2MDevice* container in turn has sub-containers [M2MAreaNetworkParameter] that each contain the value of that parameter. The ontologyRef attribute of the [M2MAreaNetworkParameter] container has a value of:

http://[oneM2M_ontology] # M2MAreaNetworkParameter

or a technology specific variant that is a sub-class of *M2MAreaNetworkParameter* of the oneM2M_ontology, e.g.:

http://[ZigBee_ontology] #ZigBee_Area_Network_Parameter.

The contentInstance of the [M2MAreaNetworkParameter] container will contain the value of that parameter as (opaque) content.
If the ontology containing the *M2MAreaNetworkParameter (*or its sub-class in the technology specific ontology) additionally specifies the data type of the parameter (e.g. as DatatypeProperty in OWL) then the ontologyRef attribute of the contentInstance may contain the value of that datatype. E.g.:

http://www.w3.org/2001/XMLSchema#positiveInteger

8.6.5.2.3 Functional description of the M2M Area Network Application

When a new non-oneM2M Device is added to/removed from the M2M Area Network then the M2M Area Network Application:

1) creates/deletes the Non-oneM2M Device Application;

2) creates/deletes the sub-container of its hasNonOneM2MDevice container whose contentInstance contains the reference to the Non-oneM2M Device Application.

8.6.5.3 Non-oneM2M Device Application

8.6.5.3.0 Introduction

The Non-oneM2M Device Application is modelled as a resource [non-oneM2M_Device_application] of Resource Type <AE>. A Non-oneM2M Device Application is created by an M2M Area Network Application.

The URI used to access this resource has the following format:

[CSEBase]/[non-oneM2M_Device_application]

e.g.: IN-CSEID.m2m.myoperator.org/CSERoot/myIPE_ZigBeeNW_HomeController

The Non-oneM2M Device Application is responsible for:

- communicating with the Non-oneM2M Device by using (technology specific) communication means of the M2M Area Network;

- exposing function blocks of the Non-oneM2M Device.

The ontologyRef attribute of the [non-oneM2M_Device_application] resource has a value of:

http://[oneM2M_ontology] #non_oneM2M_Device

or a technology specific variant that is a sub-class of the oneM2M_ontology, e.g.:

http://[ZigBee_ontology]#ZigBee_Router_node

8.6.5.3.1 Linking the non-oneM2M device to its Function blocks

The Non-oneM2M Device Application resource contains a sub resource of type <container> with the fixed name hasFunctionBlock that mirrors the relation *hasFunctionBlock* between the concepts *Non-oneM2M Device* and *FunctionBlock*.

The URI used to access this <container> resource has the following format:

[CSEBase]/[non-oneM2M_Device_application]/hasFunctionBlock

e.g.: IN-CSEID.m2m.myoperator.org/CSERoot/myIPE_ZigBeeNW_HomeController/hasFunctionBlock

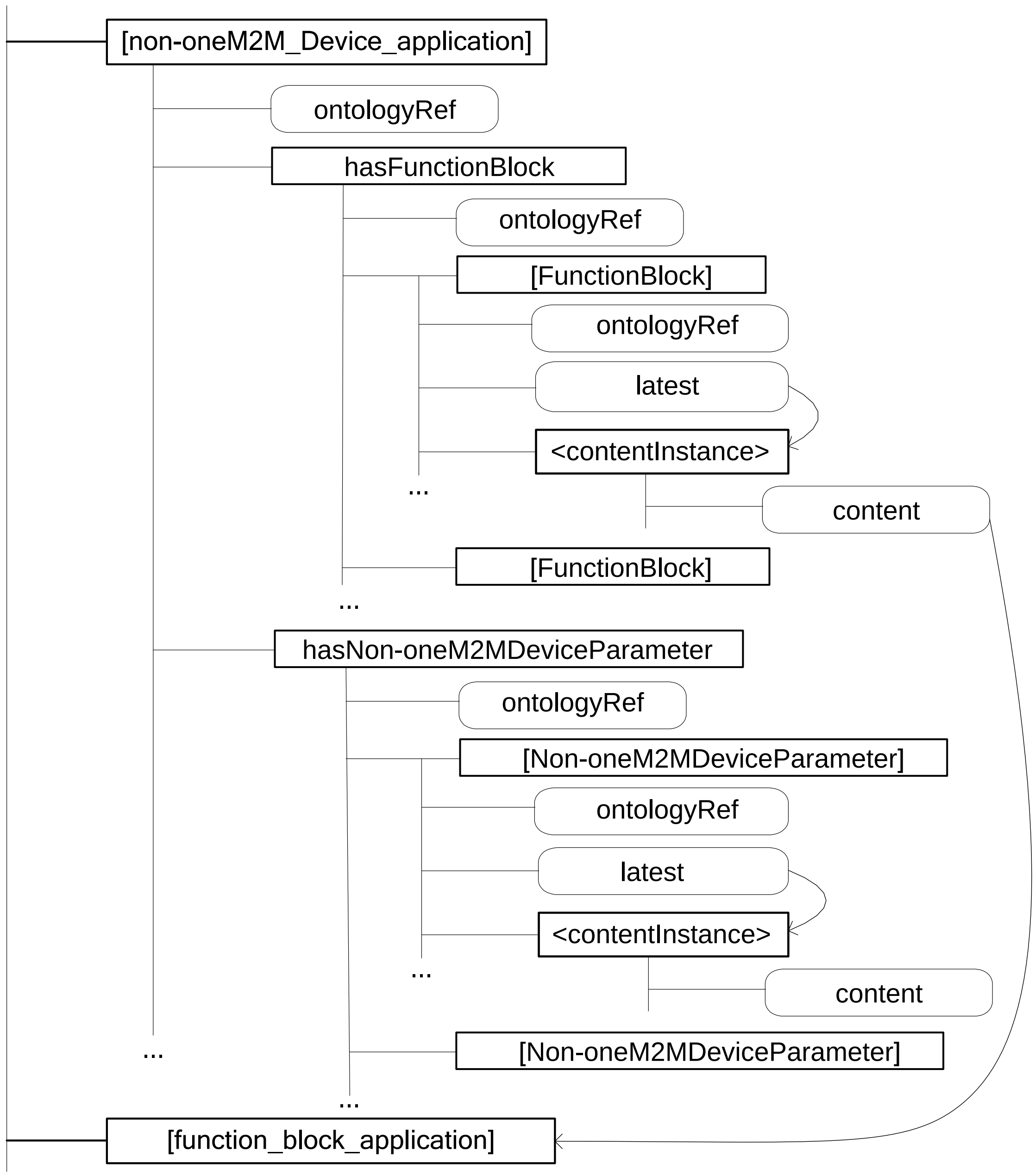


Figure 8.6.5.3.1-1

The ontologyRef attribute of the hasFunctionBlock container has a value of:

http://[oneM2M_ontology] #hasFunctionBlock

or a more specific variant that is a sub-property of the *hasFunctionBlock* property in the oneM2M ontology.

This hasFunctionBlock container in turn has sub-containers [FunctionBlock] that each hold a reference to an M2M Application that represents one Function Block of the non-oneM2M Device.

8.6.5.3.2 Parameters of the Non-oneM2M Device Application

The Non-oneM2M Device Application resource contains a sub resource of type <container> with the fixed name hasNon-oneM2M DeviceParameter that contains the parameters for that M2M Area Network.

The ontologyRef attribute of the hasNon-oneM2MDeviceParameter container has a value of:

http://[oneM2M_ontology]#hasNon-oneM2MDeviceParameter or a more specific variant that is a sub-property of the *hasNon-oneM2MDeviceParameter* property in the oneM2M ontology.

This hasNon-oneM2M DeviceParameter container in turn has sub-containers [Non-oneM2MDeviceParameter] that each contain the value of that parameter. The ontologyRef attributes of the [Non-oneM2MDeviceParameter] container and its contentInstances are built analogously to clause 8.4.5.2.2.

8.6.5.3.3 Functional description of the Non-oneM2M Device Application

When a new non-oneM2M Device has been created by the M2M Area Network Application then, by administrative means (manually or automatically if the technology of the non-oneM2M network enables transmission of the relevant information) the Non-oneM2M Device Application:

1) creates new Function Block Applications for the non-oneM2M Device;

2) creates new contentInstances in its hasFunctionBlock container that contains references to the Function Block Applications.

8.6.5.4 Function Block Application

8.6.5.4.0 Introduction

Function Blocks of a non-oneM2M device are sub-functions of the device. Communication with one function block is independent from communication with other function blocks.
An example of a function block could be a ZigBee® application on a ZigBee® node (a non-oneM2M Device).

The Function Block Application is modelled as a resource [function_block_application] of Resource Type <AE>. A Function Block Application is created by a Non-oneM2M Device Application.

The URI used to access this resource has the following format:

[CSEBase]/[function_block_application]

The Function Block Application is responsible for:

- communicating with the function block on the Non-oneM2M Device by using (technology specific) communication means of the M2M Area Network;

- exposing function blocks of the Non-oneM2M Device.

The ontologyRef attribute of the [function_block_application] resource has a value of:

http://[oneM2M_ontology] #function_Block

or a technology specific variant that is a sub-class of the oneM2M_ontology, e.g.:

http://[ZigBee_ontology] #ZigBee_Router_application

8.6.5.4.1 Linking the Function block to its Interfaces

The Function Block Application resource contains sub resources of type <container> with the fixed name hasInterface that mirrors the relation *hasInterface* between the concepts *FunctionBlock* and *Interface*.

The ontologyRef attribute of the *hasInterface* container has a value of:

http://[oneM2M_ontology] #hasInterface

or a more specific variant that is a sub-property of the *hasInterface* property in the oneM2M ontology.
This hasInterface container in turn has sub-containers [Interface] whose contentInstance each holds a reference to a resource of type <container>, representing one Interface of the Function Block.

The URI used to access this *container* resource has the following format:

[CSEBase]/[function_block_application]/hasInterface

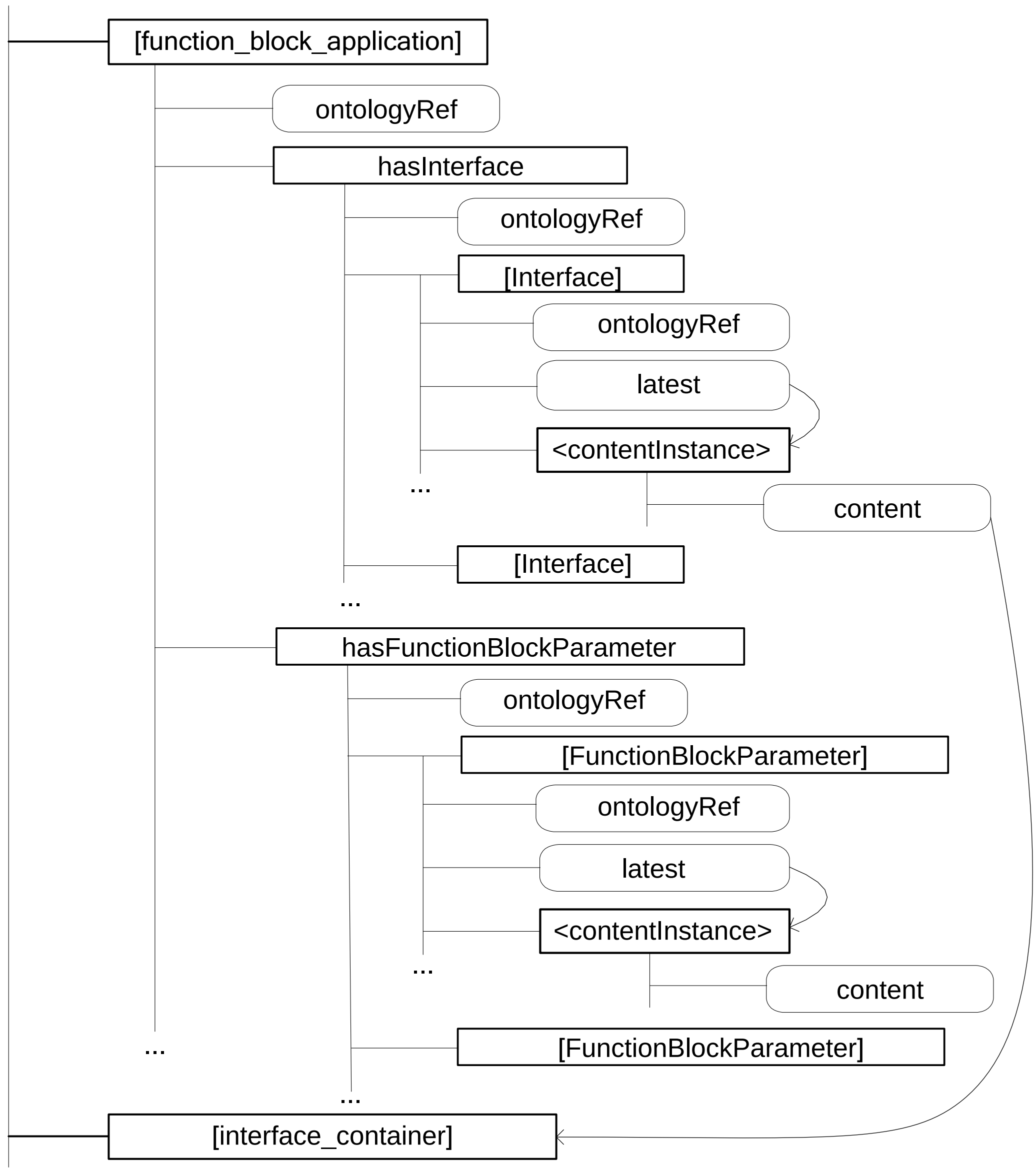


Figure 8.6.5.4.1-1

8.6.5.4.2 Parameters of the Function Block Application

The Function Block Application resource contains a sub resource of type <container> with the fixed name hasFunctionBlockParameter that contains the parameters for that Function Block.

The *ontologyRef* attribute of the hasFunctionBlockParameter container has a value of:

http://[oneM2M_ontology]#hasFunctionBlockParameter

or a more specific variant that is a sub-property of the hasFunctionBlockParameter property in the oneM2M ontology.

This hasFunctionBlockParameter container in turn has sub-containers [FunctionBlockParameter] that each contain the value of that parameter. The ontologyRef attributes of the [FunctionBlockParameter] container and its contentInstances are built analogously to clause 8.4.5.2.2.

8.6.5.4.3 Functional description of the Function Block Application

When a new Function Block Application has been created by the non-oneM2M Device Application then the Function Block Application:

1) creates new Interface containers for the Function Block;

2) creates new contentInstances in its hasInterface container that contains references to the Interfaces.

8.6.5.5 Interface container

8.6.5.5.0 Introduction

The Interface Container is modelled as a resource [interface_container] of Resource Type *container*. An Interface Container is created by a Function Block Application.

The URI used to access this resource has the following format:

[CSEBase]/[interface_container]

e.g.: IN-CSEID.m2m.myoperator.org/CSERoot/mySensorInterface

The Interface Container is the oneM2M representation of the interfaces (data fields and methods of function blocks of non-oneM2M devices:

- providing access to the data fields and/or methods of the interface;
- providing access to any parameters that are directly related to the interface.

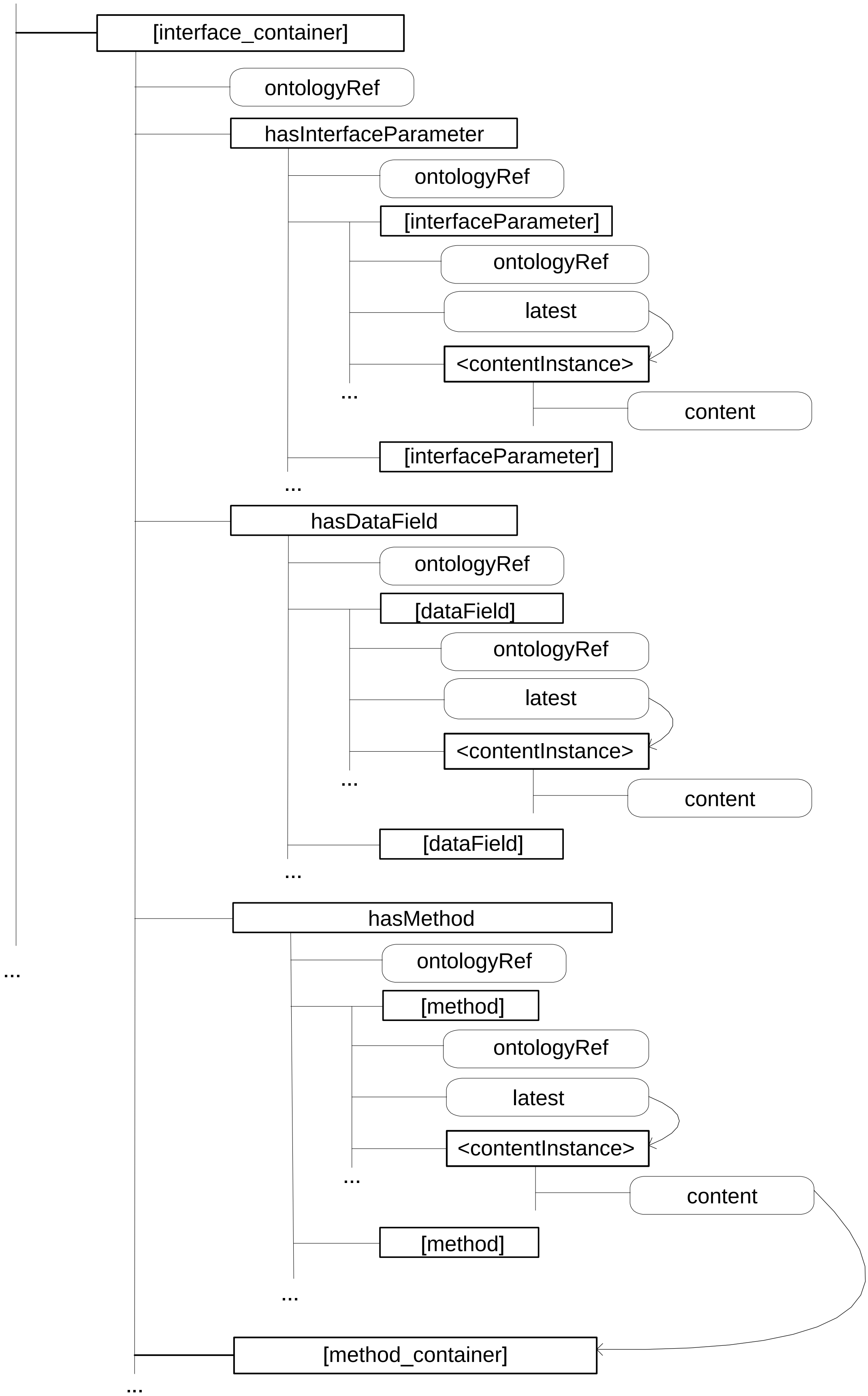

[interface_container]
ontologyRef
hasInterfaceParameter
ontologyRef
[interfaceParameter]
ontologyRef
latest
<contentInstance>
...
content
[interfaceParameter]
...
hasDataField
ontologyRef
[dataField]
ontologyRef
latest
<contentInstance>
...
content
[dataField]
...
hasMethod
...
ontologyRef
[method]
ontologyRef
latest
<contentInstance>
...
content
[method]
...
[method_container]
...

Figure 6

The ontologyRef attribute of the [interface_container] resource has a value of:

http://[oneM2M_ontology] #Interface

or a technology specific variant that is a sub-class of the oneM2M_ontology, e.g.:

http://[ZigBee_ontology] #ZigBee_Interface

8.6.5.5.1 Linking the Interface to its Data Fields

The Interface Container resource contains a sub resource of type <container> with the fixed name hasDataField that mirrors the relation *hasDataField* between the concepts *Interface* and *Data Field*.

The URI used to access this <container> resource has the following format:

[CSEBase]/[interface_container]/hasDataField

e.g.: IN-CSEID.m2m.myoperator.org/CSERoot/mySensorInterface/hasDataField

The ontologyRef attribute of the hasDataField container has a value of:

http://[oneM2M_ontology] #hasDataField

or a more specific variant that is a sub-class of the *hasDataField* class in the oneM2M_ontology.

This *hasDataField container* in turn has sub-containers [DataField] whose *contentInstance* contains the value of the Data Field. The ontologyRef attributes of the [DataField] container and its contentInstances are built analogously to clause 8.4.5.2.2.

The URI used to access a container resource whose contentInstance contains the value of a DataField has the following format:

[CSEBase]/[interface_container]/hasDataField/[dataField]

e.g.: IN-CSEID.m2m.myoperator.org/CSERoot/mySensorInterface/hasDataField/myDataField

8.6.5.5.2 Linking the Interface to its Methods

The [interface_container] resource contains a sub resource of type <container> with the fixed name hasMethod that mirrors the relation *hasMethod* between the concepts *Interface* and *Method*.

The URI used to access this <container> resource has the following format:

[CSEBase]/[interface_container]/hasMethod

e.g.: IN-CSEID.m2m.myoperator.org/CSERoot/mySensorInterface/hasMethod

The ontologyRef attribute of the hasMethod container has a value of:

http://[oneM2M_ontology] #hasMethod

or a more specific variant that is a sub-class of the *hasMethod* class in the oneM2M_ontology.This hasMethod container in turn has sub-containers [Method] whose contenInstances each holds a reference to a sub-container of the [Interface] container that represents one *Method* of the *Interface*.

The URI used to access a <container> resource whose contentInstance contains the reference to a [Method] container has the following format:

[CSEBase]/[interface_container]/hasMethod/[Method]

e.g.: IN-CSEID.m2m.myoperator.org/CSERoot/mySensorInterface/hasMethod/readTemperatureMethod

8.6.5.5.3 Parameters of the Interface

The [interface_container]resource contains a sub resource of type <container> with the fixed name hasInterfaceParameter that contains the parameters for that interface.

The URI used to access this container resource has the following format:

[CSEBase]/[interface_container]/hasInterfaceParameter

e.g.: IN-CSEID.m2m.myoperator.org/CSERoot/mySensorInterface/hasInterfaceParameter

The ontologyRef attribute of the hasInterfaceParamter container has a value of:

http://[oneM2M_ontology] #hasInterfaceParameter

or a more specific variant that is a sub-property of the *hasInterfaceParameter* property in the oneM2M_ontology.

This hasInterfaceParameter container in turn has sub-containers [interfaceParameter] whose contentInstances each contain the value of the respective parameter. The ontologyRef attributes of the [interfaceParameter] container and its contentInstances are built analogously to clause 8.4.5.2.2.

8.6.5.5.4 Methods of the Interface

[Interface] container that represents one *Method* of the *Interface*

Each [Method] sub-container of the [Interface_container] resource contains a sub resource of type <container> with the fixed name hasMethodParameter that contains the parameters for that Parameter of the method.

The URI used to access this container resource has the following format:

[CSEBase]/[interface_container]/hasMethodParameter

e.g.: IN-CSEID.m2m.myoperator.org/CSERoot/mySensorInterface/hasMethodParameter

The ontologyRef attribute of the *hasInterfaceParamter* container has a value of:

http://[oneM2M_ontology]#hasMethodParameter

or a more specific variant that is a sub-property of the *hasMethodParameter* property in the oneM2M_ontology.

This *hasMethodParameter* container in turn has sub-containers [MethodParameter] whose contentInstances each contain the value of the respective parameter. The ontologyRef attributes of the [MethodParameter] container and its contentInstances are built analogously to clause 8.4.5.2.2.

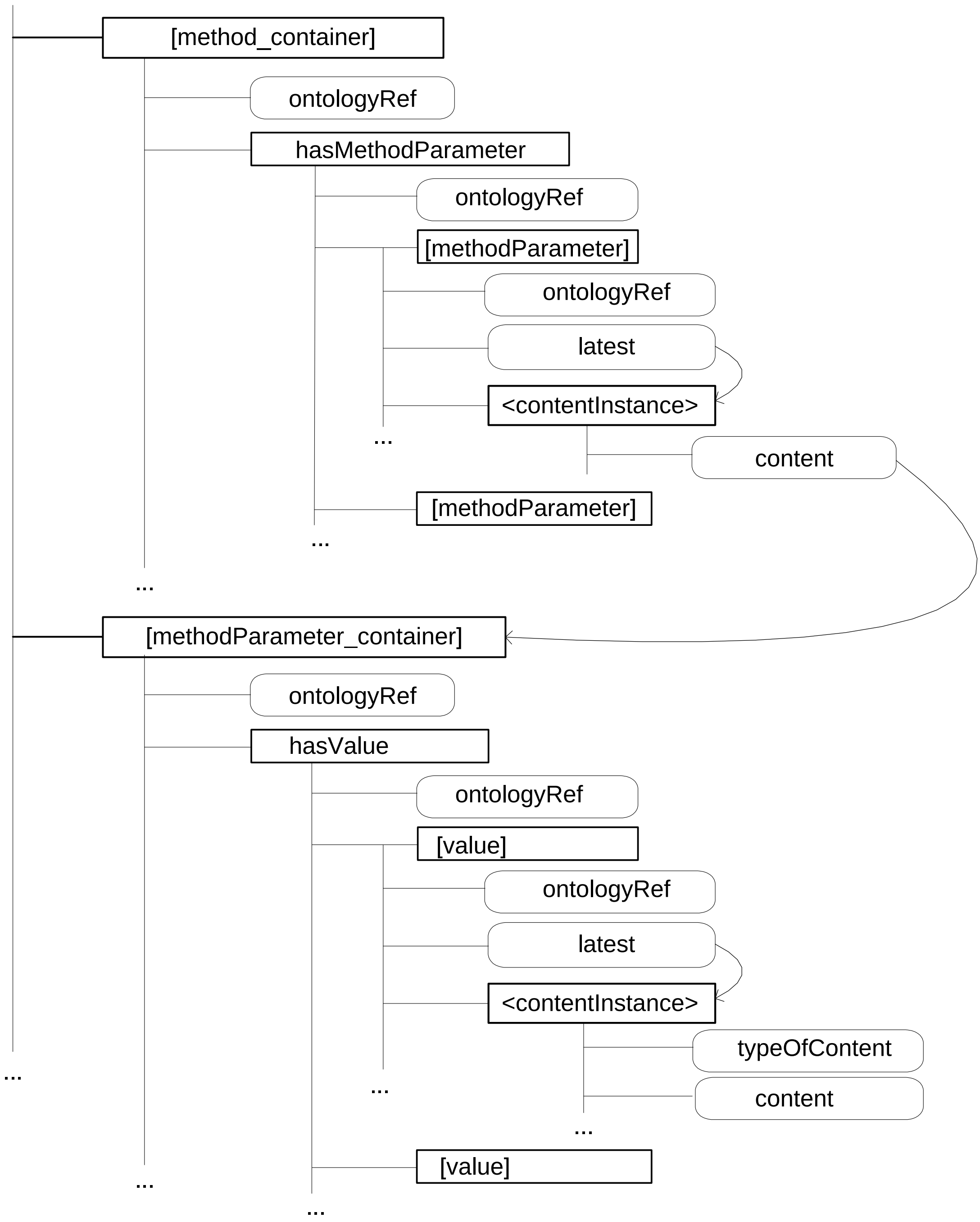


Figure 8.6.5.5.4-7

9 Conclusions

In oneM2M Release 1 the present document "Study on Abstraction and Semantics Enablements" resulted in a set of new use cases that rely on semantic support by oneM2M (annex A). In addition the study collected Abstraction technologies and Technologies for a Semantic M2M System (clauses 6 and 7). Clause 8 contains a proposal how the oneM2M System can be enhanced by including additional semantic capabilities. It contains design principles for modelling Devices, Things and their associations. Involvement of device manufacturers can be achieved through "Device Type Templates". Clause 8.4 proposes some specific modeling aspects for interworking with non-oneM2M devices and networks in support of Abstraction and Semantics in oneM2M.

In oneM2M Rel-1 only a very limited functionality that uses semantics is feasible. The only normative work related to semantics that had been performed was the introduction of the "ontologyRef" attribute for Application Entities (AEs), Containers and their Contentnstances.

Clause 8.4 proposes a way how using this "ontologyRef", together with appropriate ontologies that describe non-oneM2M Devices and Area Networks, interworking with those non-oneM2M Devices and Area Networks can already be facilitated in Release1. It provides a 'cook book' how Interworking Proxy Application Entities and related oneM2M resources can be constructed.

Thus, while not being normative in Rel-1, the method described in clause 8.4 can be used as a guideline for specifying a defined interworking with non-oneM2M solutions. It is proposed that in future releases this method for interworking is being refined and will become normative.

In addition it is recommended that the work on the TR continues beyond Rel-1 and leads to normative work that will allow for more dynamic usage of semantic information like semantic discovery, reasoning or mash-ups.

Annex A: Use Cases

A.1 An example of Home Environment Monitoring Service using semantic mash-up

A.1.1 Description

Semantic mash-up provides functionalities to support new services through the creation of new virtual entities, which do not exist in physical world, by obtaining semantic information through semantic descriptions from existing M2M resources in the M2M System.

Semantic mash-up function in the M2M system may have the following advantages:

- Communication efficiency: By using virtual entities created through mash-up, M2M Applications can obtain necessary information by using only a single query to M2M system. It reduces communication overload between the M2M System and the applications.
- Reusability: Virtual entities created by mash-up can be used by multiple M2M applications. It can improve a reusability of information.
- Authentication/security: When a mash-up needs information of entities residing in several M2M systems, authentication/security issues can be solved by M2M systems rather than applications.

For mash-up, abstract entitiy is defined as follows:

- **Abstract entitiy:** a resource represented in the M2M System through the abstraction of either a physical entitiy or a functionality implemented as a software.

Virtual entitiy is a new resource created by a mash-up of multiple abstract entities. Additionally, it also includes a composite virtual entitiy created by the mash-up of either other abstract entities or existing virtual entities. It is manipulated as a general M2M resource.

Virtual entities can provide new information which the existing resources do not contain.

In general, the virtual entities are created in the M2M System by a query from a M2M Application. They can be created through the composition of other existing virtual entities as well as physical and abstract entities. The M2M System manages the created virtual entities.

For example, if a user in a home requests home environment information like Discomfort Index (DI) or Air Pollution Index (API), new virtual entitiy (i.e. 'Home Environment Management') is created through mash-up of data from home appliances (e.g. heater, air conditioner, humidifier, air cleaner, etc.) equipped with environment sensors (e.g. sensors for temperature, humidity, CO2 level, VOC(Volatile Organic Compound) level, etc.) in the home. The virtual entitiy-'Home Environment Management' provides users with DI or API calculated using average values of temperature, humidity, CO2 level or VOC level based on collected data from various environment sensors.

A.1.2 Source (as applicable)

Modacom (TTA)

A.1.3 Actors

- M2M Application: An application to provide a M2M application service based on M2M resources to M2M application service users.
- M2M System: A system to provide M2M service functions.
- Physical Device: A physical M2M appliance equipped with environment sensors (e.g. fan/heater, air conditioner, composite sensor, humidifier, air cleaner, etc.).

A.1.4 Pre-conditions

- A M2M System has capabilities for semantic processing.
- Physical entities and abstract entities for home appliances equipped with environment sensors are registered in a M2M System.
- A M2M resource has semantic description for semantic based searching and discovery.

A.1.5 Triggers (if any)

None.

A.1.6 Normal Flow (as applicable)

Figure A.1 shows the procedure for creation and execution of a virtual entitiy for the request in case that a M2M Application sends a semantic query for DI or API.

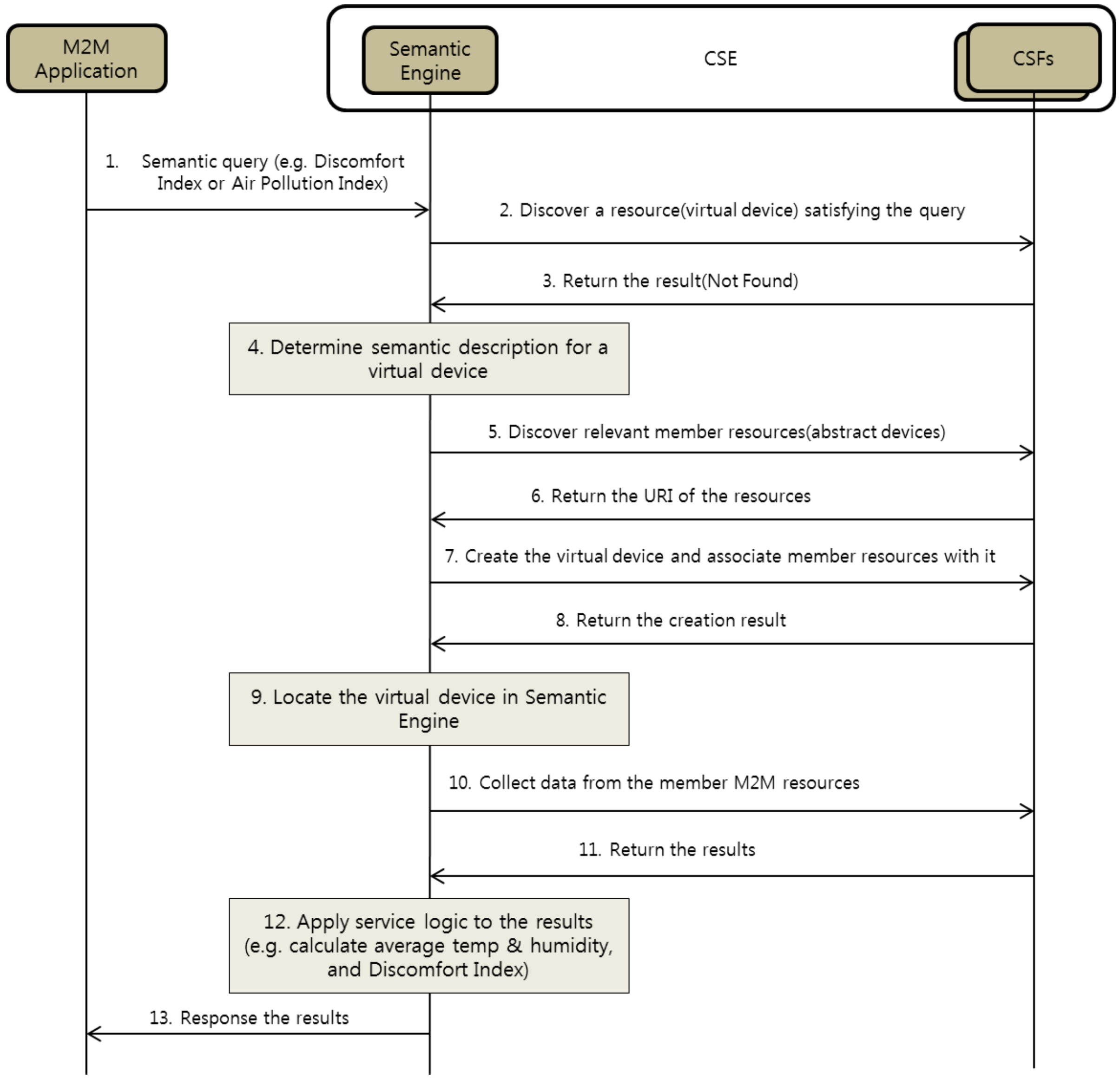


Figure A.1.6-1

1) A M2M Application sends a semantic query to a semantic engine in a M2M System (e.g. What's DI or API inside home?).

2) The Semantic Engine discovers virtual entitiy which can meet the semantic query in a CSE.

3) The CSFs return the result that there is no appropriate resource (Not Found).

4) The Semantic Engine determines semantic description to create a virtual entitiy (e.g. i) information of temperature and humidity required for calculating DI, ii) the method for calculating DI from data on temperature and humidity, etc.).

5) The Semantic Engine discovers related member resources (i.e. abstract entities).

6) The CSFs return URIs of discovered member resources.

7) The Semantic Engine requests to create a virtual entitiy and associate member resources with the virtual entitiy.

8) The CSFs return information for created virtual entitiy.

9) The Semantic Engine starts to run the virtual entitiy.

10) The Semantic Engine collects M2M data based on information from member resources of the virtual entitiy (e.g. values of temperature and humidity obtained from sensors in a home, etc.).

11) The CSFs return the result.

12) The Semantic Engine applies a service logic using the collected values (e.g. the calculation of average temperature and humidity in a home, the calculation of DI value, etc.).

13) The Semantic Engine returns the result to the M2M Application (e.g. the current DI value inside home).

A.1.7 Post-conditions (if any)

None.

A.1.8 High Level Illustration (as applicable)

In case that a M2M Application requests the information for DI or API, a M2M System creates a new virtual entitiy (i.e. 'Home Environment Management') through mash-up of related data after analysing the request and identifying required data. DI and API are created as new attributes inside the 'Home Environment Management' virtual entitiy. To find a DI value, a Semantic Engine inside the M2M System calculates average values of temperature and humidity from the data obtained through mash-up. After that, the DI value calculated from the average values is provided to the M2M Application. Similarly to DI, the API value is also calculated through mash-up of data for CO2, VOC level and is provided to the M2M Application.

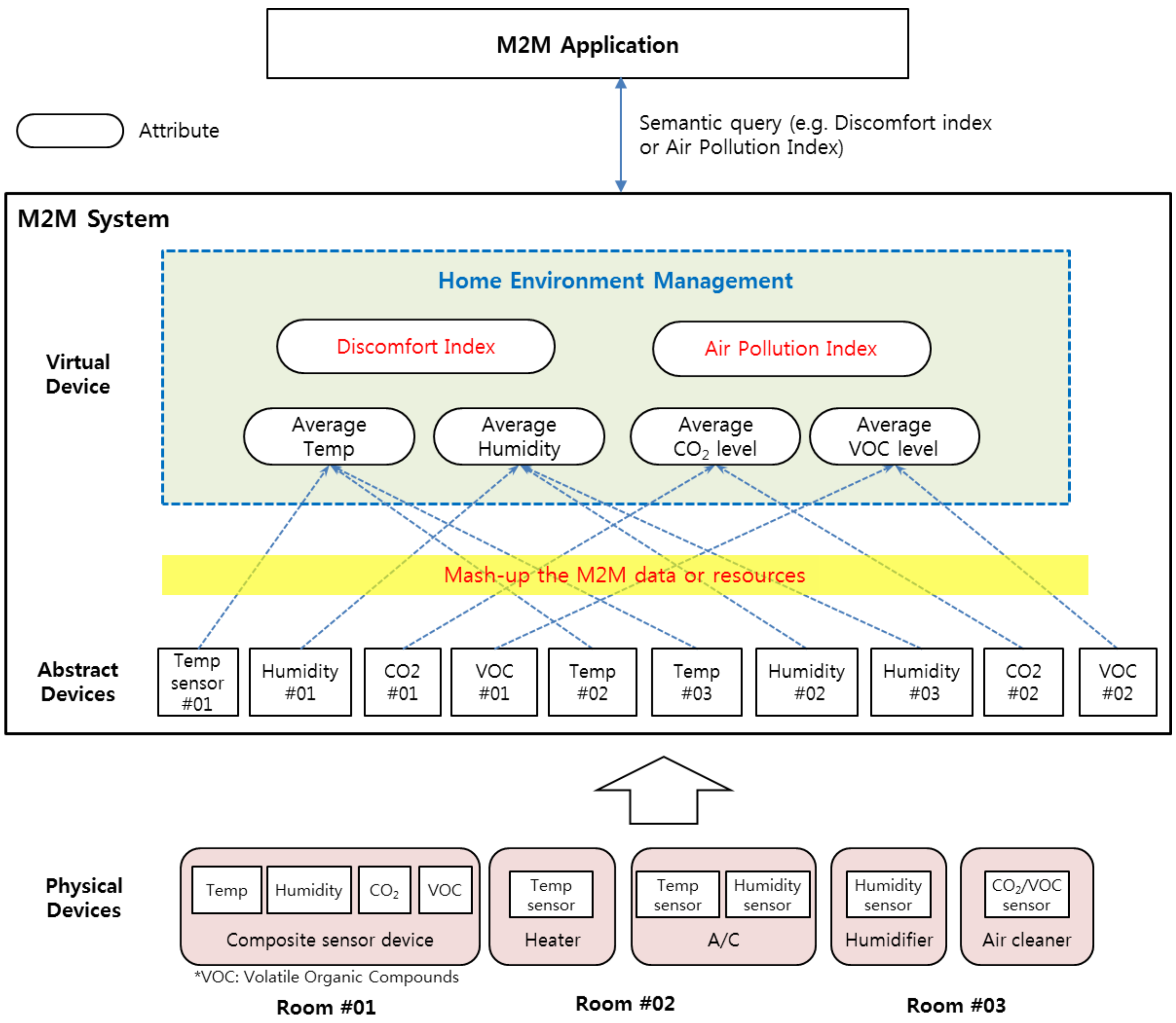


Figure A.1.8-1

A.2 Semantic Home Control

A.2.1 Description

The Semantic Home Control use case has been described in the oneM2M Use cases collection (see [i.25], clause 9.6). The complete use case description will not be repeated here. Instead the semantic aspects of it will be detailed. This includes an example of how the semantic aspects could be modelled and how the use case could be realized on this basis.

In the use case, there are two applications, a Building Management System (BMS) and a Home Energy Management System (HEMS). The BMS has knowledge about all structural elements of the building, i.e. the apartments, rooms, doors, windows, etc. as well as equipment installed in the house like heaters, air conditioning systems, etc. The HEMS configures itself for a given apartment based on the information available in the BMS. This means it has to find out about the rooms and the heaters and air conditioning systems deployed there in order to control the temperature in the apartment.

In the following, an example is given on how the use case could semantically be modelled based on an OWL [i.22] ontology. The example is used to illustrate the semantic approach - it is not claimed to be complete and there are surely other modelling options.

Figure A.2.1-1 shows the semantic concepts of Thing Type, i.e. describing real world things, modelled as ontology concepts. The relation between buidling and apartment, i.e. hasApartment, and the relation between apartment and room, i.e. hasRoom, are modelled as object properties of the ontology and are shown as dashed arrows in figure A.2.1-1.

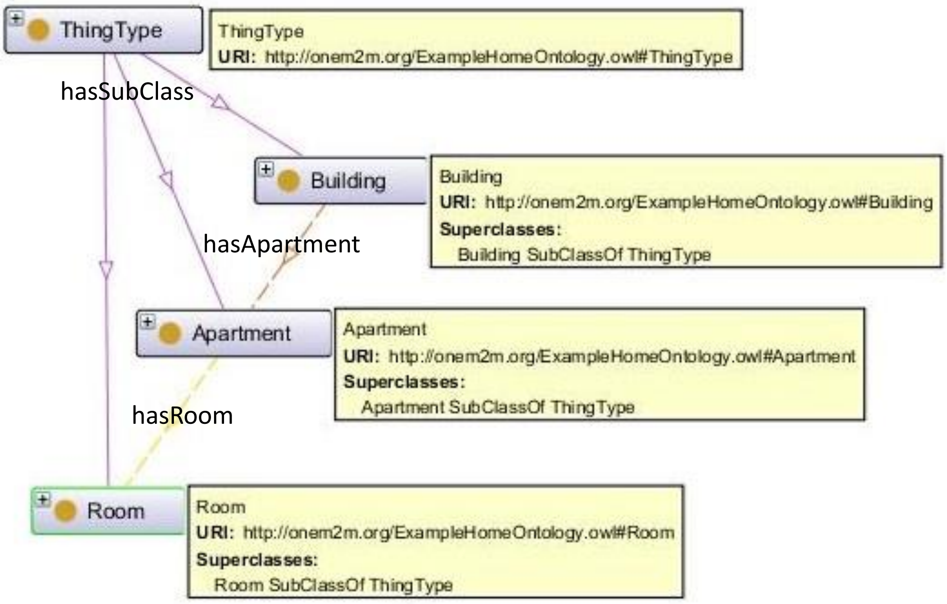


Figure A.2.1-1: Thing Types describing the structural elements of the building

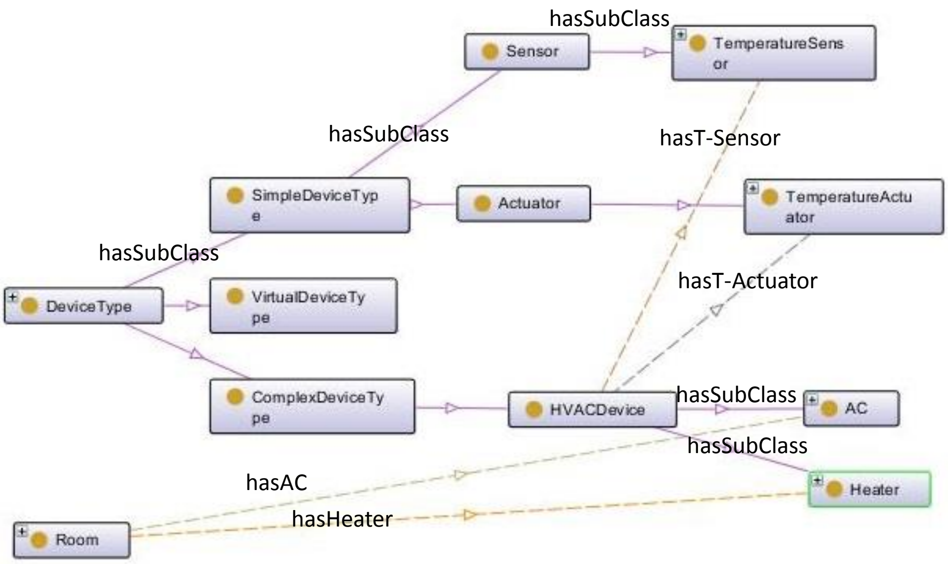


Figure A.2.1-2: Device Types describing the devices relevant for the HEMS

Figure A.2.1-2 shows the concepts of DeviceType, which are relevant for the HEMS. The Device Type has SimpleDeviceType, ComplexDeviceType and VirtualDeviceType as subclasses. A SimpleDeviceType can either be a Sensor or an Actuator. Relevant for HEMS are the TermperatureSensor and TemperatureActuator types. ComplexDeviceTypes represent more complex devices, which may contain simple devices. For the HEMS case, there are HVAC (heating, ventilation, air conditioning) devices, which contain temperature sensors and temperature actuators. The specific devices used are heaters and ACs (air conditioning systems). They inherit their relations to termperature sensors and actuators from the HVAC Device Type. VirtualDeviceTypes represent virtual devices, i.e. entities that can be accessed in the same way as real devices, but only consist of software. Virtual devices can be used to provide processed information, e.g. an average calculated from the output of a number of physical devices.

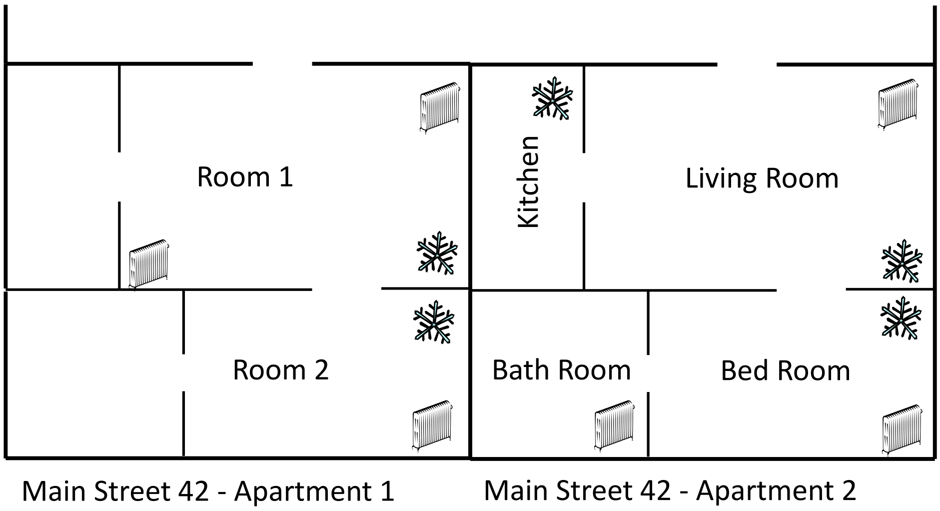


Figure A.2.1-3: Visualization of example instances for HEMS use case.

Figure A.2.1-3 shows a visualization of example instances for the HEMS use case. Two apartments have been modelled with different levels of details. In Apartement 1 only the two main rooms are covered, whereas Apartment 2 provides the details for all the rooms. Icons also indicate which rooms where the heaters and ACs are installed.

Figure A.2.1-4 shows the ontology model corresponding to the visualization shown in figure A.2.1-3.

In addition to relationships among types it shows instantiation of the individual types (as "hasIndividual" relationship).

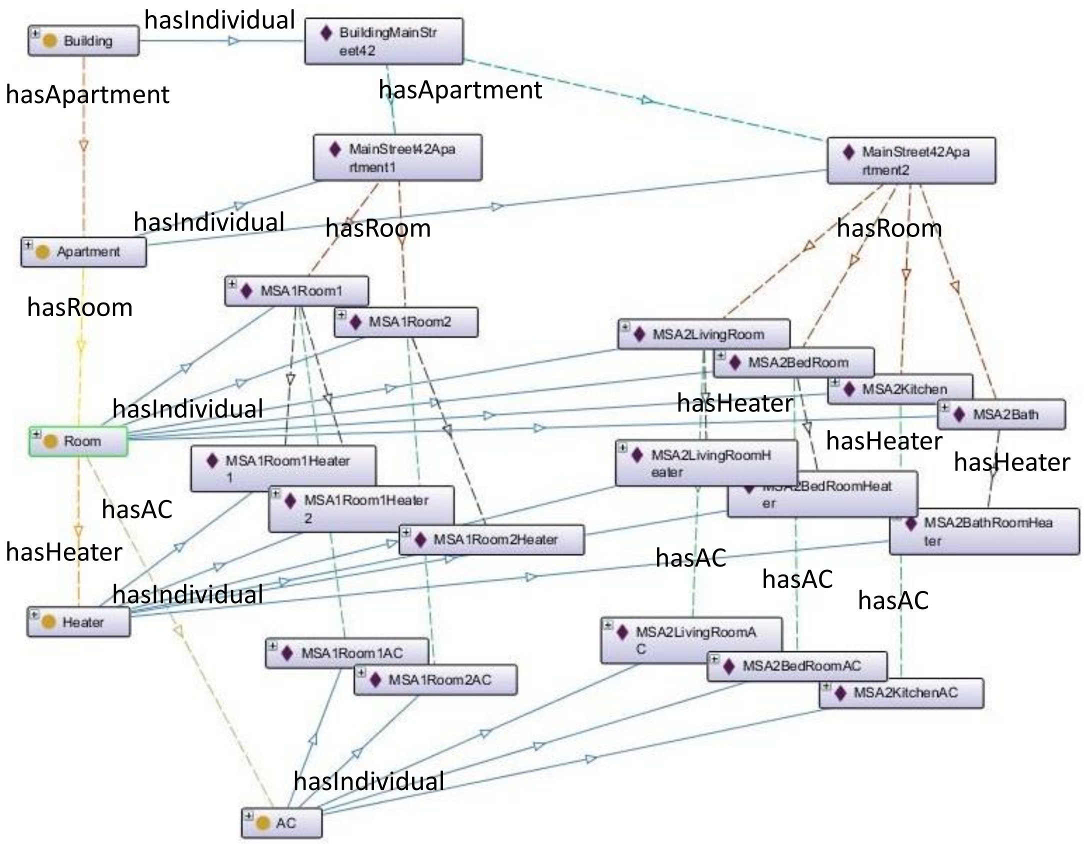


Figure A.2.1-4: Ontology model with instances

Given that the Ontology shown above is available in a triple store [i.24], the following SPARQL [i.23] queries can be executed to find the rooms and devices that the HEMS application needs in order to control the temperature in apartment 2 of building Main Street 42.

```
PREFIX home: <http://onem2m.org/ExampleHomeOntology.owl#>
SELECT ?room
    WHERE {     home:MainStreet42Apartment2 home:hasRoom ?room .
        }
```

Table A.2.1-1

room
MSA2BedRoom
MSA2LivingRoom
MSA2Bath
MSA2Kitchen

With the following query, the heaters and ACs for the respective rooms can be found:

```
PREFIX home: <http://onem2m.org/ExampleHomeOntology.owl#>
SELECT ?room ?ac ?heater
    WHERE {     home:MainStreet42Apartment2 home:hasRoom ?room .
        OPTIONAL { ?room home:hasAC ?ac }
        OPTIONAL { ?room  home:hasHeater ?heater }
        }
```

Table A.2.1-2

room	ac	heater
MSA2BedRoom	MSA2BedRoomAC	MSA2BedRoomHeater
MSA2LivingRoom	MSA2LivingRoomAC	MSA2LivingRoomHeater
MSA2Bath		MSA2BathHeater
MSA2Kitchen	MSA2KitchenAC	

Given the ACs and the Heaters, the respective temperature sensors and temperature actuators can be found, e.g.

```
PREFIX home: <http://onem2m.org/ExampleHomeOntology.owl#>
SELECT ?temp_sensor ?temp_actuator
    WHERE {     OPTIONAL { home:MSA2BedRoomHeater home:hasTemperatureSensor ?temp_sensor }
        OPTIONAL { home:MSA2BedRoomHeater home:hasTemperatureActuator ?temp_actuator }
        }
```

Table A.2.1-3

temp_sensor	temp_actuator
MSA2BedRoomHeaterTemperatureSensor	MSA2BedRoomHeaterTemperatureActuator

A.2.2 Source

NEC (ETSI).

A.3 Gym Use Case

A.3.1 Description

In the gym use case, as shown in figures A.3.1-1 and A.3.1-2, a gym local area network is set up to provide capabilities enabled locally or, by a Service Provider, via a Service Layer, with which the gym CSE registers at set-up. Devices available in the area network host various applications and are used to enable services. For example, treadmill applications and ambient sensor applications discover the local network and register with it.

The local area network capabilities are used by the users/gym members through devices like a phone, which discover the network and connect to the Service Layer when the member/user arrives. The member also uses devices such as a watch to collect biometric data (pulse, blood pressure, etc.) and feed it to the phone, which might use it within different applications. In this use case an application from the insurance company gathers data to give him points for staying in shape, and another gym-specific application is used to find and schedule available training devices (such as treadmills) or as personal trainer to monitor goals.

In order for the gym application on the phone able to discover and select available training devices, their semantic descriptions should be known to the gym application.

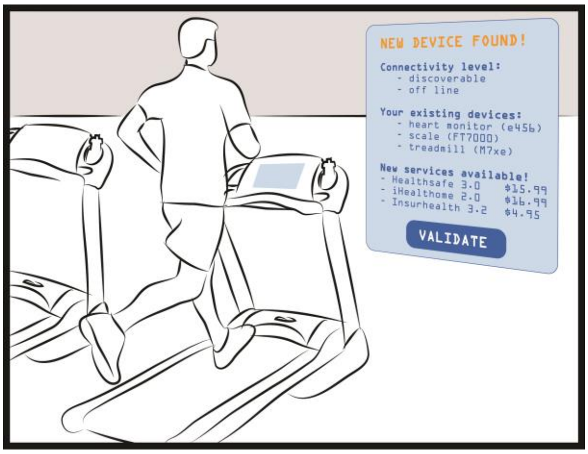


Figure A.3.1-1: Gym Use Case

Similarly, the health insurance application on the phone collects biometric measurements from the watch. It also collects the data from the gym ambient sensors (e.g. temperature, humidity), and the finished gym training program from the used training device via the gym CSE. The health score of the user can be calculated only based on the collected data and its associated semantics.

The health insurance application on the phone shares its information with the health insurance company application, which can determine and adjust the user's insurance premium based on it. In order to let the health insurance company application have the same interpretation of the data, the semantics of those resources need to be appropriately added and enabled.

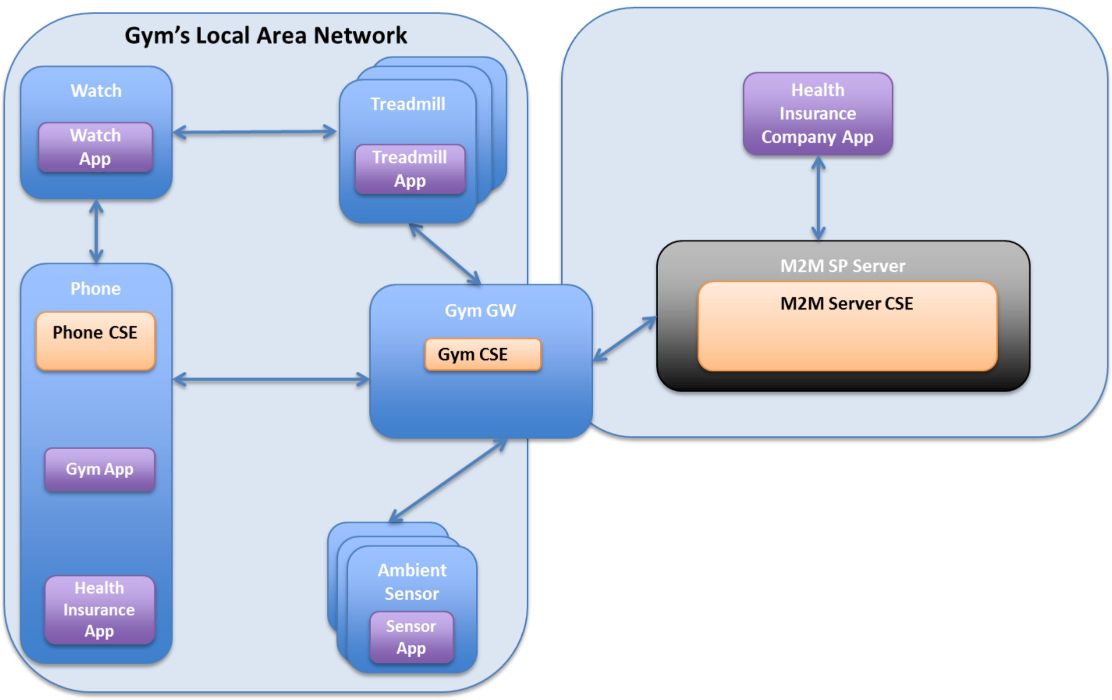


Figure A.3.1-2: Gym Use Case Architecture

Figure A.3.1-3 gives the ontology model of the gym use case. Figure A.3.1-4 gives the ontology model of the treadmills in the gym.

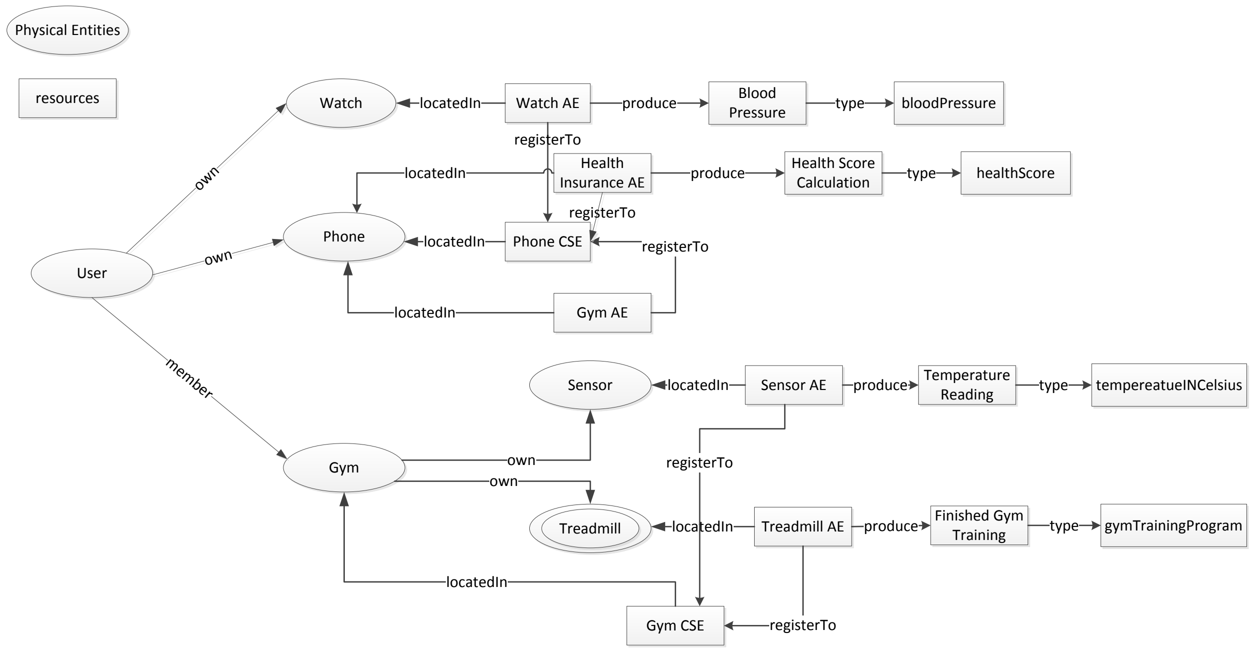


Figure A.3.1-3: Ontology Model of Gym Use Case

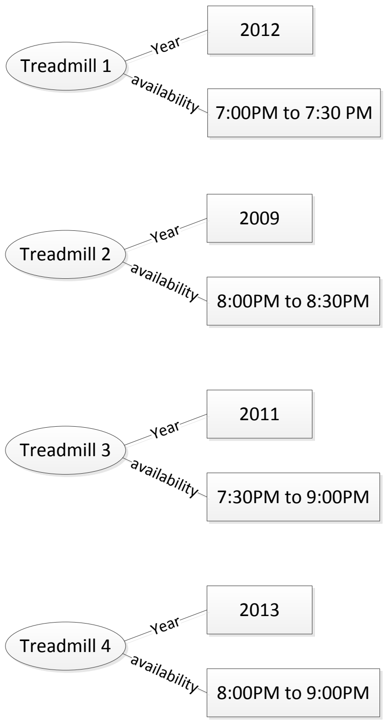


Figure A.3.1-4: Ontology Model of Treadmills

A.3.2 Source

InterDigital Communications.

A.3.3 Actors

- Gym area network: The local area network in the gym that provides Internet connection to the gym members' personal devices.
- Ambient sensors: The ambient sensors that are deployed in the gym to measure the environmental parameter in the gym, such as the temperature, humidity.
- Training devices: The training devices that are deployed and available in the gym for the gym members, such as treadmills, elliptical.
- Gym member: The members of the gym who consumes services provided by the gym owners, such as the training devices, gym applications, network connection to their personal wireless devices.
- Gym member's personal wireless devices: The personal wireless devices that are carried with the gym member into the gym, such as phone, watch, which can measure the person's vitals, collect the person's training data, and calculate the person's health score.
- Health insurance company: The health insurance company provides service to its members for any health insurance related information as well as keeps track of the members' gym training history and health scores.
- M2M service platform: Gym and health insurance system host which collects, stores, manages and processes data from the user' devices, the training devices used by the user. It accepts registration from the service providers (gym owner, health insurance company) and the users' personal devices. It accepts/enforces policies governing data exchange and access from the M2M Service Providers.

A.3.4 Pre-conditions

The person has membership contract with the gym service provider, the health insurance provider. The gym service provider and the health insurance provider have business relationship with the M2M Service Provider.

The M2M Service Provider provides the mechanisms to govern the data exchange and access.

A.3.5 Triggers

A variety of triggers might be associated with the use case, for examples see flow.

A.3.6 Normal Flow

See figure A.3.8-3.

1), 4), 6), 9) AEs register to the CSEs: watch AE, health insurance AE, gym AE on the phone registers to the phone CSE; ambient sensor AEs and device AEs (e.g. treadmills) register to the gym CSE. The registration message includes semantics associated with each application.

2), 5), 7), 10) Applications create resources to store measurement data in the phone CSE and gym CSE. The semantics of the data needs to be published by the corresponding application as well, which may be linking to other resources on other locations such as the semantics repository. In order to understand it, the relevant resource representation needs to be retrieved by the phone CSE, gym CSE.

3), 8), 11) Semantics relevant resources to the newly created resources (AEs, measurement resources) are retrieved by the phone CSE, gym CSE and made available. Resources will be made available together with the context included in resource representation and associated semantics. The phone CSE and the gym CSE are able to parse and interpret the newly created resources, and support semantics-based query on those resources.

12) Registered applications request actions based on available resources. In this case the gym application on the phone can query which treadmill will be unoccupied for a certain period of time.

13) The gym CSE provides a value added service using semantics, for example through the treadmill availability query. From the OWL ontology shown in figure A.3.1-4 , the following SPARQL queries can be processed to find the treadmills that are available from 8:00PM to 8:30PM. The query and its result are shown in figure A.3.6-2 and table A.3.6-1.

```
PREFIX gym:<coap://example.org/gym.owl#>

PREFIX  xsd:  <http://www.w3.org/2001/XMLSchema#>

SELECT ?treadmill

WHERE {

?treadmill gym:availability  ?availability .

FILTER(?availability  >= 20:00:00^^xsd:time   &&  ?availability  <= 20:30:00^^xsd:time ) .

}
```

Figure A.3.6-2: Query

Table A.3.6-1: Query Result

treadmill
treadmill2
treadmill3
treadmill4

14) The response to the semantics-based query is returned; in this case the query response may include the URI of the resources and associated semantics.

15) The requesting application uses the response to the query, including the semantics associated with the resources in the response.

16) An application subscribes to data stored on the gym CSE and the phone CSE. Subscribed data such as ambient sensor measurements, user biometrics and finished gym training program are collected.

17), 20) The subscribing application retrieves the relevant resource representation as well as its associated semantics from the gym CSE and the phone CSE. The resource associated semantics may link to other resources on the semantics repository, which need to be retrieved as well in order to provide data analysis. The semantic descriptions are parsed and interpreted.

18), 21) The application uses the data and associated semantics for analysis and computation.

19) An application subscribes to data stored on the gym CSE and the phone CSE. Subscribed data such as the health score computed by the health insurance application, the user finished gym training program.

A.3.7 Post-conditions

N/A.

A.3.8 High Level Illustration

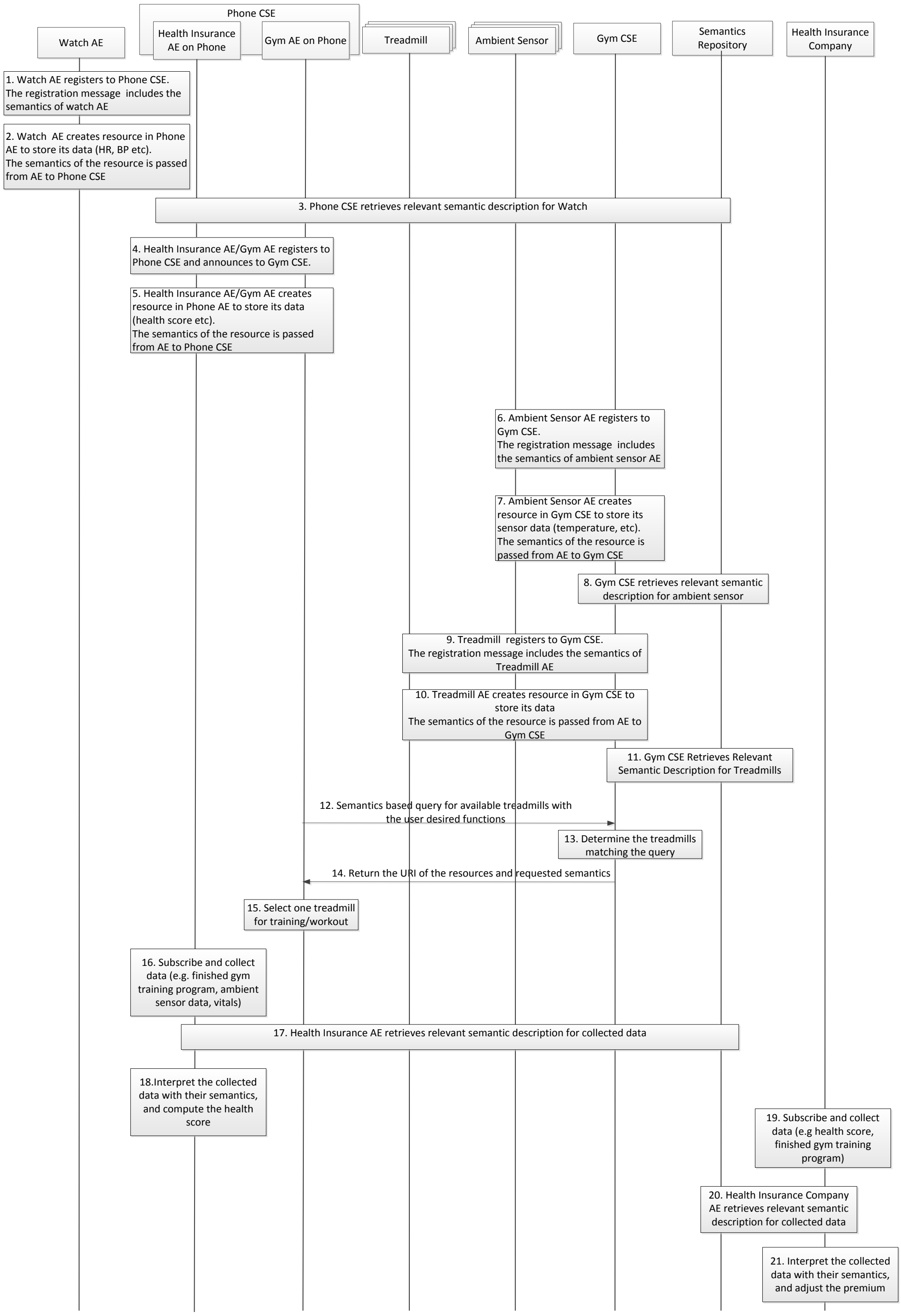


Figure A.3.8-3: Normal Flow

A.3.9 Potential requirements

- The M2M System shall provide the capability to publish semantic descriptions.
- The M2M System shall support parsing and interpreting semantic descriptions.
- The M2M System shall support resource discovery based on semantics.

References:

- This use case refers to documents in [i.19], [i.26], [i.27], [i.28], [i.29], [i.7], [i.30].

A.4 Intelligent Alarm Service using Semantic Discovery and Mash-up

A.4.1 Description

Intelligent Alarm Service, as shown in figure A.4.1-4, provides a user with a context-aware alarm service by a smart phone. The smart phone searches neighbouring nodes in order to find and select possible nodes to participate in the Intelligent Alarm Service. The selected nodes make a group. The group members consist in a smart phone, a smart TV, a smart audio device, and a camera in figure A.4.1-4. They collaborate with each other based on service logic information provided by a service provider. It also provides a mash-up service according to user's behaviour patterns.

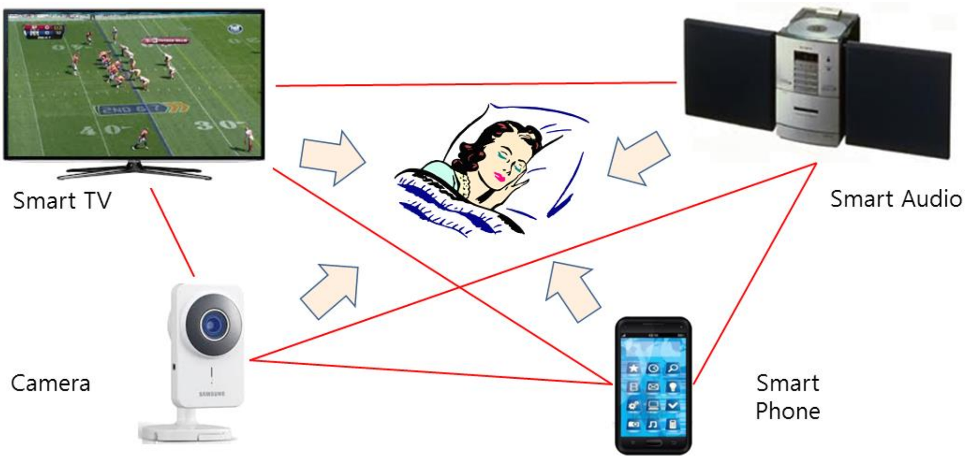


Figure A.4.1-4: Intelligent Alarm Service Use Case

The Intelligent Alarm Service can be provided based on two scenarios below.

Case#1: In the case of the user set up the alarm service on the smart phone in advance:

- At a designated time, the smart phone rings an alarm and a camera monitors the user and notifies the neighbor nodes of the user's status.
- If the user does not wake up the camera sends the status information to the smart TV and the audio device to request for the cooperation.
- Then the smart TV turns on and audio device plays the user's favorite music.

Case#2: In the case of the user did not set up the alarm service on the smart phone in advance:

- The smart phone checks the user's schedule information in the phone.
- Base on the morning schedule, the smart phone rings an alarm at the suitable time automatically.

Figure A.4.5 shows the system architecture of the Intelligent Alarm Service.

All nodes register abstract entities (such as AEs, abstract entities representing physical entities) to local ASN-CSEs. And then all nodes also register abstract entities to semantic information repository via a semantic engine in MN/IN-CSE including the device and service profile information. For the discovery and mash-up of the associated entities, the IntelligentAlarmService, a virtual entity in the smart phone, requests a sematic query to the semantic information repository, obtains the associated abstract/virtual entities resources and analyses the results to select participating nodes.

The associated abstract entities (IntelligentAlarmService, HumanMonitoringServcie, SmartTVService, SmartAudioService) in the participating nodes (i.e. Camera, smart TV, smart audio device) fulfil the announcement and subscription of the associated resources in ASN-CSEs. The nodes notify the events with each other when related events occur in the nodes and provide the mash-up service through cooperation among the node's own services.

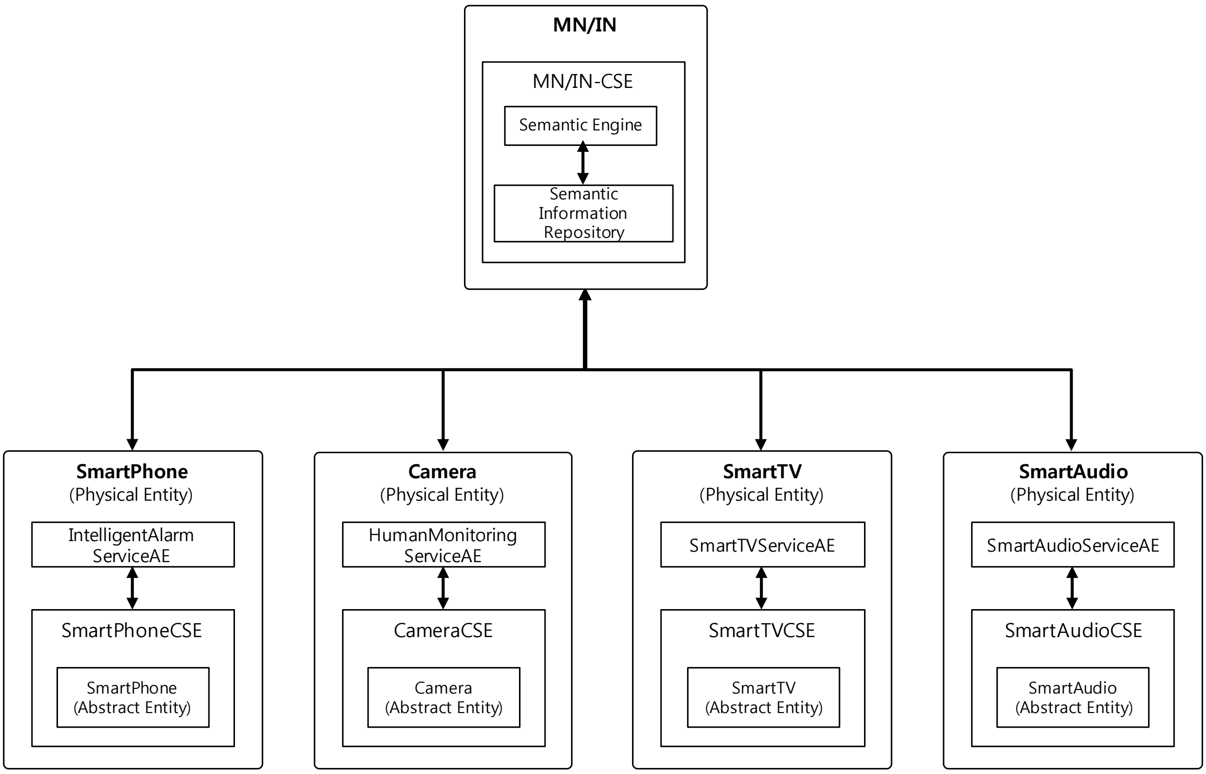


Figure A.4.5: System Architecture of Intelligent Alarm Service Use Case

Figure A.4.1-3 shows the ontology model for the Intelligent Alarm Service Use Case in oneM2M system.

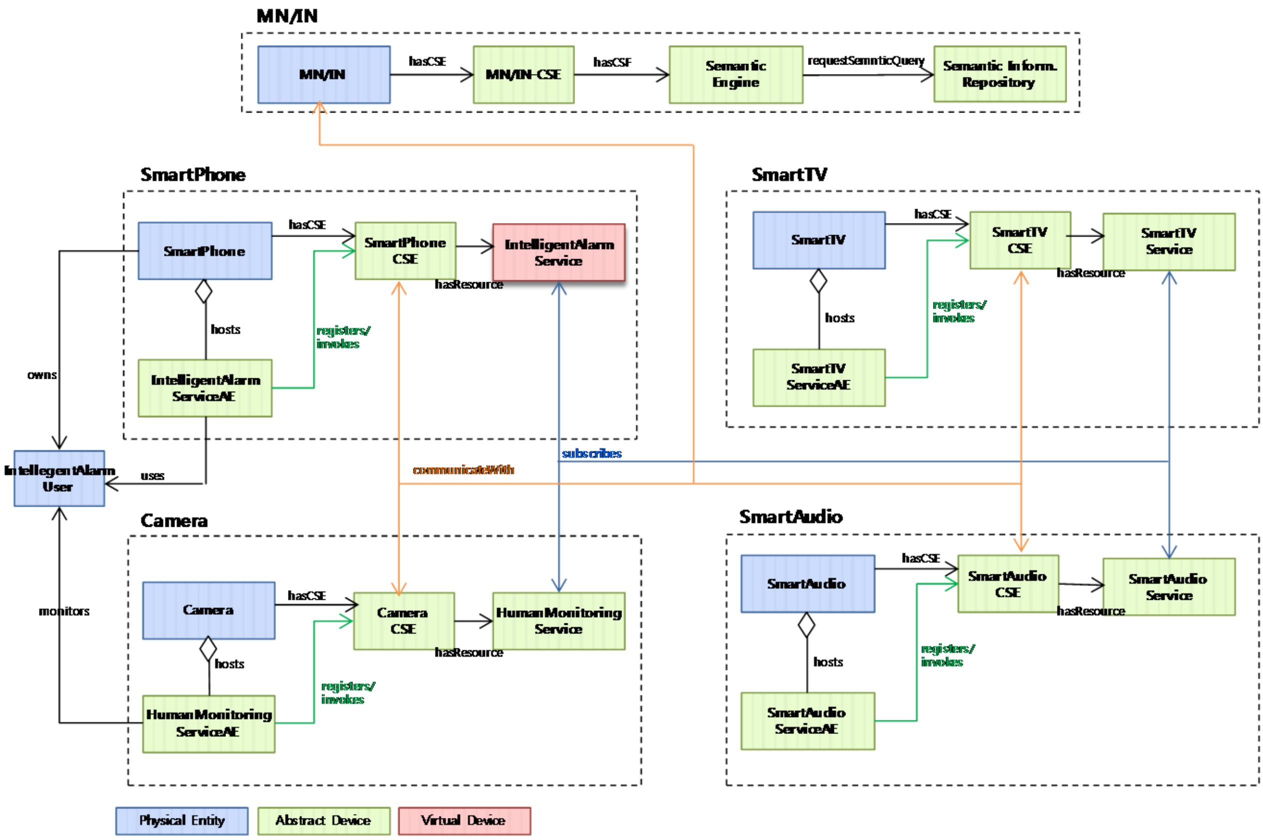


Figure A.4.1-3: Ontology model for the Intelligent Alarm Service Use Case

The oneM2M domain model consists of three primary classes whose definitions are as follows:

- Physical Entity: a tangible element that is intrinsic to the environment, and that is not specific to a particular M2M application in this environment. Depending on the environment, the physical entity may be a smart phone, a camera, a smart TV/audio, a piece of furniture, somebody, a room of a building, a car, a street of a city, etc.

- Abstract Entity: A resource represented in the M2M System through the abstraction of either a physical entity or functionality implemented as software.

NOTE 1: An Abstract Entity relates to "Thing" and "Thing Representation":

  - Thing: an element of the environment that is individually identifiable in the M2M system.

  - Thing Representation: It is the instance of the informational model of the Thing in the M2M System. A Thing Representation provides means for applications to interact with the Thing.

- Virtual Entity: A new resource created by a mash-up of multiple abstract entities. Additionally, it also includes a composite virtual entity created by the mash-up of either other abstract entities or existing virtual entities.

NOTE 2: An Abstract Entity relates to "Virtual Thing" as described in clause 7.2.1.3.

A.4.2 Source

Modacom (TTA).

A.4.3 Actors

- M2M Application: An application to provide a M2M application service based on M2M resources to M2M application service users.
  (i.e. IntelligentAlarmServcieAE, HumanMonitoringServiceAE, SmartTV/AudioServiceAE)

- M2M System: A system to provide M2M service functions.
  (i.e. SmartPhoneCSE, CameraCSE, SmartTV/AudioCSE, MN/IN-CSE)

A.4.4 Pre-conditions (if any)

N/A.

A.4.5 Triggers

The service is triggered when a user either set up the alarm or registers one's schedule information on one's smart phone with service logic for Intelligent Alarm Service.

A.4.6 Normal Flow

Figure A.6 displays the flow diagram of the Intelligent Alarm Service Use Case.

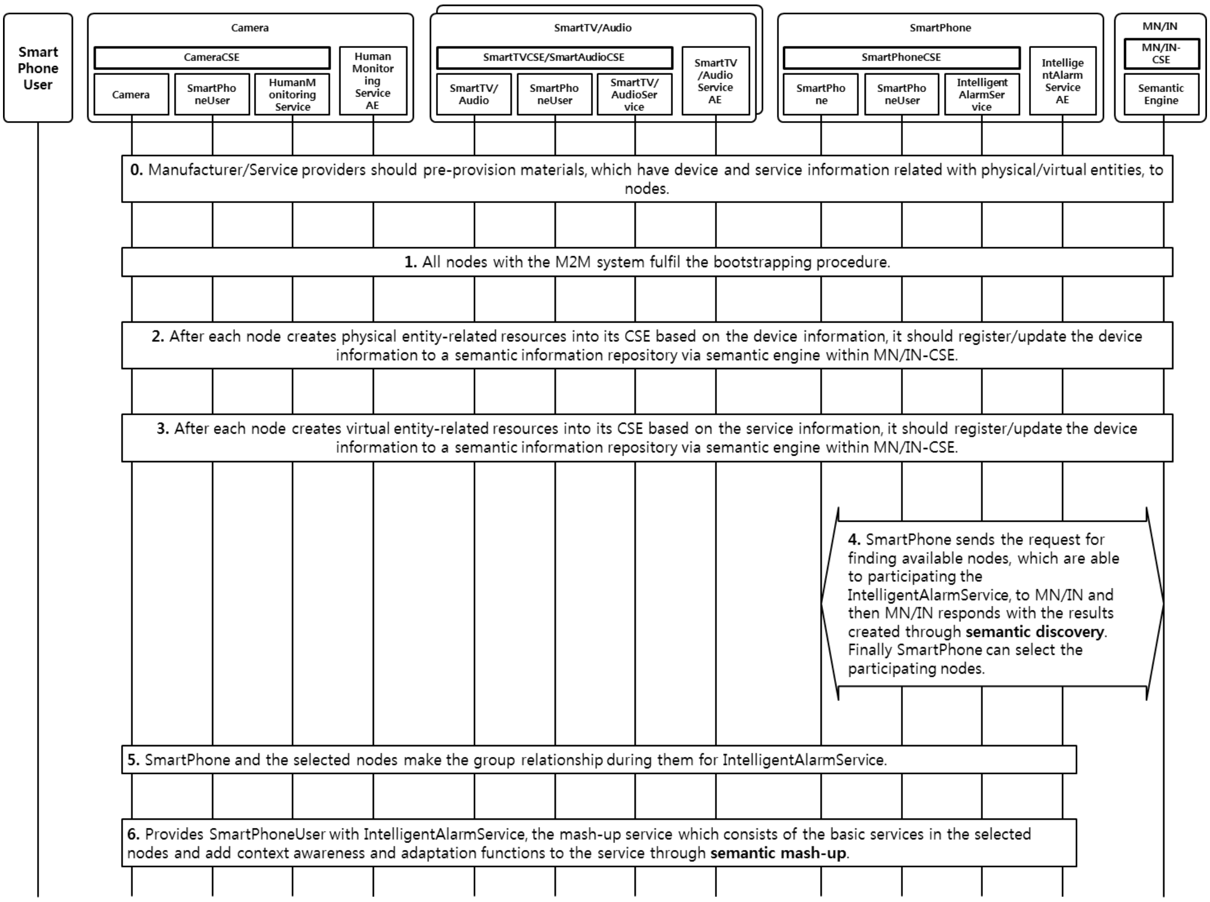


Figure A.6-1: High Level Flow Diagram of Intelligent Alarm Service

0) Nodes are provided with some pre-provisioned materials such as a semantic description containing the device and service information related with Physical/Abstract Entities.

1) The M2M System takes the procedure of bootstrapping and initialization.

2) After creating physical entity-related resources into their local CSEs with the device information, all nodes register or update their device information to a semantic information repository via a semantic engine within MN/IN-CSE (Middle Node/Infrastructure Node - Common Service Entity).

3) After creating virtual entity-related resources into their local CSEs with the service information, all nodes register or update their service information to a semantic information repository via a semantic engine within MN/IN-CSE.

4) The smart phone searches for the nodes which are possible to attend the Intelligent Alarm Service.

   a) After the smart phone extracts the information for discovering the nodes which are possible to join the Intelligent Alarm Service from the device and service information, it sends the request of semantic query to MN/IN.

   b) The MN/IN fulfils **the semantic discovery** before responding to the requester with the searching results of the associated nodes.

   c) Parsing and analyzing the results, the smart phone chooses the nodes possible to attend (i.e. camera, smart TV and smart audio).

5) With the selected nodes cooperating with each other, they organize the group to join the Intelligent Alarm Service.

   a) The selected nodes mutually register to each other.

   b) The selected nodes take the action of announcement to interesting resources. The smart phone announces SmartPhone/IntelligentAlarmService/SmartPhoneUser resoruces to other selected nodes, The smart TV and smart audio to SmartTV/SmartAudio/SmartTVService/SmartAudioService ones and the camera to Camera/HumanMonitoringService ones.

   c) The selected nodes subscribe to the interesting resources to be notified of the changes of ones. The smart phone, smart TV and smart audio subscribe to the HumanMonitroingService resource.

6) The selected nodes provide the Intelligent Alarm Service to SmartPhoneUser through **the semantic mash-up** and cooperatively perform their own basic service in accordance with the service information. The smart phone informs the selected nodes (i.e. camera, smart TV and smart audio) of the information of the alarm service related with SmartPhoneUser. The camera monitors the status of SmartPhoneUser and notifies the selected nodes of the events' information. According to the status information of SmartPhoneUser, the smart TV and audio analyzes the accumulated information on SmartPhoneUser (e.g. user's favorite channels and music) and provide the context-adaptive alarm service to SmartPhoneUser.

A.4.7 Post-conditions (if any)

None.

A.4.8 High Level Illustration (as applicable)

None.

A.4.9 Potential requirements (as applicable)

- The M2M System shall provide capabilities to represent device and service information using ontology for service discovery, mash-up and data analysis.

A.5 Semantic Home Automation Control

A.5.1 Description

This use case demonstrates the semantic home automation control system. The system consists of home automation APP, smart household appliances, and M2M service platform as shown in figure A.5.7.

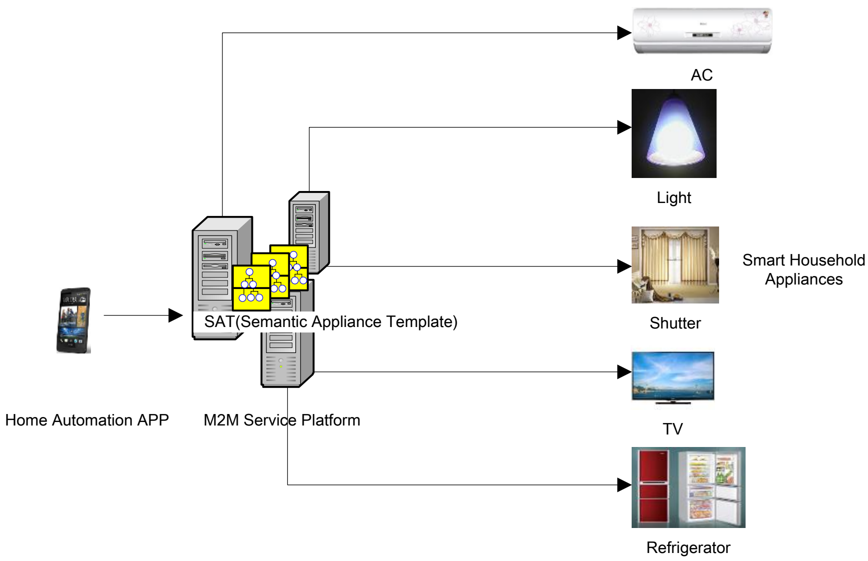


Figure A.5.7: Semantic Home Automation Control System

The system user can install the home automation APP on the user device e.g. the smart mobile terminal. The user device with the installed the home automation APP can control smart household appliances including AC, light, shutter, AC, TV and refrigerator, etc. The various appliances information including e.g. appliance ID, appliance type, operation and data types, etc. can be modelled as SAT (semantic appliance template). The user sends control command in the form of natural language to the APP, e.g. by texting "tune the air conditioner in the living room to 22 centigrade". The APP converts the natural language control command and generates e.g. SPARQL semantic query [i.23]. The APP then sends SPARQL semantic query to the M2M service platform. In this use case, the M2M service platform provides the semantic functionalities with which semantic query can be executed to locate the desired appliance, i.e. AC in the living room, and get the appliance semantic information including, e.g. appliance ID, appliance type, operation and data types from the SAT (semantic appliance template).

The APP can then generate the appliance control command by using the appliance semantic information and send the appliance control command to the desired smart household appliance.

A.5.2 Source

Haier (CCSA).

A.5.3 Actors

- Home Automation APP: is an M2M application for home automation control.
- M2M Service Platform: provides semantic appliance service for home automation control.
- Smart Household Appliance: is the household appliance with remote control feature.

A.5.4 Pre-conditions

The M2M service platform contains all the necessary information to manage the whole smart household appliances of the user if it is allowed. It in particular contains the details of the smart household appliances including the location of the appliances and their capabilities, etc.

The M2M APP can generate SPARQL semantic query to retrieve the semantic appliance information from the M2M service platform.

The smart household appliance can interact with the M2M service platform which can execute the appliance control command and send back the appliance status.

A.5.5 Triggers

None.

A.5.6 Normal Flow

Figure A.5.6-1 provides the basic flow of semantic home automation control.

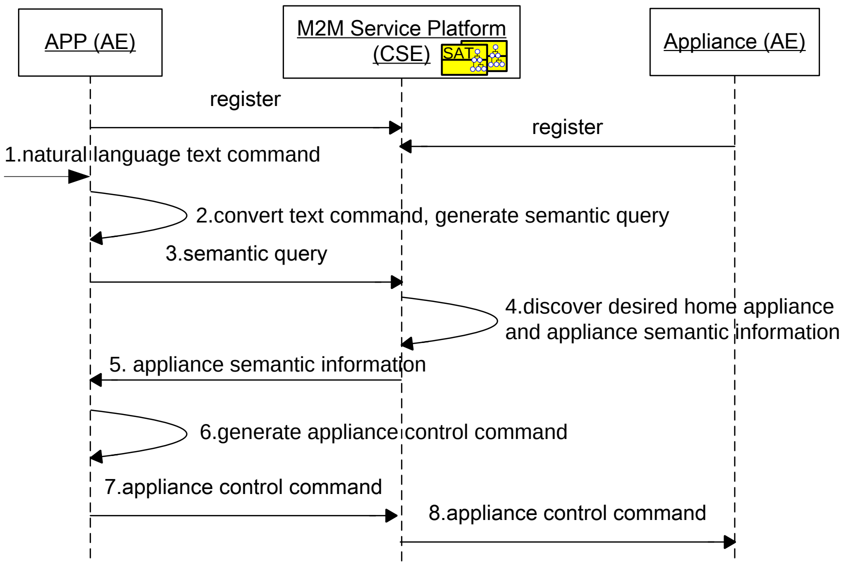


Figure A.5.6-8: Semantic Automation Home Control Flow

1) The APP receives the natural language control command, e.g. "tune the air conditioner in the living room to 22 centigrade".

2) The APP converts the natural language control command and generates semantic query e.g. SPARQL semantic query.

3) The APP sends semantic query to the M2M service platform.

4) The M2M service platform executes the semantic query to locate the desired appliance, i.e. AC in the living room, and get the appliance semantic information including e.g. appliance ID, appliance type, operation and data types from the appliance semantic description.

5) The M2M service platform sends the appliance semantic information to the M2M APP.

6) The M2M APP generates the appliance control command by using the appliance semantic information.

7) The APP sends the appliance control command to the M2M service platform.

8) The M2M service platform delivers the appliance control command to the AC.

A.5.7 Post-conditions

None.

A.5.8 High Level Illustration

This use case applies home automation ontology model for semantic information sharing. Figure A.5.8-1 illustrates the logical view of ontology-based model to specify the semantic information of smart household appliances and their relationships.

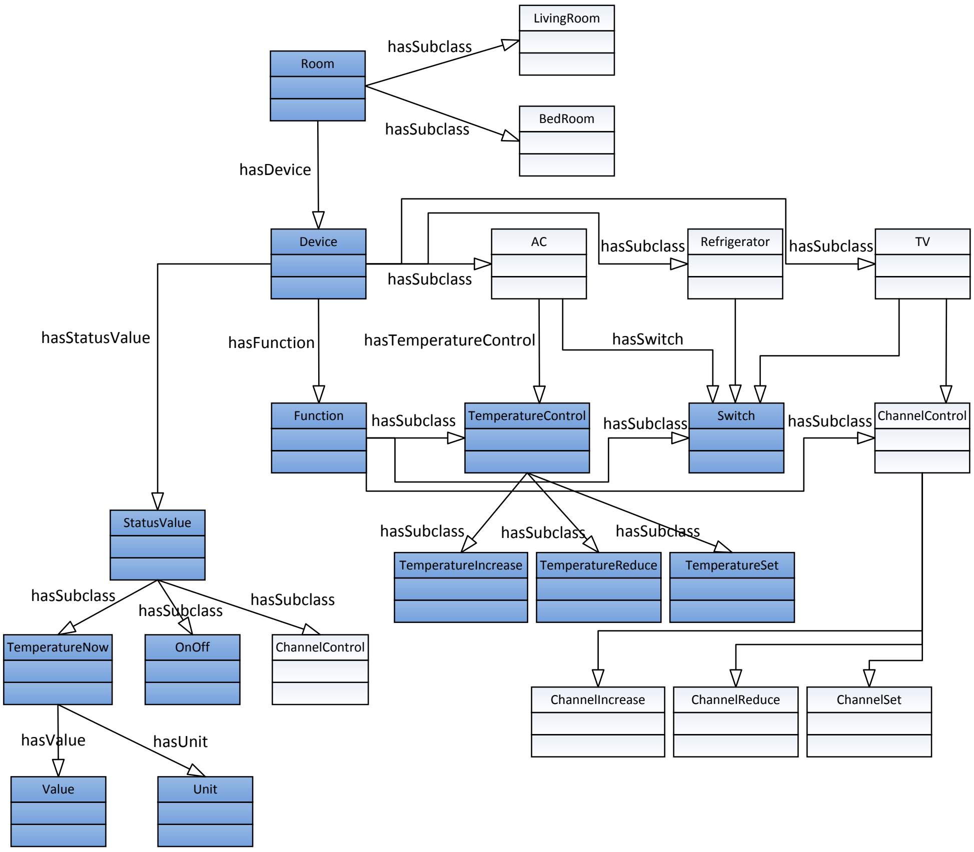


Figure A.5.8-9: Home Automation Ontology Model

The basic information of this ontology model is described as follows.

- AC is an appliance which has there functions including controlling temperature (TemperatureControl) and switch (Switch).
- TemperatureControl function consists of three sub-functions including temperature increasing (TemperatureIncrease), temperature reducing (TemperatureReduce) and temperature setting (TemperatureSet).
- Switch function set the AC into working time running mode or out of working time running mode.
- TemperatureIncrease sub-function increases AC temperature.
- TemperatureReduce sub-function reduces AC temperature.
- TemperatureSet sub-function set AC temperature.
- StatusValue is a record to represent the current appliance status including current temperature (TemperatureNow), on-off state (OnOff) and current channel (ChannelControl).
- TemperatureNow is a status value to record the current temperaturewhich consists of value and unit.

- OnOff is a status value to decide the working time of the AC.

Figure A.5.8-2 illustrates the instance of semantic home automation ontology according to the ontology-based model. Given the home automation ontology model, the APP can locate the desired appliance through semantic query.

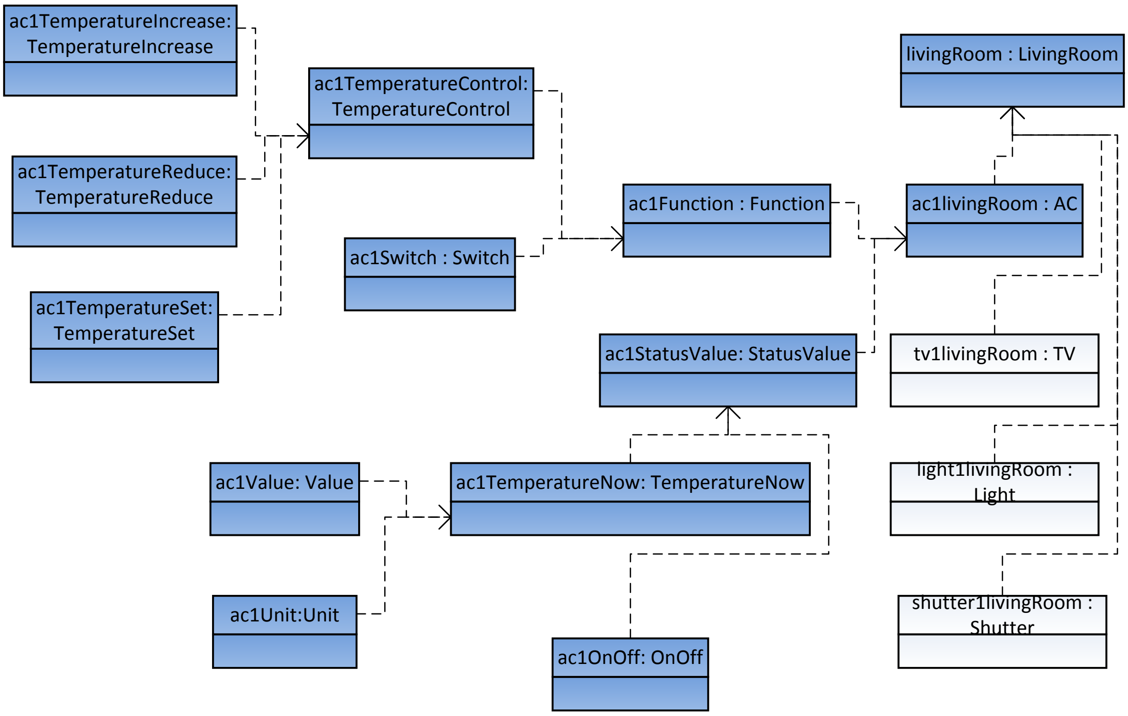


Figure A.5.8-10: Home Ontology Model with AC Instances

A.5.9 Potential requirements

The M2M system shall be able to support semantic modelling device template for diverse M2M devices (e.g. household appliances).

The M2M System shall support common ontology to model the semantic information of M2M devices and the real-world entities (e.g. rooms) that associate with M2M devices.

The M2M System shall support semantic query to enable the discovery of target M2M devices based on their semantic information.

A.6 Semantic smart building light control

A.6.1 Description

The smart building light control use case has been mentioned in the oneM2M use case collection (see [i.25], clause 6.1) from group management aspects. In this contributionthe smart building light control use case from semantic aspects is described. The light control for one target room is given as an example.

In the use case, all the smart building appliances are connected with M2M gateway. M2M gateway and Smart building control centre are connected with oneM2M platform. The light control application sends the semantic query request to the smart building control centre to find the desired appliances.

The following shows how the use case could semantically be modelled based on OWL [i.22] ontology. There are two ontologies involved in this use case. One ontology is a basic M2M ontology named ***ontology A***, which is referred by all actors (e.g. oneM2M platform, smart building control centre and light control application). The other ontology is a specific ontology on light domain named ***ontology B***, which is referred by light control application.

Note that ontology A and ontology B are compatible ontologies in this use case, which indicates that there exists corresponding relationships between the same concepts (that may corresponding different vocabularies) in ontology A and ontology B. The compatibility can be realized in the ways such has:

1) Ontology B refers to ontology A and use the vocabularies in ontology A to describe same concepts, or vice versa.

2) There are some triple stores [i.24] to express corresponding relationships (e.g. be equivalent to) between the vocabularies in ontology A and ontology B. Some techniques can be deployed for ontologies mapping

In this use case, the first way is adopted.

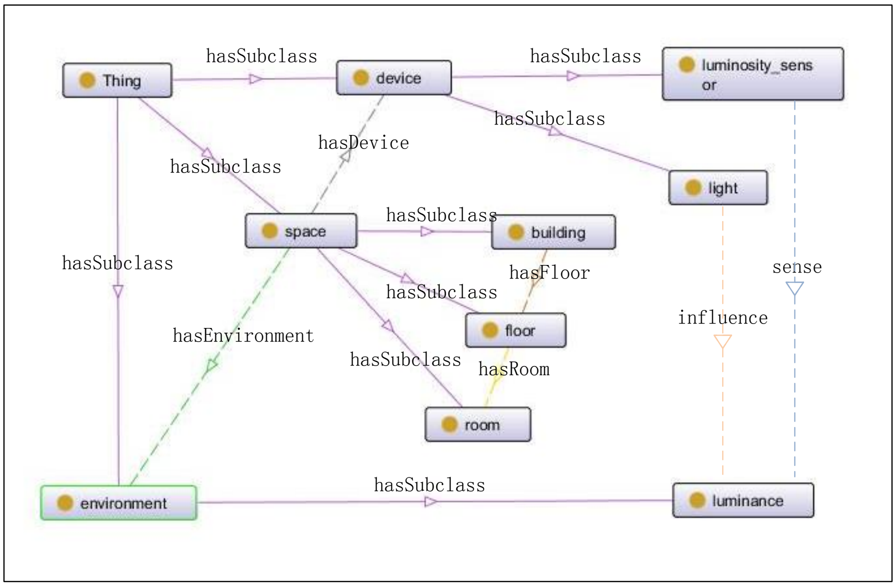


Figure A.6.1-11: Class Concepts in ontology A

The class concepts in ontology A for this use case are described in figure A.6.1-11, and the data properties model for class "light" and class "luminosity_sensor" in ontology A are described in figure A.6.1-12. The class concepts in ontology B for this use case are described in figure A.6.113. Some basic concepts in ontology B are referred from ontology A. The "brightness_adjustable_light" in ontology B is defined as the "light" which "brightness_adjustable" data property is "true", i.e. the domain of class "brightness_adjustable_light" is intersection of the domain of class "light" and the domain of the class where "brightness_adjustable" data property is "true", which is shown in figure A.6.1-14. Similarly, the "led_light" and "fluorescent_light" in ontology B are defined as the "light" which "kind" are "led" and "fluorescent" respectively.

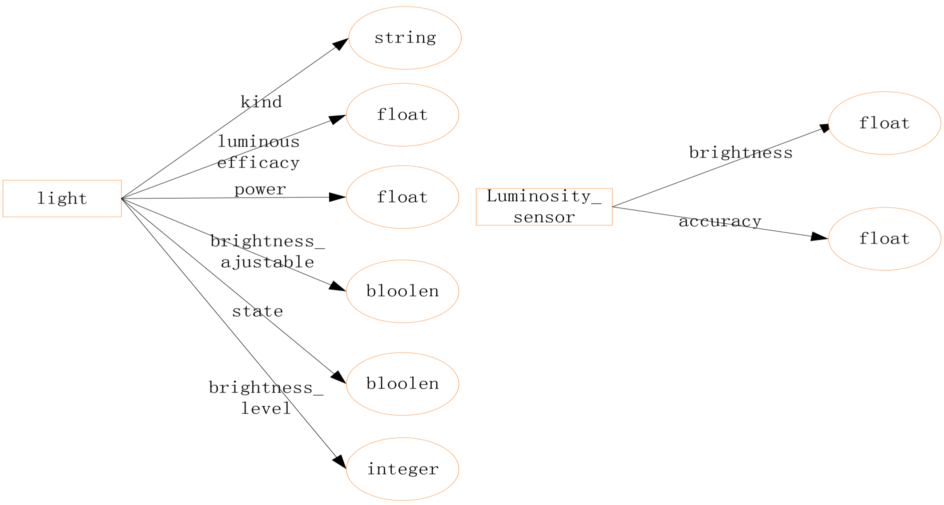


Figure A.6.1-12: data properties model in ontology A

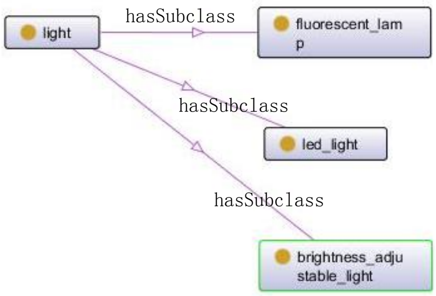


Figure A.6.113: Class related Concepts in ontology B

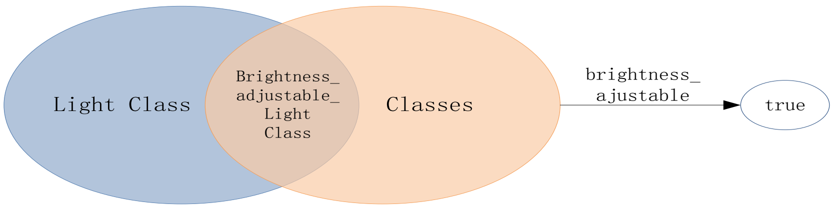


Figure A.6.1-14: intersection description for Class "brightness_adjustable_light"

Examples of instances related to this use case are described in figure A.15, where "type" means "instance of".

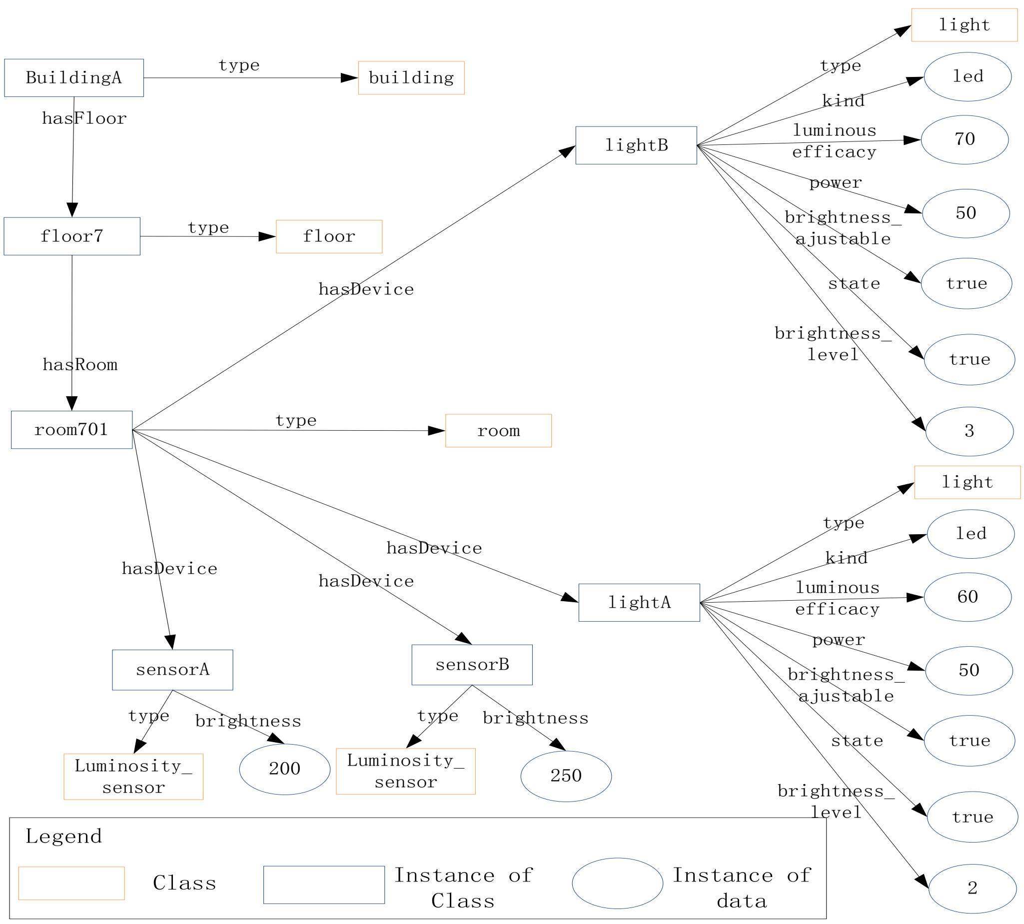


Figure A.15: Example of instances

A.6.2 Source

China Unicom.

A.6.3 Actors

- Light control application (developed by application provider).
- Smart building Control centre (operated by smart building service provider).
- M2M gateway (operated by M2M service provider).
- oneM2M platform (operated by M2M service provider).
- Smart building appliances (installed by smart building service provider).

A.6.4 Pre-conditions

- The smart building service provider establishes a business relationship with the M2M service provider so that the Smart building Control centre and Smart building appliances can use M2M gateway and oneM2M platform.
- All smart building appliances are registered in oneM2M platform, and smart building control centre can control smart building appliances via oneM2M platform.
- Ontology A is referenced by all actors, and Ontology B is referenced by light control application.

A.6.5 Triggers

- Light control application intends to control the light to adjust the brightness conditions in target room (perhaps when the motion sensor in that room detects that there are people entering that room).

A.6.6 Normal Flow

1) Since luminosity sensors can sense the brightness conditions of rooms, light control application sends semantic request to smart building control centre to find luminosity sensors in room701.

```
PREFIX building: <http://example.org /ontologyA.owl#>
Select ?luminosity_sensor ?brightness
WHERE { building:room701 hasDevice ?luminosity_sensor.
?luminosity_sensor a building: luminosity_sensor.
?luminosity_sensor building:brightness ?brightness}
```

2) Smart building Control centre forwards the semantic query requests to oneM2M platform.

3) oneM2M platform returns the URI of target luminosity sensor resources and the associated semantic information.

Table A.6.6-1: Search results of requested semantic query

luminosity_sensor	brightness
SensorA	200
SensorB	250

4) Light control application send requests to smart building control centre to create the group of the returned resources and to subscribe the brightness data of created group.

5) Smart building control centre forwards the group management requests and subscription requests to oneM2M platform.

6) The brightness data is notified to light control application.

7) Light control application finds that the brightness is a little lower than desired value, and intends to adjust the brightness of some lights instead of switching on more lights for power saving. The Led light is preferred, and the brightness level of led light should be lower than the highest level 5. So light control application sends semantic query request to smart building control centre to find the target brightness_adjustable light.

```
PREFIX building: <http://example.org/ontologyA.owl#>
PREFIX light:< http://example.org /ontologyB.owl#>
Select ?light ?brightness_level
WHERE { building:room701 hasDevice ?light.
?light a light: brightness_adjustable_light.
?light a light: led_light.
?light building:brightness_level ?brightness_level.
FILTER(?brightness_level < 5)}
```

8) Smart building Control centre forwards the semantic query requests to oneM2M platform.

9) Although the instances of "light" (e.g. "lightA" and "lightB") are semantically annotated based on ontology A which do not explicitly indicate that they are instances of "brightness_adjustable light" and "led_light", with the help of reasoning, oneM2M platform can still correctly return the URI of target light resources and the associated semantic information.

The following two ways can be adopted to support such operation.

1) Deriving implicit knowledge and changing the semantic annotation information of instances according to the concepts in referenced ontologies

EXAMPLE 1: The reasoning function can derive that lightA and lightB are instances of brightness_adjustable light and led_light, and can change the semantic annotation information of lightA and lightB as:

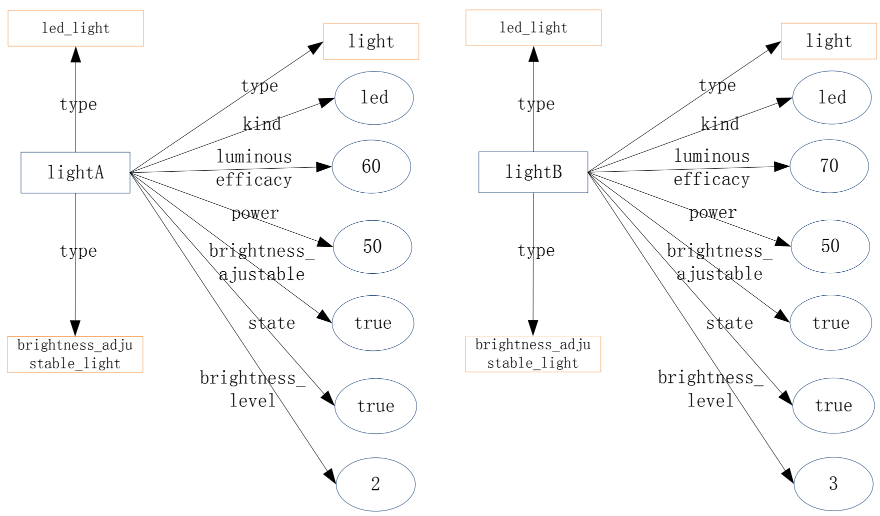


Figure A.6.16-1: examples of changed semantic annotation of instances

After changing the semantic annotation information, using the original semantic request can correctly find the target resources.

2) Changing the semantic query request according to the concepts in referenced ontologies.

EXAMPLE 2: According to the knowledges in ontology B, the reasoning function can change the semantic request as:

```
 PREFIX building: <http://example.org/ontologyA.owl#>
PREFIX light:< http://example.org /ontologyB.owl#>
PREFIX xsd:  <http://www.w3.org/2001/XMLSchema#>
Select ?light ?brightness_level
WHERE {{ building:room701 hasDevice ?light.
?light a light: brightness_adjustable_light.
?light a light: led_light.
?light building:brightness_level ?brightness_level.
FILTER(?brightness_level < 5)}
```

```
UNION
{ building:room701 hasDevice ?light.
?light a building: light.
?light building:brightness_level ?brightness_level.
?light building:brightness_adjustable ?brightness_adjustable.
?light building:kind ?kind.
FILTER(?brightness_level < 5 && regex(?kind,"led") && ?brightness_adjustable = "true"^^xsd:boolean)}
```

Using the changed semantic query request can correctly find the target resources.

Table A.4.6-2: Search results of requested semantic query

light	brightness_level
lightA	2
lightB	3

3) Light control application sends requests to smart building control centre to create the group of the returned resources.

4) Smart building control centre forwards the group management requests to oneM2M platform.

5) The Light control application sends the requests to adjust the brightness level of created light group.

A.6.7 Post-conditions

None.

A.6.8 High Level Illustration

Figure A.6.8-1 describes the deployment of all actors in this use case form high level aspects. The smart building appliances are connected with M2M gateways, and M2M gateways are connected with oneM2M platform via underlying network. The Smart building control centre is connected with oneM2M platform, and the light control applications are connected with smart building control centre.

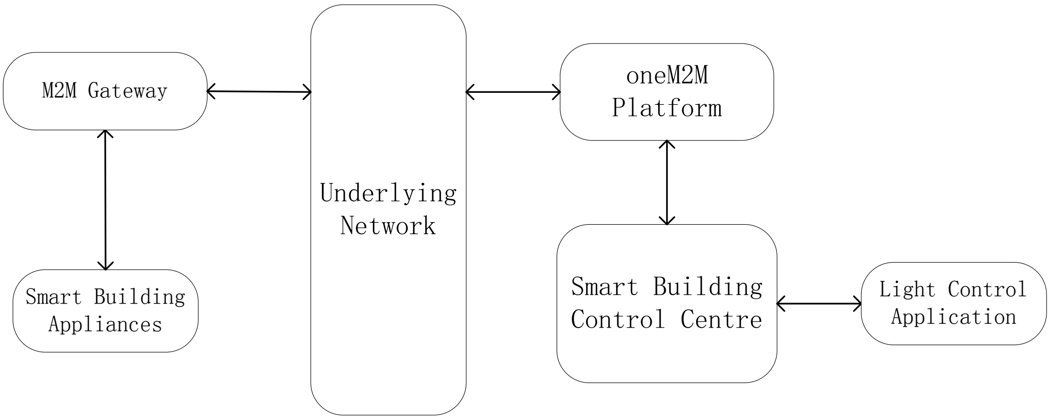


Figure A.6.8-1: connection relationships of actors

A.6.9 Potential requirements

- The oneM2M systems shall support the reasoning capability for deriving implicit knowledge from semantically annotated information according to referenced ontologies.

A.7 Smart Home load control use case

A.7.1 Description

This use case is a key constituent of home energy management. Fine-grain control of all electrical appliances, be they connected to a communication network or not, makes it possible for the home to be fully integrated in a smart grid, where supply does not merely follow variations of aggregate loads , but a two-way mutual adaptation is possible to support a much better overall efficiency and a better matching of available energy resources to the needs of consumers.

Two variants of this use case are, respectively, power-based or energy-based load shedding.

In power-based load shedding, a threshold on aggregate power may vary on different time-scales and requires instant adaptation by either shedding(turning off) or deferring loads. This corresponds to the case where the home is connected to a regular distribution grid where demand management is dictated by the distribution system operator.

In energy-based load control, the home operates on a limited energy budget for a given time span (e.g. 24 hours) and loads may be shed or shifted to stay within these limits. This corresponds to the case where the home is part of a microgrid that sets a priority on the use of limited local energy resources, with recourse to the external distribution grid used only as a last resort.

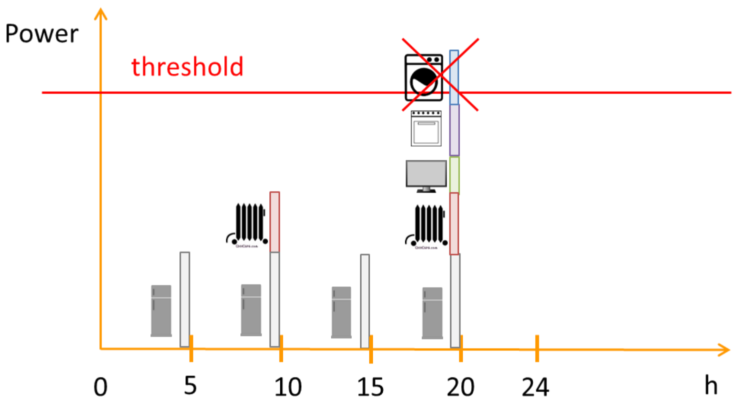


Figure A.7.1-1

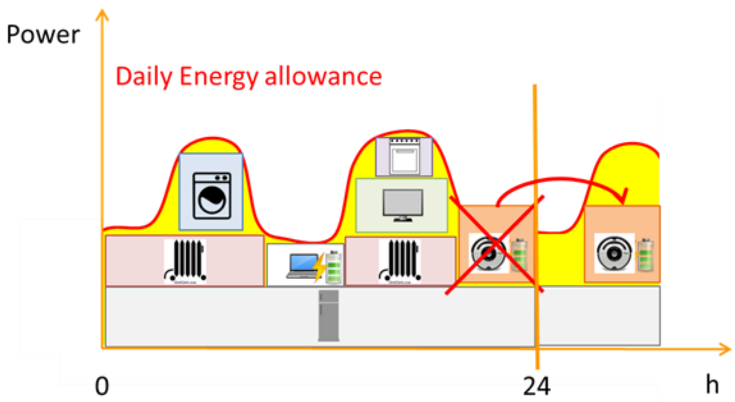


Figure A.7.1-17

A.7.2 Source

Orange Labs (ETSI).

A.7.3 Actors

- Energy distribution operator or microgrid management system.
- Home energy management system.
- Individual home appliances that can be controlled either directly or indirectly.
- Home M2M platform.
- Home users.
- Home context.

A.7.4 Pre-conditions

A.7.5 Triggers

- Change of power threshold set for home electricity consumption by energy distribution operator (for power-based variant).
- Turn on appliance that leads to exceed set aggregate power threshold.
- Change of energy budget for target time-span(for energy variant): this may be due to e.g. better than predicted solar generation from PV panel.
- Deviation from predict use that leads to significantly deviate from allocated resources.

A.7.6 Normal Flow

1) In both variants of this use case, appliances whose use can be deferred are deferred in order of decreasing priority, then appliances that can be turned off are shed in order of decreasing priority, according to the category they belong to :comfort, entertainment, non-critical assistance, security, safety& critical assistance (the latter category shall in principle never be turned off), until the constraints are met. Closed loop control can be applied iteratively to individual appliances.

2) In case the power threshold or energy budget increases, a similar iterative procedure can be applied in reverse order by turning on appliances whose use might have been previously deferred.

A.7.7 Post-conditions

- Aggregate power returned under threshold.
- Aggregate energy within budget in target time span.

A.7.8 High Level Illustration

See figures A.28 and A.29.

A.7.9 Potential requirements

- The M2M System shall provide the capability for entities of the M2M system (e.g. AEs, or CSEs) to publish semantic descriptions within the M2M system.
- The M2M System shall support parsing and interpreting semantic descriptions.
- The M2M System shall support resource discovery based on semantics.

A.8 Intelligent Indoor Air Cleaning Use Case

A.8.1 Description

This use case extends the use case of A.1 with the emphasis on semantic data analytics combining heterogeneous data sources (e.g. VOC, CO2 and PM2.5) and operation triggering based on service logics (i.e. rules) assisted by semantic annotation and ontology.

In the newly decorated house, building materials and furniture may release volatile organic compounds (VOC) such as CH_2O, which is harmful to health. Thus, people usually spend several months to ventilate the house after decoration in order to reduce the VOC level. However, the volatilization rate of VOC decreases significantly with the drop of temperature and humidity indoors. In winter, it is not efficient to reduce the VOC just by opening windows.

Therefore, an intelligent indoor air cleaning system based on home automation appliances is proposed as shown in figure A.8.1-1. The air cleaning system is composed of the following elements: IntelligentAirCleaning Application (IN-AE locates in smart phone), M2M Service Platform (IN-CSE), heating device (ASN) such as smart air-conditioner, air cleaning device (ASN) such as smart air cleaner or fresh air system, humidification device (ASN) such as smart humidifier, and air quality monitoring device (ASN).

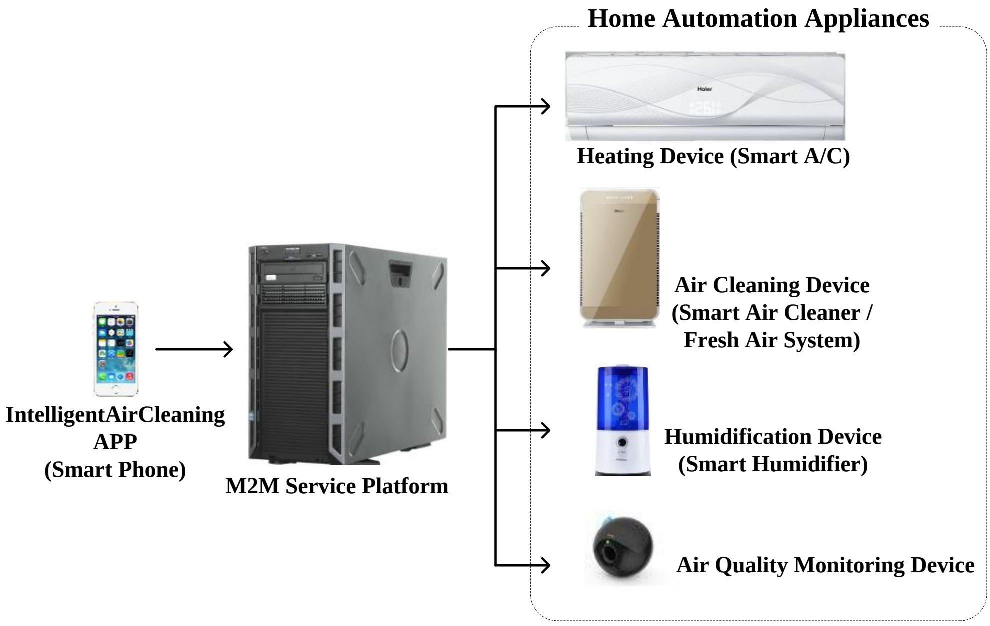


Figure A.8.1-1: Intelligent Indoor Air Cleaning System

- IntelligentAirCleaning APP (IN-AE):
  - Generates semantic query for existing home automation appliances involved in the pre-defined intelligent air cleaning system (such as air cleaning device, air quality monitoring device , heating device, humidification device), and send it to M2M Service Platform.
  - Allows users to define the detailed intelligent air cleaning service logic based on the semantic annotation and ontology of existing appliances, and the threshold of air quality index (AQI) /temperature/humidity to switch on/off the air cleaning device/heating device/humidification device.
  - Send the details of user-defined intelligent air cleaning service logic (such as appliances involved, the threshold of AQI/temperature/humidity) to M2M Service Platform.
  - Receives data (such as current AQI value) from M2M Service Platform.
- M2M Service Platform (IN-CSE with Semantics CSF):

- Discovers and selects existing appliances according to semantic annotations when receiving semantic query, and returns the information (such as URIs of semantic data and functions) of selected appliances to IntelligentAirCleaning APP.
- Generates virtual AQI Calculation Device resource and virtual Air Quality Control Device resource according to user-defined intelligent air cleaning service logic through the mash-up function of itsSemantics CSF.
- Calculates AQI with collected air quality data according to the virtual AQI Calculation Device resource description, and pushes it to IntelligentAirCleaning APP.
- Monitors the change of AQI, starts air cleaning device when AQI exceeds the user-defined threshold, meanwhile starts the heating device and the humidification device respectively if the indoor temperature and humidity exceeds the user-defined threshold, according to the virtual Air Quality Control Device resource description.

- Air Cleaning Device (ASN):
  - Offers air cleaning service according to received command.
- Air Quality Monitoring Device (ASN):
  - Monitors the temperature, humidity, VOC, CO_2 and PM 2.5/10 indoors, reports the data to the M2M service platform periodically.
- Heating Device (ASN):
  - Offers heating service according to received command, to ensure that the indoor temperature is high enough for the VOC to volatilize in relatively high speed.
- Humidification Device (ASN):
  - Offers humidification service according to received command, to ensure that the indoor humidity is high enough for the VOC to volatilize in relatively high speed.

Figure A.8.18 shows the ontology model for Intelligent Indoor Air Cleaning System.

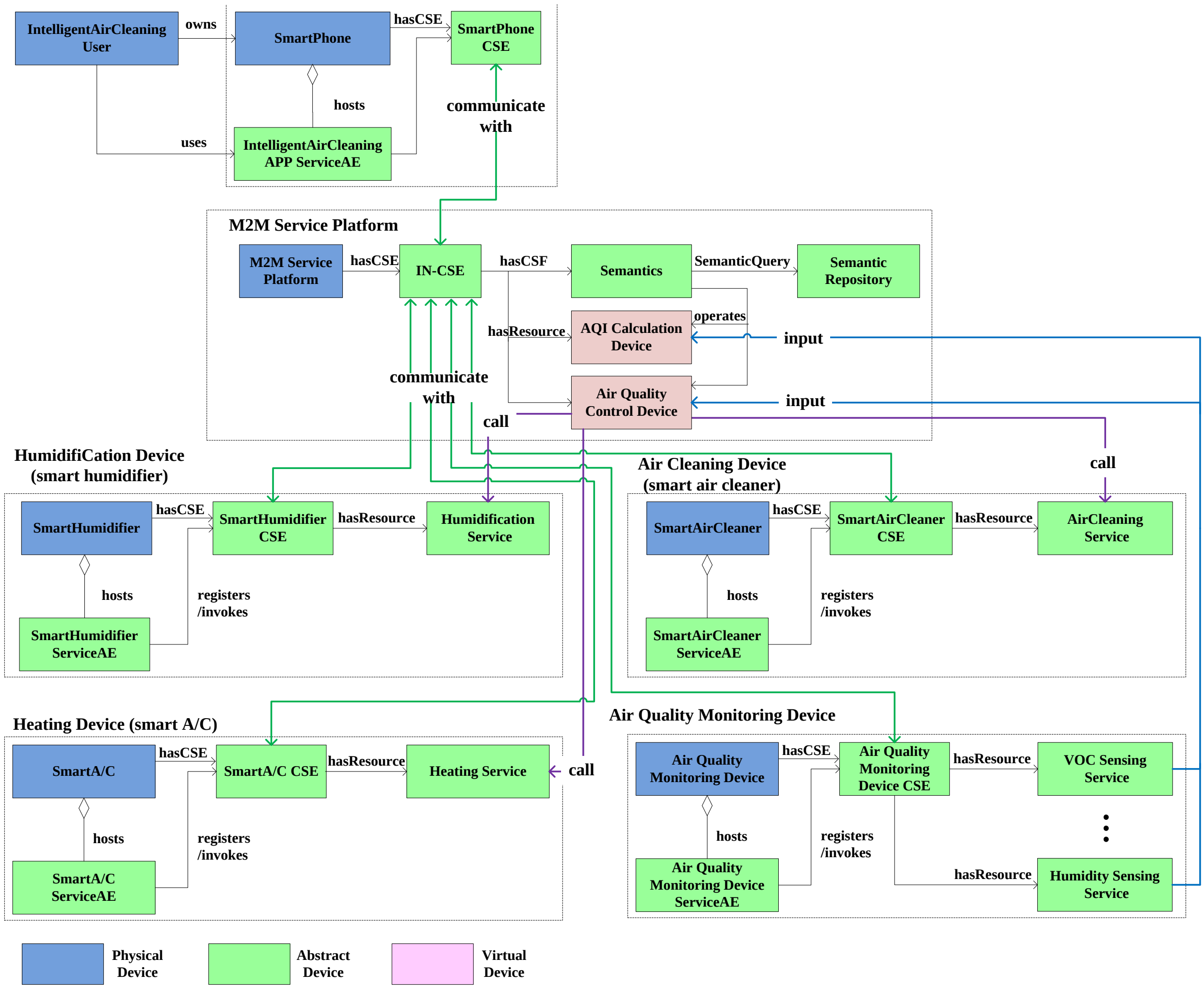


Figure A.8.18-2: Ontology Model for Intelligent Indoor Air Cleaning System

The service mapping relationships between virtual resources and home automation appliances is shown in figure A.8.1-19.

- AQI Calculation Device:
  - Takes the data of abstract device VOC/CO2/PM2.5/PM10 Sensors as inputs, calculates the AQI value with given service logic.
- Air Cleaning System:
  - AirCleaning Service: takes AQI as input, and generates device-control message to call the functions of abstract device AirCleaning when AQI exceeds the user-defined threshold.
  - Humidity Control Service: takes AQI and data of abstract device Humidity Sensor as inputs, and generates device-control message to call the function of abstract device Humidification when AQI exceeds the user-defined threshold and the humidity is below the user-defined threshold.
  - Temperature Control Service: takes AQI and data of abstract device Temperature Sensor as inputs, and generates device-control message to call the function of abstract device Heating when AQI exceeds the user-defined threshold and the temperature is below the user-defined threshold.

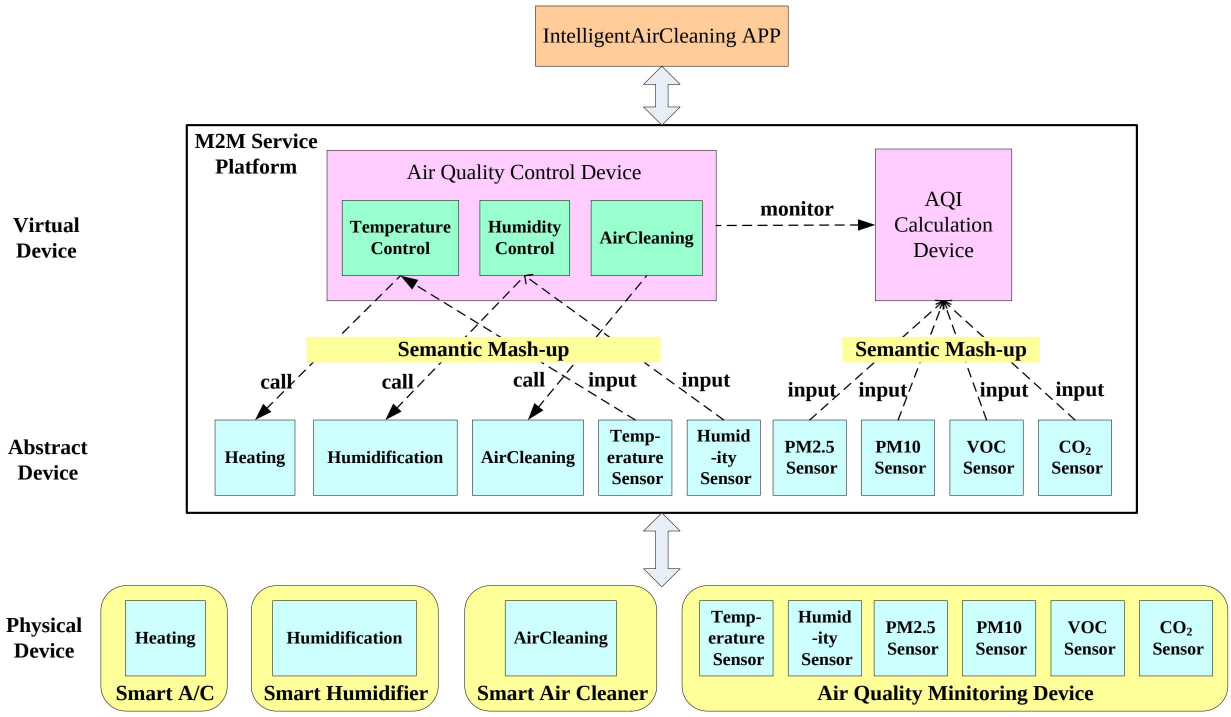


Figure A.8.1-19: Service Mapping Relationships

A.8.2 Source

- Haier (CCSA)
- Huawei (CCSA)

A.8.3 Actors

- Home Automation APP
- M2M Service Platform
- Air Cleaning Device
- Air Quality Monitoring Device
- Heating Device
- Humidification Device

A.8.4 Pre-conditions

The M2M Service Platform contains all the necessary information to manage the whole home automation appliances. It contains a Semantics CSF to support semantic functions such as mash-up and reasoning.

The IntelligentAirCleaning APP can generate semantic query to retrieve the appliance information from M2M Service Platform.

The home automation appliances can interact with the M2M Service Platform which can execute the appliance control command and report the appliance status to IntelligentAirCleaning APP.

A.8.5 Triggers

None.

A.8.6 Normal Flow

Figure A.8.6-1 displays the flow diagram of Intelligent Indoor Air Cleaning System.

- Step 0: The smart home service provider creates the ontology model and profile for the types of mentioned home automation appliances in M2M Service Platform in advance.
- Step 1: All nodes register resources to both local ASN-CSEs and ontology repository in M2M Service Platform respectively.
- Step 2: The IntelligentAirCleaning APP generates semantic queries for existing home automation appliances based on the pre-defined intelligent air cleaning system(e.g. gets air cleaning devices, gets air quality monitor devices, gets heating devices, gets humidification devices), and send it to M2M Service Platform.
- Step 3: The Semantic CSF of M2M Service Platform obtains the required appliances through semantic discovery.
- Step 4: The Semantic CSF returns the information (such as URIs of semantic data and functions) of obtained appliances to IntelligentAirCleaning APP.
- Step 5: The user defines the detailed intelligent air cleaning service logic based on the semantic annotation and ontology of existing appliances and the threshold of AQI/temperature/humidity.
- Step 6: The IntelligentAirCleaning APP sends the user-defined intelligent air cleaning service logic to M2M Service Platform.
- Step 7: The Semantic CSF of M2M Service Platform mashes up a virtual AQI Calculation Device resource and a virtual Air Quality Control Device resource according to user-defined intelligent air cleaning service logic.
- Step 8: The virtual resources of AQI Calculation Device and Air Quality Control Device offer intelligent air cleaning service for users. For example, it periodically calculates AQI and reports to IntelligentAirCleaning APP; starts air cleaning service (locates in the smart air cleaner) when AQI exceeds the user-defined threshold; starts heating service (locates in the smart air-conditioner) when the temperature is below the user-defined threshold; starts the humidification service (locates in the smart humidifier) when the humidity is below the user-defined threshold.

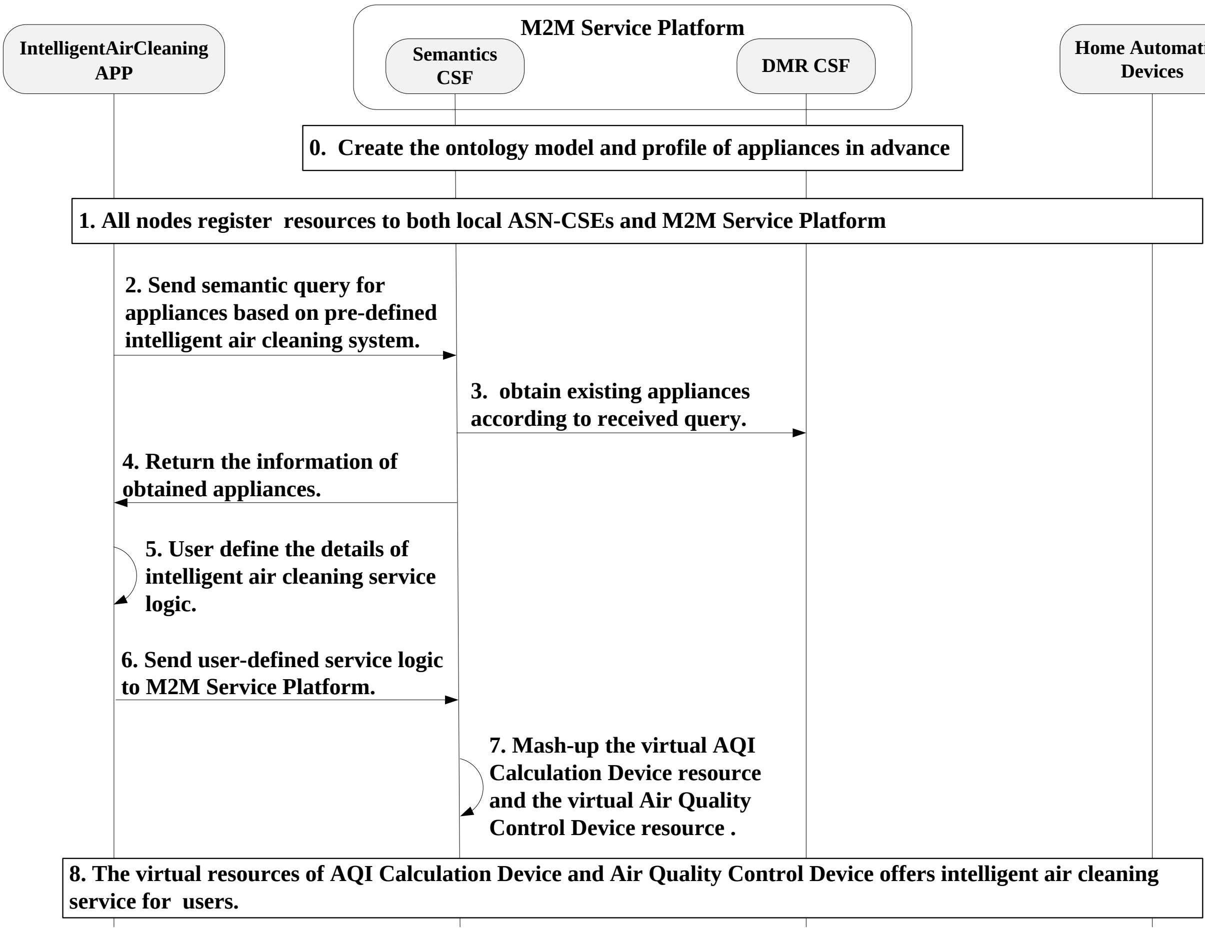


Figure A.8.6-1: High Level Flow Diagram of Intelligent Indoor Air Cleaning System

A.8.7 Post-conditions

None.

A.8.8 High Level Illustration

See figures A.8.1-1 and A.8.1-19.

A.8.9 Potential requirements

The M2M System shall provide the capability of mashing up virtual device resources based on existing device resources using semantic description and reasoning.

The M2M System shall provide the capability to embed semantic mash-up processing into M2M System to enable data analytics (e.g. calculation).

The M2M System shall provide the capability of interpreting and applying service logic (e.g. rules/policies of triggering operations upon other resources or attributes according to the change of the monitoring resource) described with semantic annotation and ontology.

A.9 Semantic home care for smart aging

A.9.1 Description

Home care is a key part of smart aging, it provides elder's health supervision service for taking care elder, tracking and monitoring rehabilitation care. In this use case, as shown in figure A.9.1-1 one elder wears/uses one or more professional monitoring devices which collect the daily/real-time data that reflect the elder's health state, such as blood pressure, heartbeat, body temperature, pulse, blood sugar, position status etc. These data is sent to M2M platform via home gateway or cell phone. The home care applications/services provide the smart home care services based on semantic processing of the collection data.

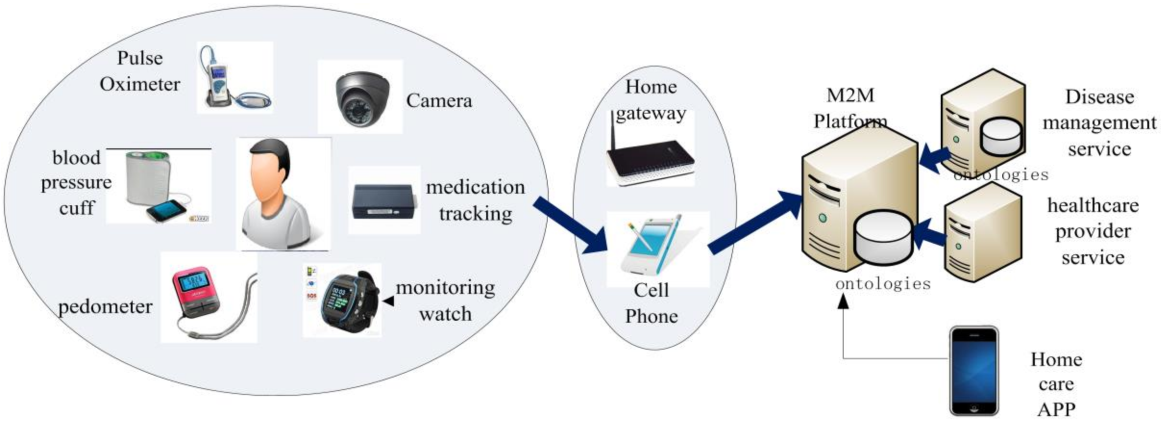


Figure A.9.1-1

In this use case, M2M system defines ontologies for describing/associating the variant home care resources (e.g. entity, data) in the M2M platform, and annotates these resources with semantic information based on the defined ontologies.

For the different service levels, if M2M applications/services have strong capabilities, they can also define professional domain/application ontologies describe home care by themselves, and merge with the M2M system's ontologies. M2M applications/services can also use self-defined ontologies or M2M system's ontologies individually.

M2M system, M2M applications/services can automatically infer the collection data by semantic processing (include semantic annotation, reasoning, storage etc.), and then M2M applications/services provide elders customized home care services, e.g. comprehensive health evaluating for intervening/treating disease in early stage, emergency rescue, etc.

A.9.2 Source

- ZTE

A.9.3 Actors

- Home care applications

M2M platform:

- Home care monitoring device, provided by service provider or application provider
- The third party system, provide customized/personal home care service, such as Disease management system.
- Home care users

A.9.4 Pre-conditions

- The monitoring devices register into the M2M platform that associates with the home care user.

- Home care applications/the third party systems have access rights for obtaining ontologies information.
- Home care applications/the third party systems can define ontologies and has the capability of merging with the ontologies defined in M2M system.

A.9.5 Triggers

Home care application/the third party system establish elders' files and want to collect elders' data for providing home care service.

A.9.6 Normal Flow

1) If home care application or the third party system self-defines ontologies, home care application/the third party system requests to query home care related ontologies information from the M2M platform, then merges the self-defined ontologies with M2M system responded ontologies information. Home care application/the third party system could send the merged ontologies to M2M platform to be stored.
2) Home care application/the third party system subscribes/queries the monitoring device data with semantic information. When the M2M platform receives the semantic data from monitoring device, it stores the semantic data and sends them to home care application/the third party system.
3) Home care application/the third party system semantically processes (e.g. analyze/infer)the semantic data from the M2M platform with merged ontologies, and comes to a conclusion, such as keeping monitoring, calling the elder's family.

A.9.7 Post-conditions

None.

A.9.8 High Level Illustration

See figure A.9.1-1.

A.9.9 Potential requirements

The M2M System shall support storing ontology information which can be discovered and retrieved by M2M application or the third party system..

A.10 Air environment monitoring service

A.10.1 Description

Nowadays air quality is concerned by more and more people. Among various air quality related parameters, overall air pollution indexes are more attractive for most people compared with direct air pollution parameters (e.g. NO2, SO2, PM2.5). However, the overall air pollution index is generally calculated based on multiple air pollution parameters and cannot be directly measured by single device.

To facilitate the access of overall air pollution index, the use case described in clause A.1 provides a feasible method for direct access to the overall air pollution index via creating the virtual resources by the mash-up of multiple resources related to direct air pollution parameters.

This use case extends the use case described in clause A.1 with the emphasis on the sharing of the mash-up resources. As shown in figure A.10.1-1, in this use case, there are two M2M platforms, M2M platform A and M2M platform B. M2M platform A is in charge of the city air environment monitoring devices, which may be a government infrastructure. M2M platform B provides the value-added air environment monitoring services to customers. But M2M platform B may be only in charge of some private air environment monitoring devices or even has none air environment monitoring devices. The air quality monitoring APP in use's mobile phone requests M2M platform B for the overall air quality index. Then M2M platform B will forward the request to M2M platform A and get the returned results. Some additional processing (e.g. introducing the effect of the data from private air environment monitoring devices or using private algorithm to re-calculate the air quality index) will be made by M2M platform B before forwarding the results to user's APP for providing value-added services.

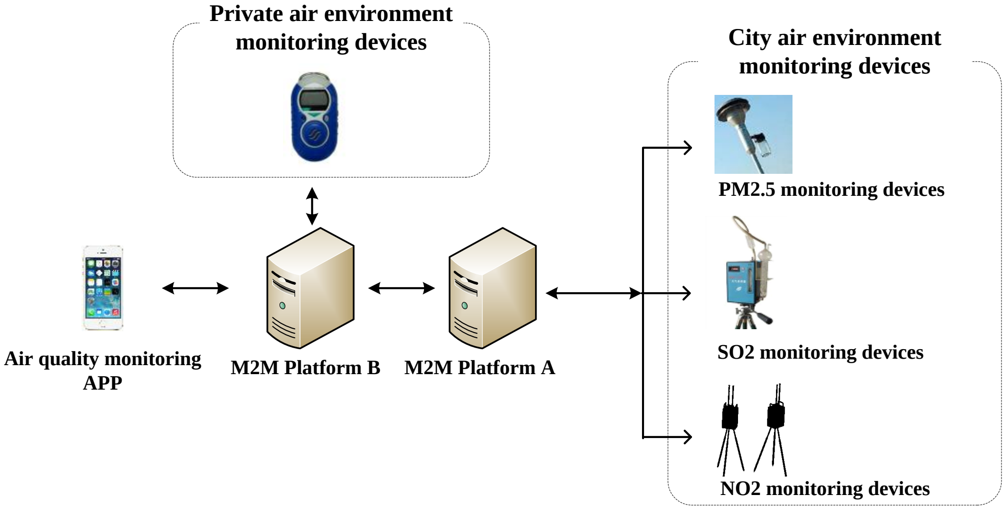


Figure A.10.1-1: An example of air environment monitoring systems

A.10.2 Source

- China Unicom

A.10.3 Actors

- Air quality monitoring APP
- M2M platform A (hosted by government)
- M2M platform B
- City air environment monitoring devices (hosted by government)
- Private air environment monitoring devices

A.10.4 Pre-conditions

- City air environment monitoring devices are registered in M2M platform A.

A.10.5 Triggers

None

A.10.6 Normal Flow

Figure 10.10.20-1 displays the flow diagram of the considered air environment monitoring system.

- Step 0: the air quality monitoring APP and private air environment monitoring devices are registered in M2M platform B. City air environment monitoring devices are registered in M2M platform A.

- Step 1: the air quality monitoring APP sends a request to M2M platform B for querying air quality index.
- Step 2: M2M platform B checks that there are no matched resources in the CSE, and forward the request to M2M platform A.
- Step 3: M2M platform A discovers a corresponding virtual resource created by a mash-up of multiple related air environment monitoring devices.
- Step 4: M2M platform A returns the identifier of the corresponding virtual resource to M2M platform B.
- Step 5: M2M platform B get the information of mash-up logic and the identifiers of the resources involved in the mash-up through the returned identifier.
- Step 6: M2M platform B re-creates a virtual air quality resource by the mash-up of air quality related resources (including the related resources hosted by M2M platform B and the related resources hosted by M2M platform A) using private mash-up logic.
- Step 7: M2M platform B returns the identifier of the created virtual air quality resource.

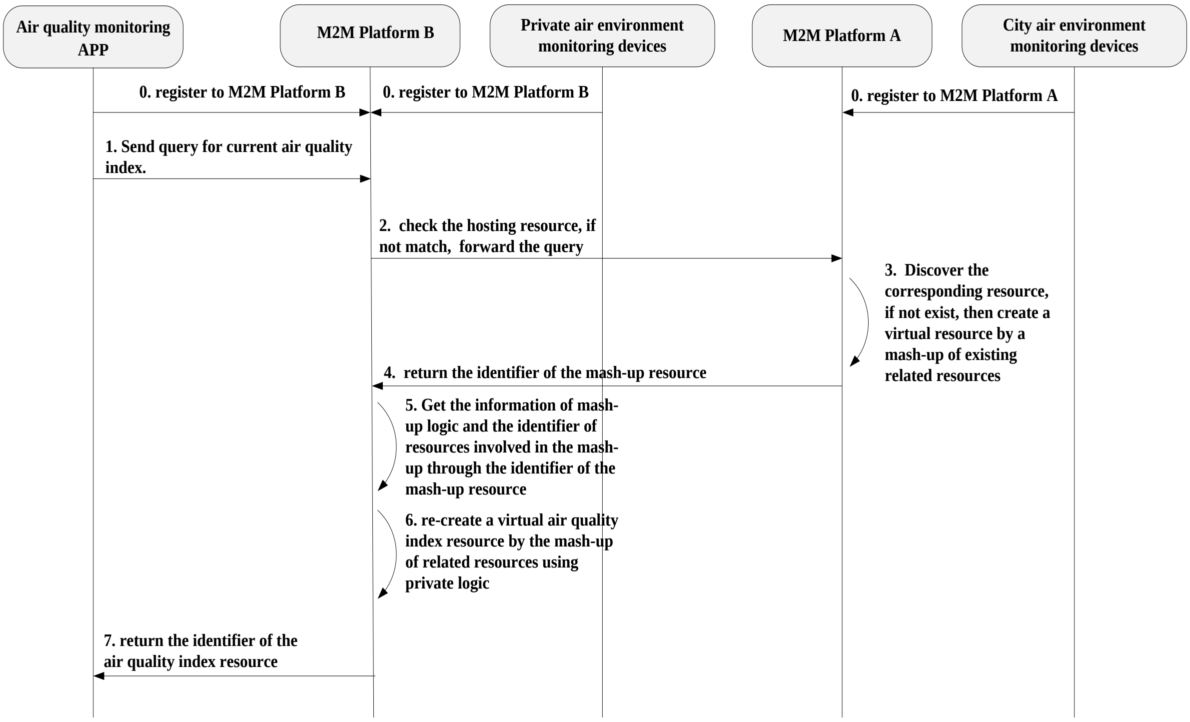


Figure 10.20-1: High level flow diagram of the considered air environment monitoring system

A.10.7 Alternative flow

None.

A.10.8 Post-conditions

None.

A.10.9 High Level Illustration

See figure A.10.1-1.

A.10.10 Potential requirements

1) The oneM2M system should support the capability for sharing the mash-up resources within a single M2M service provider domain.

2) The oneM2M system should support the capability for sharing the mash-up resources across different M2M service provider domains.

3) The oneM2M system should support the exposure of the mash-up service logic of mash-up resources.

4) The oneM2M system should support the exposure of the identifiers or contents of the resources involved in the mash-up.

History

Publication history		
V2.11.1	30-Aug-2016	Publication