TTC仕様書
Technical Specification

# TS-M2M-0034v4.2.0

# セマンティクスのサポート

## Semantics Support

2023 年 3 月 17 日制定

一般社団法人
情報通信技術委員会

THE TELECOMMUNICATION TECHNOLOGY COMMITTEE

TS-M2M-0034v4.2.0

セマンティクスのサポート　[Semantics Support]

＜参考＞ [Remarks]

１．英文記述の適用レベル [Application level of English description]

適用レベル [Application level]：E2

本標準の本文、付属資料および付録の文章および図に英文記述を含んでいる。

[English description is included in the text and figures of main body, annexes and appendices.]

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

本標準は、oneM2M で承認された Technical Specification TS-0034-V4.2.0 に準拠している。

[This standard is standardized based on the Technical Specification TS-0034-V4.2.0 approved by oneM2M.]

３．上記国際勧告等に対する追加項目等 [Departures from international recommendations]

原標準に対する変更項目 [Changes to original standard]

原標準が参照する標準のうち、TTC 標準に置き換える項目。[Standards referred to in the original standard, which are replaced by TTC standards.]

原標準が参照する標準のうち、それらに準拠した TTC 標準等が制定されている場合は自動的に最新版 TTC 標準等に置き換え参照するものとする。 [Standards referred to in the original standard should be replaced by derived TTC standards.]

４．工業所有権 [IPR]

本標準に関わる「工業所有権等の実施の権利に係る確認書」の提出状況は、ＴＴＣホームページによる。

[Status of “Confirmation of IPR Licensing Condition” submitted is provided in the TTC web site.]

５．作成専門委員会 [Working Group]

oneM2M 専門委員会 [oneM2M Working Group]

-

# ONEM2M
# TECHNICAL SPECIFICATION

| | |
|---|---|
| Document Number | TS-0034-V4.2.0 |
| Document Name: | Semantics Support |
| Date: | 2020-01-08 |
| Abstract: | This specification provides normative text for semantic enablement in oneM2M |

Template Version: January 2017 (Do not modify)

-

About oneM2M

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

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

-

# Contents

-

-

# 1 Scope

The present document specifies several semantic functions for oneM2M functional architecture [1] including basic resource procedures and functional descriptions.

# 2 References

## 2.1 Normative references

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

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

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

[2] W3C Recommendation: "SPARQL 1.1 Query Language".

[3] oneM2M TS-0004: "Service Layer Core Protocol Specification".

[4] W3C Recommendation 25 February 2014: "RDF 1.1 Concepts and Abstract Syntax".

[5] oneM2M TS-0012: "Base Ontology"

## 2.2 Informative references

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

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

[i.1] W3C Editor's Draft: "Semantic Sensor Network Ontology".

NOTE: Available at http://w3c.github.io/sdw/ssn/.

[i.2] ETSI TS 103 264 (V1.1.1): "SmartM2M; Smart Appliances; Reference Ontology and oneM2M Mapping".

[i.3] oneM2M TR-0033: "Study on Enhanced Semantic Enablement".

[i.4] oneM2M Drafting Rules.

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

# 3 Abbreviations

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

| | |
|---|---|
| ACP | Access Control Policy |
| ACR | Access Control Rule |
| AE | Application Entity |
| CRUD | Create, Retrieve, Update, Delete |
| CSE | Common Service Entity |
| CSF | Common Service Function |

-

| | |
|---|---|
| IoT | Internet of Things |
| IRI | Internationalized Resource Identifier |
| JSON | JavaScript Object Notation |
| MR | Mashup Requestor |
| RDF | Resource Description Framework |
| RH | Resource Host |
| SAREF | Smart Appliances REFerence ontology |
| SD | Semantic Descriptor |
| SEM | Semantics |
| SGS | Semantic Graph Store |
| SMF | Semantic Mashup Function |
| SMI | Semantic Mashup Instance |
| SMJP | Semantic Mashup Job Profile |
| SPARQL | SPARQL Protocol and RDF Query Language |
| SSN | Semantic Sensor Network |
| URI | Uniform Resource Identifier |
| URL | Uniform Resource Locator |
| XML | Extensible Markup Language |

# 4 Conventions

The key words "Shall", "Shall not", "May", "Need not", "Should", "Should not" in this document are to be interpreted as described in the oneM2M Drafting Rules [i.4].

# 5 Architectural Model and Concepts

The architectural model assumed in this specification is based on the generic oneM2M architecture for the Common Service Layer specified in oneM2M TS-0001 [1]. The core functionality supporting semantics resides at various CSEs, providing services to the AEs via the Mca reference point and interacting with other CSEs via the Mcc reference point.

The Semantics (SEM) CSF (see clause 6.2.14 in oneM2M TS-0001 [1]) is an oneM2M Common Service Function (CSF) which enables semantic information management and provides the related functionality based on this semantic information. The functionality of this CSF is based on semantic descriptions and implemented through the specialized resources and procedures described in this specification. This functionality is also enabled by other, more generic, resources and procedures described in oneM2M TS-0001 [1] and further referenced in this specification. The main features of the SEM CSF are listed in clause 10.2.14 of [1] and further detailed in clauses 6 and 7 of this specification. The SEM CSF includes specialized functional blocks such as: SPARQL engine, repositories for ontologies and semantic descriptions, which may be implemented via permanent or temporary Semantic Graph Stores, etc.

# 6 Basic Resource Procedures

## 6.1 <*semanticDescriptor*> Operations

### 6.1.1 Introduction

The <semanticDescriptor> resource is used to store a semantic description pertaining to a resource and potentially sub-resources. Such a description may 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. For resource type description see [1] clause 9.6.30.

### 6.1.2 Create <*semanticDescriptor*>

This procedure shall be used for creating a <*semanticDescriptor* > resource.

-

**Table 6.1.2-1: <*semanticDescriptor*> CREATE**

| **<*semanticDescriptor*> CREATE** | |
|---|---|
| Associated Reference Point | Mca, Mcc and Mcc' |
| Information in Request message | All parameters defined in oneM2M TS-0001 [1] table 8.1.2-2 apply with the specific details for:<br>***Content:*** The resource content shall provide the information as defined in clause 9.6.30 in oneM2M TS-0001 [1] |
| Processing at Originator before sending Request | According to clause 10.1.2 in oneM2M TS-0001 [1] |
| Processing at Receiver | The Hosting CSE shall follow the basic procedure according to clause 10.1.2 of [1], with the following specific details:<br>• shall check that the *descriptor* attribute conforms to the syntax as defined in the *descriptorRepresentation* attribute.<br>• shall trigger the semantic validation process as specified in clause 7.10 if the *validationEnable* attribute of the <*semanticDescriptor*> resource is set to true, and shall set the *semanticValidated* attribute of <*semanticDescriptor*> resource according to the validation result. |
| Information in Response message | According to clause 10.1.2 in oneM2M TS-0001 [1] |
| Processing at Originator after receiving Response | According to clause 10.1.2 in oneM2M TS-0001 [1] |
| Exceptions | According to clause 10.1.2 in oneM2M TS-0001 [1] |

## 6.1.3 Retrieve <*semanticDescriptor*>

This procedure shall be used for retrieving the attributes of a <*semanticDescriptor*> resource.

**Table 6.1.3-1: <*semanticDescriptor*> RETRIEVE**

| **<*semanticDescriptor* > RETRIEVE** | |
|---|---|
| Associated Reference Point | Mca, Mcc and Mcc'. |
| Information in Request message | All parameters defined in oneM2M TS-0001 [1] table 8.1.2-2. |
| Processing at Originator before sending Request | According to clause 10.1.3. in oneM2M TS-0001 [1] |
| Processing at Receiver | According to clause 10.1.3 in oneM2M TS-0001 [1]. |
| Information in Response message | All parameters defined in oneM2M TS-0001 [1] table 8.1.3-1 apply. |
| Processing at Originator after receiving Response | According to clause 10.1.3 in oneM2M TS-0001 [1]. |
| Exceptions | According to clause 10.1.3 in oneM2M TS-0001 [1]. |

-

## 6.1.4 Update <*semanticDescriptor*>

This procedure shall be used for updating attributes of a <*semanticDescriptor* > resource.

**Table 6.1.4-1: <*semanticDescriptor*> UPDATE**

| **<*semanticDescriptor*> UPDATE** | |
|---|---|
| Associated Reference Point | Mca, Mcc and Mcc' |
| Information in Request message | All parameters defined in oneM2M TS-0001 [1] table 8.1.2-2 apply with the specific details for:<br>the ***Content*** request parameter which may contain the new descriptor information in one of the following ways:<br>1) full representation of the *descriptor* attribute; or<br>2) partial representation of the *descriptor* attribute as described in SPARQL statements [2] in the *semanticOpExec* attribute. |
| Processing at Originator before sending Request | According to clause 10.1.4 in oneM2M TS-0001 [1]. |
| Processing at Receiver | The hosting CSE shall follow the basic procedure according to clause 10.1.4 in oneM2M TS-0001 [1], with the following specific details:<br>• check if both *semanticOpExec* attribute and *ontologyContent* attribute exist in the the ***Content*** request parameter, if so, return an error code;<br>• shall update the *descriptor* attribute according to the execution result of the SPARQL statements [2] in the *semanticOpExec* attribute, if it presents in the ***Content*** request parameter;<br>• shall check that the *descriptor* attribute conforms to the syntax as defined in the *descriptorRepresentation* attribute, if it presents in the ***Content*** request parameter.<br>• shall trigger the semantic validation process as specified in clause 7.10 if the *validationEnable* attribute of the <*semanticDescriptor*> resource is set to true, and shall update the *semanticValidated* attribute of <*semanticDescriptor*> resource according to the validation result. |
| Information in Response message | According to clause 10.1.4 in oneM2M TS-0001 [1]. |
| Processing at Originator after receiving Response | According to clause 10.1.4 in oneM2M TS-0001 [1]. |
| Exceptions | According to clause 10.1.4 in oneM2M TS-0001 [1]. |

## 6.1.5 Delete <*semanticDescriptor*>

This procedure shall be used for deleting a <*semanticDescriptor*> resource.

**Table 6.1.5-1: <*semanticDescriptor*> DELETE**

| **<*semanticDescriptor*> DELETE** | |
|---|---|
| Associated Reference Point | Mca, Mcc and Mcc' |
| Information in Request message | All parameters defined in table 8.1.2-2 in oneM2M TS-0001 [1] apply |
| Processing at Originator before sending Request | According to clause 10.1.5 in oneM2M TS-0001 [1] |
| Processing at Receiver | According to clause 10.1.5 in oneM2M TS-0001 [1] |
| Information in Response message | According to clause 10.1.5 in oneM2M TS-0001 [1] |
| Processing at Originator after receiving Response | According to clause 10.1.5 in oneM2M TS-0001 [1] |
| Exceptions | According to clause 10.1.5 in oneM2M TS-0001 [1] |

-

## 6.2 <semanticFanOutPoint> Operations

### 6.2.1 Introduction

The <*semanticFanOutPoint*> resource is a virtual resource because it does not have a representation. It is the child resource of a <*group*> resource and shall be targeted only by RETRIEVE requests. When a request (for semantic discovery or semantic query) is sent to the <*semanticFanOutPoint*> resource the host uses the *memberIDs* attribute of the parent <*group*> resource to retrieve all the related descriptors, then proceeds with the corresponding processing.

See clause 9.6.14a in oneM2M TS-0001 [1] for a full description of the resource type. The use of <*semanticFanOutPoint*> for semantic resource discovery and semantic query is further described in clause 7.4.

### 6.2.2 Retrieve <*semanticFanOutPoint*>

The RETRIVE operation on <*semanticFanOutPoint*> shall be used for two purposes:

1) performing semantic resource discovery; and

2) performing semantic query.

The procedure below shall be used for performing a semantic discovery or a semantic query procedure using the descriptor content of all member semantic resources belonging to an existing <*group*> resource.

**Table 6.2.2-1: <*semanticFanOutPoint*> RETRIEVE for Semantic Resource Discovery and Semantic Query**

| **<*semanticFanOutPoint*> RETRIEVE** | |
|---|---|
| Associated Reference Point | Mca, Mcc and Mcc' |
| Information in Request message | According to clause 10.1.3 in oneM2M TS-0001 [1].<br><br>For the semantic query case, the request message shall include the parameter *Semantic Query Indicator*, which shall not be included in the request for semantic resource discovery. |
| Processing at Originator before sending Request | For the semantic resource discovery case, the Originator shall request a semantic discovery to be performed using the content of the semantic descriptors of all member resources belonging to an existing <*group*> resource.<br><br>For the semantic query case, the Originator may discover various <*group*> resources defining different explicit query scopes and select the one having the desired query scope. Then, the Originator shall request a semantic query to be performed using the semantic information of all member resources belonging to this <*group*> resource.<br><br>The Originator may be an AE or CSE. |
| Processing at Receiver | The Receiver shall:<br>• Check if the Originator has RETRIEVE privilege in the <*accessControlPolicy*> resource referenced by the *membersAccessControlPolicyIDs* in the parent <*group*> resource. In the case *membersAccessControlPolicyIDs* is not provided, the access control policy defined for the parent <*group*> resource shall be used.<br>• Upon successful validation, obtain the URIs of all the member semantic resources from the *memberIDs* attribute of the parent<*group*> resource.<br>• If there are semantic resources stored on different CSEs, individual RETRIEVE requests are sent to each CSE for retrieving the descriptors, otherwise the *descriptor* attributes are simply retrieved for all the semantic resources hosted locally. All semantic descriptors are accessed based on the respective access control policies.<br>• Once all of the related *descriptor* attributes have been retrieved, the SPARQL request is being executed on the combined content. |
| Information in Response message | The result of the SPARQL request executed on the content retrieved from the semantic resources. |
| Processing at Originator after receiving Response | According to clause 10.1.3 in oneM2M TS-0001 [1]. |
| Exceptions | According to clause 10.1.3 in oneM2M TS-0001 [1]. |

-

## 6.3 <*semanticMashupJobProfile*> Operations

### 6.3.1 Introduction

The <*semanticMashupJobProfile*> resource represents a Semantic Mashup Job Profile (SMJP). The <*semanticMashupJobProfile*> resource type description is specified in the clause 9.6.53 in oneM2M TS-0001 [1].

A <*semanticMashupJobProfile*> resource can be provisioned to a Hosting CSE which provides semantic mashup function; alternatively, an AE or CSE can request to create <*semanticMashupJobProfile*> resource at the Hosting CSE. Once a <*semanticMashupJobProfile*> resource is provisioned or created at the Hosting CSE, other oneM2M CSEs/AEs, which act as Mashup Requestors, can discover, retrieve, update, or delete it based on the requirements.

Figure 6.3.1-1 illustrates a generic procedure (e.g. Create/Retrieve/Update/Delete) to operate on a <*semanticMashupJobProfile*> resource.

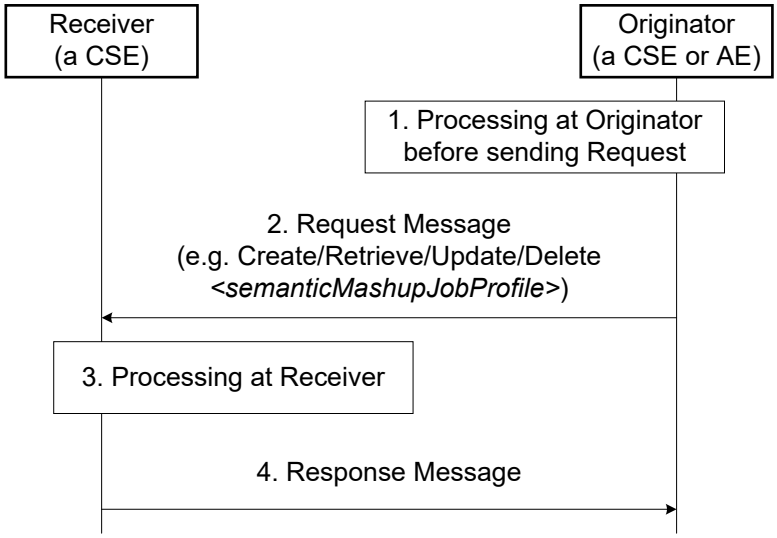


**Figure 6.3.1-1: Procedures for operating a <*semanticMashupJobProfile*> resource**

### 6.3.2 Create <semanticMashupJobProfile>

This procedure shall be used for creating a <*semanticMashupJobProfile*> resource as described in Table 6.3.2-1.

**Table 6.3.2-1: <*semanticMashupJobProfile*> CREATE**

| **<*semanticMashupJobProfile*> CREATE** | |
|---|---|
| Associated Reference Point | Mca, Mcc and Mcc'. |
| Information in Request message | All parameters defined in Table 8.1.2-3 in oneM2M TS-0001 [1] apply with the specific details for:<br>**Content**: The resource content shall provide the information about an <*semanticMashupJobProfile*> resource (e.g. attribute values) as described in the clause 9.6.53 in oneM2M TS-0001 [1]. |
| Processing at Originator before sending Request | According to clause 10.1.1.1 in oneM2M TS-0001 [1]. |
| Processing at Receiver | According to clause 10.1.1.1 in oneM2M TS-0001 [1]. |
| Information in Response message | All parameters defined in Table 8.1.3-1 in oneM2M TS-0001 [1] apply with the specific details for:<br>**Content:** Address of the created <*semanticMashupJobProfile*> resource, according to clause 10.1.1.1 in oneM2M TS-0001 [1]. |
| Processing at Originator after receiving Response | According to clause 10.1.1.1 in oneM2M TS-0001 [1]. |
| Exceptions | According to clause 10.1.1.1 in oneM2M TS-0001 [1]. |

-

## 6.3.3 Retrieve <semanticMashupJobProfile>

This procedure shall be used for retrieving the attributes of a <*semanticMashupJobProfile*> resource as described in Table 6.3.3-1.

**Table 6.3.3-1: <*semanticMashupJobProfile*> RETRIEVE**

| **<*semanticMashupJobProfile*> RETRIEVE** | |
|---|---|
| Associated Reference Point | Mca, Mcc and Mcc'. |
| Information in Request message | All parameters defined in Table 8.1.2-3 in oneM2M TS-0001 [1] apply with the specific details for:<br>**Content:** void. |
| Processing at Originator before sending Request | According to clause 10.1.2 in oneM2M TS-0001 [1]. |
| Processing at Receiver | The Receiver shall verify the existence (including **Filter Criteria** checking, if it is given) of the target resource or the attribute and check if the Originator has appropriate privileges to retrieve information stored in the resource/attribute. Otherwise clause 10.1.2 in oneM2M TS-0001 [1] applies. |
| Information in Response message | All parameters defined in Table 8.1.3-1 in oneM2M TS-0001 [1] apply with the specific details for:<br>**Content:** attributes of the <*semanticMashupJobProfile*> resource as defined in the clause 9.6.53 in oneM2M TS-0001 [1]. |
| Processing at Originator after receiving Response | According to clause 10.1.2 in oneM2M TS-0001 [1]. |
| Exceptions | According to clause 10.1.2 in oneM2M TS-0001 [1].<br>In addition, a timer has expired. The Receiver responds with an error. |

## 6.3.4 Update <semanticMashupJobProfile>

This procedure as described in Table 6.3.4-1 shall be used to update an existing <*semanticMashupJobProfile*> resource, e.g. an update to its *inputDescriptor* attribute.

**Table 6.3.4-1: <*semanticMashupJobProfile*> UPDATE**

| **<*semanticMashupJobProfile*> UPDATE** | |
|---|---|
| Associated Reference Point | Mca, Mcc and Mcc'. |
| Information in Request message | All parameters defined in Table 8.1.2-3 in oneM2M TS-0001 [1] apply with the specific details for:<br>**Content:** attributes of the <*semanticMashupJobProfile*> resource as defined in the clause 9.6.53 in oneM2M TS-0001 [1] to be updated. |
| Processing at Originator before sending Request | According to clause 10.1.3 in oneM2M TS-0001 [1]. |
| Processing at Receiver | According to clause 10.1.3 in oneM2M TS-0001 [1]. |
| Information in Response message | According to clause 10.1.3 in oneM2M TS-0001 [1]. |
| Processing at Originator after receiving Response | According to clause 10.1.3 in oneM2M TS-0001 [1]. |
| Exceptions | According to clause 10.1.3 in oneM2M TS-0001 [1]. |

-

### 6.3.5 Delete <semanticMashupJobProfile>

This procedure as described in Table 6.3.5-1 shall be used to delete an existing <*semanticMashupJobProfile*> resource.

**Table 6.3.5-1: <*semanticMashupJobProfile*> DELETE**

| **<*semanticMashupJobProfile*> DELETE** | |
|---|---|
| Associated Reference Point | Mca, Mcc and Mcc'. |
| Information in Request message | All parameters defined in Table 8.1.2-3 in oneM2M TS-0001 [1] apply. |
| Processing at Originator before sending Request | According to clause 10.1.4.1 in oneM2M TS-0001 [1]. |
| Processing at Receiver | According to clause 10.1.4.1 in oneM2M TS-0001 [1]:<br>• If the <*semanticMashupJobProfile*> to be deleted has *smilD* attribute and the *smilD* attribute has a value, the Receiver notifies each <*semanticMashupInstance*> resource as included in the *smilD* attribute of the removal of the <*semanticMashupJobProfile*> since those <*semanticMashupInstance*> resources use this <*semanticMashupJobProfile*>.<br>• If the <*semanticMashupJobProfile*> to be deleted has <*semanticMashupInstance*> child resources, all those <*semanticMashupInstance*> child resources shall be removed accordingly. |
| Information in Response message | According to clause 10.1.4.1 in oneM2M TS-0001 [1]. |
| Processing at Originator after receiving Response | According to clause 10.1.4.1 in oneM2M TS-0001 [1]. |
| Exceptions | According to clause 10.1.4.1 in oneM2M TS-0001 [1]. |

## 6.4 <*semanticMashupInstance*> Operations

### 6.4.1 Introduction

<*semanticMashupInstance*> models and represents a Semantic Mashup Instance (SMI) resource. <*semanticMashupInstance*> resource type is specified in the clause 9.6.54 in oneM2M TS-0001 [1].

A CSE/AE as a Mashup Requestor can request to create <*semanticMashupInstance*> resources at another oneM2M CSE which implements the semantic mashup function. Each created <*semanticMashupInstance*> resource corresponds to a semantic mashup job profile (i.e. a <*semanticMashupJobProfile*> resource); in other words, how the <*semanticMashupInstance*> resource should execute the mashup operation to calculate the mashup result is specified in the corresponding <*semanticMashupJobProfile*> resource. Note that the <*semanticMashupInstance*> and its corresponding <*semanticMashupJobProfile*> resources may be placed at the same CSE or at different CSEs, and the *smjpID* attribute of the <*semanticMashupInstance*> allows locating the corresponding <*semanticMashupJobProfile*> resource. If the <*semanticMashupInstance*> resource has a <*semanticMashupResult*> as its child resource, the Mashup Requestor may use it to retrieve the mashup result.

Figure 6.4.1-1 illustrates the procedural flow to operate a <*semanticMashupInstance*> resource (e.g. Create/Retrieve/Update/Delete a <*semanticMashupInstance*> resource).

-

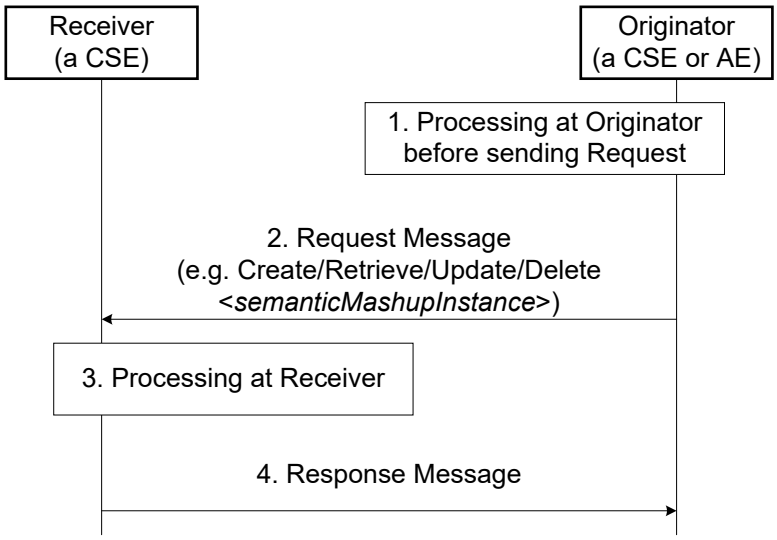


**Figure 6.4.1-1: Procedures for Operating a <*semanticMashupInstance*> Resource**

## 6.4.2 Create <semanticMashupInstance>

This procedure shall be used for creating a <*semanticMashupInstance*> resource as described in Table 6.4.2-1.

**Table 6.4.2-1: <*semanticMashupInstance*> CREATE**

| **<*semanticMashupInstance*> CREATE** | |
|---|---|
| Associated Reference Point | Mca, Mcc and Mcc'. |
| Information in Request message | All parameters defined in Table 8.1.2-3 in oneM2M TS-0001 [1] apply with the specific details for:<br>**Content**: The resource content shall provide the information about a <*semanticMashupInstance*> resource (e.g. attribute values) as described in the clause 9.6.54 in oneM2M TS-0001 [1]. |
| Processing at Originator before sending Request | According to clause 10.1.1.1 in oneM2M TS-0001 [1]:<br>• If the Originator knows the identifier or URI of each mashup member, it can include the value of *mashupMember* in the Request message. |
| Processing at Receiver | According to clause 10.1.1.1 in oneM2M TS-0001 [1]:<br>• The Receiver shall first check if the corresponding <*semanticMashupJobProfile*> as denoted by *smjpID* attribute exists or not. If it does not exist, the Receiver shall not create the <*semanticMashupInstance*> and shall report an error (e.g. "<*semanticMashupJobProfile*> does not exist") in the Response message to the Originator. If it exists, the Receiver shall retrieve its content.<br>• The Receiver shall check if *smjpInputParameter* included in the Request message meets the input parameter requirement as specified by the *inputDescriptor* attribute of corresponding <*semanticMashupJobProfile*>. If it does not meet the requirement, the Receiver shall not create the <*semanticMashupInstance*> and shall report an error (e.g. "*smjpInputParameter*" does not meet the requirement") in the Response message to the Originator.<br>• According to the *memberFilter* attribute of the retrieved <*semanticMashupJobProfile*>, the Receiver extracts the SPARQL query contained in *memberFilter* and use it to discover and determine mashup member resources for the <*semanticMashupInstance*> to be created.<br>• Dependent on the *memberStoreType* attribute contained in the Request message, the Receiver maintains each member resource in different ways. If *memberStoreType*="URI Only", the Receiver creates the *mashupMember* attribute containing the URIs of the determined member resources. If *memberStoreType*="URI and Value", the Receiver creates the *mashupMember* attribute, retrieves the content value of each member resource and then stores both the identifier and the content value of each member resource in the *mashupMember* attribute.<br>• Depending on the *resultGenType* attribute contained in the Request message, |

-

| <*semanticMashupInstance*> CREATE | |
|---|---|
| | the Receiver prepares to execute the corresponding semantic mashup job profile as follows:<br>– If *resultGenType*=" When SMI Is Created", the Receiver retrieves the content value of each member resource if not retrieved yet; then it executes mashup functions as specified by the <*semanticMashupJobProfile*> and generates semantic mashup result, which shall be stored in the <*semanticMashupResult*> child resource.<br>– If *resultGenType*="When A Mashup Requestor Requests", there is no further processing at the Receiver.<br>– If *resultGenType*="Periodically", the Receiver shall set up a timer according to the *periodForResultGen* attribute contained in the Request message. When the timer expires, the Receiver shall retrieve the content value of each member resource and re-generate the mashup result; then it renews the timer.<br>– If *resultGenType*="When A Mashup Member Is Updated", there is no further processing at the Receiver. |
| Information in Response message | All parameters defined in Table 8.1.3-1 in oneM2M TS-0001 [1] apply with the specific details for:<br>**Content:** Address of the created <*semanticMashupInstance*> resource and address of created <*semanticMashupResult*> resource if any, according to clause 10.1.1.1 in oneM2M TS-0001 [1]. |
| Processing at Originator after receiving Response | According to clause 10.1.1.1 in oneM2M TS-0001 [1]. |
| Exceptions | According to clause 10.1.1.1 in oneM2M TS-0001 [1]. |

## 6.4.3 Retrieve <semanticMashupInstance>

This procedure shall be used for retrieving the attributes of a <*semanticMashupInstance*> resource as described in Table 6.4.3-1.

**Table 6.4.3-1: <*semanticMashupInstance*> RETRIEVE**

| <*semanticMashupInstance*> RETRIEVE | |
|---|---|
| Associated Reference Point | Mca, Mcc and Mcc'. |
| Information in Request message | All parameters defined in Table 8.1.2-3 in oneM2M TS-0001 [1] apply with the specific details for:<br>**Content:** void. |
| Processing at Originator before sending Request | According to clause 10.1.2 in oneM2M TS-0001 [1]. |
| Processing at Receiver | The Receiver shall verify the existence (including **Filter Criteria** checking, if it is given) of the target resource or the attribute and check if the Originator has appropriate privileges to retrieve information stored in the resource/attribute. Otherwise clause 10.1.2 in oneM2M TS-0001 [1] applies. |
| Information in Response message | All parameters defined in Table 8.1.3-1 in oneM2M TS-0001 [1] apply with the specific details for:<br>**Content:** attributes of the <*semanticMashupInstance*> resource as defined in the clause 9.6.54 in oneM2M TS-0001 [1]. |
| Processing at Originator after receiving Response | According to clause 10.1.2 in oneM2M TS-0001 [1]. |
| Exceptions | According to clause 10.1.2 in oneM2M TS-0001 [1]. |

-

## 6.4.4 Update <semanticMashupInstance>

This procedure as described in Table 6.4.4-1 shall be used to update an existing <*semanticMashupInstance*>, e.g. an update to its *memberStoreType* attribute.

**Table 6.4.4-1: <*semanticMashupInstance*> UPDATE**

| **<*semanticMashupInstance*> UPDATE** | |
|---|---|
| Associated Reference Point | Mca, Mcc and Mcc'. |
| Information in Request message | All parameters defined in Table 8.1.2-3 in oneM2M TS-0001 [1] apply with the specific details for:<br>**Content:** attributes of the <*semanticMashupInstance*> resource as defined the clause 9.6.54 in oneM2M TS-0001 [1] to be updated. |
| Processing at Originator before sending Request | According to clause 10.1.3 in oneM2M TS-0001 [1]. |
| Processing at Receiver | According to clause 10.1.3 in oneM2M TS-0001 [1]:<br>• If the updated attribute in the Request message is *smjpInputParameter*, the Receiver shall recalculate the semantic mashup result using the new values of input parameters.<br>• If the mashupMember attribute is updated (e.g. an existing mashup member is not available anymore and a new mashup member is identified) and *resultGenType*="When A Mashup Member Is Updated", the Hosting CSE shall re-calculate the semantic mashup result using the new mashup members.<br>• If the updated attribute in the Request message is *memberStoreType*, the Receiver needs to change the way to maintain mashup member resources. For example, if *memberStoreType* is updated from "URI Only" to "URI and Value", the Receiver needs to retrieve the content value of each mashup member resource and store the values together with URI in *mashupMember* attribute. If *memberStoreTy*pe is updated from "URI and Value" to "URI Only", the Receiver needs *mashupMember* attribute to only maintain the identifier of each mashup member.<br>• If the updated attribute in the Request message is *resultGenType*, the Receiver changes the way to calculate/generate the semantic mashup result accordingly. |
| Information in Response message | According to clause 10.1.3 in oneM2M TS-0001 [1]. |
| Processing at Originator after receiving Response | According to clause 10.1.3 in oneM2M TS-0001 [1]. |
| Exceptions | According to clause 10.1.3 in oneM2M TS-0001 [1]. |

## 6.4.5 Delete <semanticMashupInstance>

This procedure as described in Table 6.4.5-1 shall be used to delete an existing <*semanticMashupInstance*>.

-

**Table 6.4.5-1: <*semanticMashupInstance*> DELETE**

| **<*semanticMashupInstance*> DELETE** | |
|---|---|
| Associated Reference Point | Mca, Mcc and Mcc'. |
| Information in Request message | All parameters defined in Table 8.1.2-3 in oneM2M TS-0001 [1] apply. |
| Processing at Originator before sending Request | According to clause 10.1.4.1 in oneM2M TS-0001 [1]. |
| Processing at Receiver | According to clause 10.1.4.1 in oneM2M TS-0001 [1].<br>• In addition, The Receiver removes this <*semanticMashupInstance*> from the *smilID* attribute of the corresponding <*semanticMashupJobProfile*>. |
| Information in Response message | According to clause 10.1.4.1 in oneM2M TS-0001 [1]. |
| Processing at Originator after receiving Response | According to clause 10.1.4.1 in oneM2M TS-0001 [1]. |
| Exceptions | According to clause 10.1.4.1 in oneM2M TS-0001 [1]. |

# 6.5 <*mashup*> Operations

## 6.5.1 Introduction

<*mashup*> is a virtual resource because it does not have a representation. It is the child resource of a <*semanticMashupInstance*> resource. When a RETRIEVE operation is sent to the <*mashup*> resource, it triggers a calculation and generation of the mashup result based on its parent resource <*semanticMashupInstance*>.

The <*mashup*> resource type is specified in oneM2M TS-0001 [1], clause 9.6.55.

Only Retrieve operation shall be allowed on a <*mashup*> virtual resource. A Create, an Update, or a Delete operation on a <*mashup*> virtual resource shall not be supported.

## 6.5.2 Retrieve <*mashup*>

This procedure shall be used for triggering the CSE which hosts the <*semanticMashupInstance*> to recalculate mashup results and returning the mashup result back to the requestor (e.g. an AE) of this retrieve request as described in Table 6.5.2-1.

-

**Table 6.5.2-1: <*mashup*> RETRIEVE**

| **<*mashup*> RETRIEVE** | |
|---|---|
| Associated Reference Point | Mca, Mcc and Mcc'. |
| Information in Request message | All parameters defined in Table 8.1.2-3 in oneM2M TS-0001 [1] apply with the specific details for:<br>**To:** <*semanticMashupInstance*>/<*mashup*><br>**Content:** void. |
| Processing at Originator before sending Request | According to clause 10.1.2 in oneM2M TS-0001 [1]. |
| Processing at Receiver | The Receiver shall check if the Originator has appropriate privileges. Otherwise clause 10.1.2 in oneM2M TS-0001 [1] applies:<br>• The Hosting CSE triggers the recalculation of semantic mashup result for <*mashup*>'s parent resource <*semanticMashupInstance*>. The recalculated mashup result shall be stored in the <*semanticMashupInstance*>'s child resource <*semanticMashupResult*>. |
| Information in Response message | All parameters defined in Table 8.1.3-1 in oneM2M TS-0001 [1] apply with the specific details for:<br>**Content:** the mashup result, if indicated in the request. |
| Processing at Originator after receiving Response | According to clause 10.1.2 in oneM2M TS-0001 [1]. |
| Exceptions | According to clause 10.1.2 in oneM2M TS-0001 [1].<br>In addition: a timer has expired. The Receiver responds with an error. |

# 6.6 <*semanticMashupResult*> Operations

## 6.6.1 Introduction

<*semanticMashupResult*> resource stores the mashup result. It is the child resource of a <*semanticMashupInstance*> resource. A <*semanticMashupResult*> resource shall be automatically generated by a Hosting CSE when it executes a semantic mashup operation on a <*semanticMashupInstance*> resource. The < *semanticMashupResult* > resource type is specified in the clause 9.6.56 in oneM2M TS-0001 [1].

Figure 6.6.1-1 illustrates the procedure to operate a <*semanticMashupResult*> resource. A <*semanticMashupResult*> resource shall be automatically created when a Hosting CSE executes semantic mashup operation on a <*semanticMashupInstance*> resource. Only Retrieve and Delete operations shall be allowed on a <*semanticMashupResult*> resource. Detail descriptions are given in following clauses.

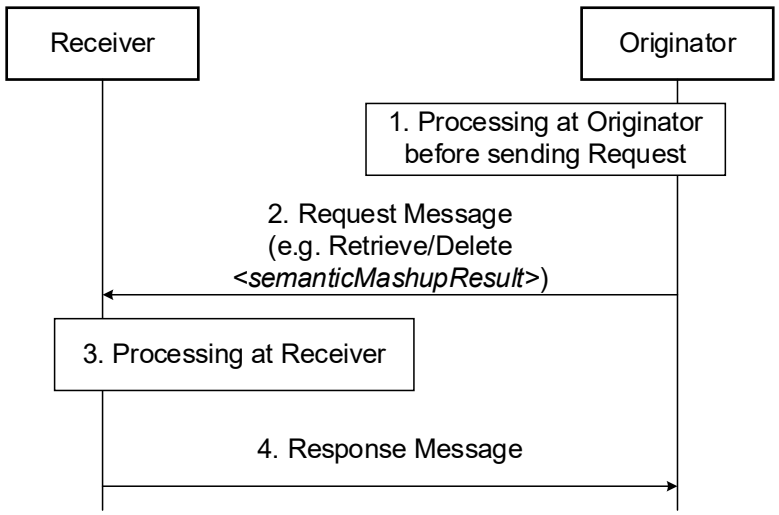


**Figure 6.6.1-1: Procedures for operating a <*semanticMashupResult*> Resource**

-

### 6.6.2 Retrieve <semanticMashupResult>

This procedure shall be used for retrieving the attributes of a <*semanticMashupResult*> resource as described in Table 6.6.2-1.

**Table 6.6.2-1: <*semanticMashupResult*> RETRIEVE**

| **<*semanticMashupResult*> RETRIEVE** | |
|---|---|
| Associated Reference Point | Mca, Mcc and Mcc'. |
| Information in Request message | All parameters defined in Table 8.1.2-3 in oneM2M TS-0001 [1] apply with the specific details for:<br>***Content*****:** void. |
| Processing at Originator before sending Request | According to clause 10.1.2 in oneM2M TS-0001 [1]. |
| Processing at Receiver | The Receiver shall verify the existence (including ***Filter Criteria*** checking, if it is given) of the target resource or the attribute and check if the Originator has appropriate privileges to retrieve information stored in the resource/attribute. Otherwise clause 10.1.2 in oneM2M TS-0001 [1] applies. |
| Information in Response message | All parameters defined in Table 8.1.3-1 in oneM2M TS-0001 [1] apply with the specific details for:<br>***Content*****:** attributes of the <*semanticMashupResult*> resource as defined in the clause 9.6.56 in oneM2M TS-0001 [1]. |
| Processing at Originator after receiving Response | According to clause 10.1.2 in oneM2M TS-0001 [1]. |
| Exceptions | According to clause 10.1.2 in oneM2M TS-0001 [1].<br>In addition: a timer has expired. The Receiver responds with an error. |

### 6.6.3 Delete <semanticMashupResult>

This procedure as described in Table 6.6.3-1 shall be used to delete an existing <*semanticMashupResult*> resource.

**Table 6.6.3-1: <*semanticMashupResult*> DELETE**

| **<*semanticMashupResult*> DELETE** | |
|---|---|
| Associated Reference Point | Mca, Mcc and Mcc'. |
| Information in Request message | All parameters defined in Table 8.1.2-3 in oneM2M TS-0001 [1] apply. |
| Processing at Originator before sending Request | According to clause 10.1.4.1 in oneM2M TS-0001 [1]. |
| Processing at Receiver | According to clause 10.1.4.1 in oneM2M TS-0001 [1]. |
| Information in Response message | According to clause 10.1.4.1 in oneM2M TS-0001 [1]. |
| Processing at Originator after receiving Response | According to clause 10.1.4.1 in oneM2M TS-0001 [1]. |
| Exceptions | According to clause 10.1.4.1 in oneM2M TS-0001 [1]. |

## 6.7 <*ontologyRepository*> Operations

### 6.7.1 Introduction

The <*ontologyRepository*> represents an ontology repository which may contain any number of managed ontologies represented as <ontology> child resources (see clause 6.8). The ontology repository may further provide semantic validation function by the <*semanticValidation*> virtual child resource (see clause 6.9).

-

## 6.7.2 Create <*ontologyRepository*>

This procedure shall be used for creating a <*ontologyRepository*> resource.

**Table 6.7.2-1: <*ontologyRepository*> CREATE**

| **<*ontologyRepository*> CREATE** | |
|---|---|
| Associated Reference Point | Mca, Mcc and Mcc' |
| Information in Request message | All parameters defined in table 8.1.2-2 in oneM2M TS-0001 [1] apply with the specific details for:<br>***Content:*** The resource content shall provide the information as defined in the clause 9.6.50 in oneM2M TS-0001 [1]. |
| Processing at Originator before sending Request | According to clause 10.1.2 in oneM2M TS-0001 [1]. |
| Processing at Receiver | According to clause 10.1.2 in oneM2M TS-0001 [1].<br>The hosting CSE shall also create the <*semanticValidation*> virtual child-resource if the addressed <*ontologyRepository*>resource is successfully created. |
| Information in Response message | According to clause 10.1.2 in oneM2M TS-0001 [1]. |
| Processing at Originator after receiving Response | According to clause 10.1.2 in oneM2M TS-0001 [1]. |
| Exceptions | According to clause 10.1.2 in oneM2M TS-0001 [1]. |

## 6.7.3 Retrieve <*ontologyRepository*>

This procedure shall be used for retrieving <*ontologyRepository*> resource.

**Table 6.7.3-1: <*ontologyRepository*> RETRIEVE**

| **<*ontologyRepository*> RETRIEVE** | |
|---|---|
| Associated Reference Point | Mca, Mcc and Mcc'. |
| Information in Request message | All parameters defined in table 8.1.2-2 in oneM2M TS-0001 [1]. |
| Processing at Originator before sending Request | According to clause 10.1.3 in oneM2M TS-0001 [1]. |
| Processing at Receiver | According to clause 10.1.3 in oneM2M TS-0001 [1]. |
| Information in Response message | All parameters defined in table 8.1.3-1 in oneM2M TS-0001 [1] apply. |
| Processing at Originator after receiving Response | According to clause 10.1.3 in oneM2M TS-0001 [1]. |
| Exceptions | According to clause 10.1.3 in oneM2M TS-0001 [1]. |

-

### 6.7.4 Update <*ontologyRepository*>

This procedure shall be used for updating an existing <*ontologyRepository*> resource.

**Table 6.7.4-1: <*ontologyRepository*> UPDATE**

| **<*ontologyRepository*> UPDATE** | |
|---|---|
| Associated Reference Point | Mca, Mcc and Mcc'. |
| Information in Request message | All parameters defined in table 8.1.2-2 in oneM2M TS-0001 [1]. |
| Processing at Originator before sending Request | According to clause 10.1.4 in oneM2M TS-0001 [1]. |
| Processing at Receiver | According to clause 10.1.4 in oneM2M TS-0001 [1]. |
| Information in Response message | According to clause 10.1.4 in oneM2M TS-0001 [1]. |
| Processing at Originator after receiving Response | According to clause 10.1.4 in oneM2M TS-0001 [1]. |
| Exceptions | According to clause 10.1.4 in oneM2M TS-0001 [1]. |

### 6.7.5 Delete <*ontologyRepository*>

This procedure shall be used for deleting an existing <*ontologyRepository*> resource.

**Table 6.7.5-1: <*ontologyRepository*> DELETE**

| **<*ontologyRepository*> DELETE** | |
|---|---|
| Associated Reference Point | Mca, Mcc and Mcc' |
| Information in Request message | All parameters defined in table 8.1.2-2 apply in oneM2M TS-0001 [1]. |
| Processing at Originator before sending Request | According to clause 10.1.5 in oneM2M TS-0001 [1]. |
| Processing at Receiver | According to clause 10.1.5 in oneM2M TS-0001 [1].<br>The hosting CSE shall also delete the <*semanticValidation*> virtual child-resource if the addressed <*ontologyRepository*>resource is successfully created. |
| Information in Response message | According to clause 10.1.5 in oneM2M TS-0001 [1]. |
| Processing at Originator after receiving Response | According to clause 10.1.5 in oneM2M TS-0001 [1]. |
| Exceptions | According to clause 10.1.5 in oneM2M TS-0001 [1]. |

## 6.8 <*ontology*> Operations

### 6.8.1 Introduction

Each <ontology> resource represents an ontology under management in the oneM2M system. It may contain the full representation or the IRI reference of the managed ontology. It is managed by simple CRUD operations as ordinary resource or by more advanced SPARQL operations (contained in the payload of the Update and Retrieve) at the granularity of RDF-triple level.

-

## 6.8.2 Create <*ontology*>

This procedure shall be used for deleting an existing <*ontology*> resource.

**Table 6.8.2-1: <*ontology*> CREATE**

| **<*ontology*> CREATE** | |
|---|---|
| Associated Reference Point | Mca, Mcc and Mcc' |
| Information in Request message | All parameters defined in table 8.1.2-2 in oneM2M TS-0001 [1] apply with the specific details for:<br>***Content:*** The resource content shall provide the information as defined in clause 9.6.51 in oneM2M TS-0001 [1]. |
| Processing at Originator before sending Request | According to clause 10.1.2 in oneM2M TS-0001 [1]. |
| Processing at Receiver | According to clause 10.1.2 in oneM2M TS-0001 [1] with the specific details as follows:<br>• The Hosting CSE shall check that the *ontologyContent* attribute conforms to the syntax as defined in the *ontologyFormat* attribute. |
| Information in Response message | According to clause 10.1.2 in oneM2M TS-0001 [1]. |
| Processing at Originator after receiving Response | According to clause 10.1.2 in oneM2M TS-0001 [1]. |
| Exceptions | According to clause 10.1.2 in oneM2M TS-0001 [1]. |

## 6.8.3 Retrieve <*ontology*>

This procedure shall be used for deleting an existing <*ontology*> resource.

**Table 6.8.3-1: <<*ontology*> RETRIEVE**

| **<*ontology*> RETRIEVE** | |
|---|---|
| Associated Reference Point | Mca, Mcc and Mcc'. |
| Information in Request message | All parameters defined in table 8.1.2-2 in oneM2M TS-0001 [1]. |
| Processing at Originator before sending Request | According to clause 10.1.3 in oneM2M TS-0001 [1]. |
| Processing at Receiver | According to clause 10.1.3 in oneM2M TS-0001 [1]. |
| Information in Response message | All parameters defined in table 8.1.3-1 in oneM2M TS-0001 [1] apply. |
| Processing at Originator after receiving Response | According to clause 10.1.3 in oneM2M TS-0001 [1]. |
| Exceptions | According to clause 10.1.3 in oneM2M TS-0001 [1]. |

-

## 6.8.4 Update <*ontology*>

This procedure shall be used for deleting an existing <*ontology*> resource.

**Table 6.8.4-1: <*ontology*> UPDATE**

| **<*ontology*> UPDATE** | |
|---|---|
| Associated Reference Point | Mca, Mcc and Mcc' |
| Information in Request message | All parameters defined in table 8.1.2-2 in oneM2M TS-0001 [1] shall apply with the specific details for the ***Content*** request parameter which may contain the new ontology information in one of the following ways:<br>1) the full representation of the new ontology triples in the *ontologyContent* attribute; or<br>2) the new IRI of the ontology in the *ontologyContent* attribute; or<br>3) the partial representation of the new ontology as described in SPARQL statements [2] in the *semanticOpExec* attribute in the case that the *ontologyFormat* is not 'IRI'. |
| Processing at Originator before sending Request | According to clause 10.1.4 in oneM2M TS-0001 [1]. |
| Processing at Receiver | According to clause 10.1.4 in oneM2M TS-0001 [1] with the specific details as follows:<br>• Check if both *semanticOpExec* attribute and *ontologyContent* attribute exist in the the ***Content*** request parameter, if so, return an error code.<br>• In the case that the ***Content*** request parameter contains partial representation of the new ontology as described in SPARQL statements [2] in the *semanticOpExec* attribute, and the *ontologyFormat* is not set to 'IRI', the Hosting CSE shall update the *ontologyContent* attribute according to the execution result of the SPARQL statements [2].<br>• In the case that the ***Content*** request parameter contains the *ontologyContent* attribute, the Hosting CSE shall check that the *ontologyContent* attribute conforms to the syntax as defined in the *ontologyFormat* attribute. |
| Information in Response message | According to clause 10.1.4 in oneM2M TS-0001 [1]. |
| Processing at Originator after receiving Response | According to clause 10.1.4 in oneM2M TS-0001 [1]. |
| Exceptions | According to clause 10.1.4 in oneM2M TS-0001 [1]. |

## 6.8.5 Delete <*ontology*>

**Table 6.8.5-1: <*ontology*> DELETE**

| **<*ontology*> DELETE** | |
|---|---|
| Associated Reference Point | Mca, Mcc and Mcc' |
| Information in Request message | All parameters defined in table 8.1.2-2 in oneM2M TS-0001 [1] apply. |
| Processing at Originator before sending Request | According to clause 10.1.5 in oneM2M TS-0001 [1]. |
| Processing at Receiver | According to clause 10.1.5 in oneM2M TS-0001 [1]. |
| Information in Response message | According to clause 10.1.5 in oneM2M TS-0001 [1]. |
| Processing at Originator after receiving Response | According to clause 10.1.5 in oneM2M TS-0001 [1]. |
| Exceptions | According to clause 10.1.5 in oneM2M TS-0001 [1]. |

-

## 6.8.6 Semantic query on <*ontology*> resource via Retrieve

**Table 6.8.6-1: Semantic query on <*ontology*> resource via RETRIEVE**

| **Semantic query on <*ontology*> resource via RETRIEVE** | |
|---|---|
| Associated Reference Point | Mca, Mcc and Mcc' |
| Information in Request message | All parameters defined in table 8.1.2-2 in oneM2M TS-0001 [1] apply.<br>In addition, the semantic query request shall be issued as a RETRIEVE operation with:<br>1) A SPARQL query statement in the ***semanticsFilter*** condition tag of the ***Filter Criteria*** request parameter.<br>2) A ***Result Content*** request parameter with the value set to **'semantic-content'.**<br>3) A ***Semantic Query Indicator*** request parameter with the value set to 'TRUE**'.**<br>See more details in clause 7.5. |
| Processing at Originator before sending Request | According to clause 10.1.3 in oneM2M TS-0001 [1]. |
| Processing at Receiver | According to clause 10.1.3 in oneM2M TS-0001 [1] with the following specific details:<br>The hosting CSE shall execute the SPARQL query statement against the ***content*** attribute of the <*ontology*> resource and return the SPARQL result to the Originator. If the ***content*** attribute contains IRI of an external ontology, the hosting CSE shall retrieve the referenced ontology following the IRI and perform the SPARQL query against it. If the ***content*** attribute contains the RDF triples, the SPARQL query can be performed directly against it. |
| Information in Response message | According to clause 10.1.3 in oneM2M TS-0001 [1]. |
| Processing at Originator after receiving Response | According to clause 10.1.3 in oneM2M TS-0001 [1]. |
| Exceptions | According to clause 10.1.3 in oneM2M TS-0001 [1]. |

Semantic query defined in clause 7.5 shall be used for retrieving the semantic information (triples) from an <*ontology*> resource.

Some SPARQL query statement examples are given as follows:

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:

   - 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;

   - SELECT ?subject WHERE { ?subject rdfs:subClassOf + saref:Command }

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;

   - SELECT ?object WHERE { saref:SetAbsoluteLevelCommand rdfs:subClassOf + ?object }

-

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;
   - 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:

   - e.g. if information concerning property A is requested all sub-properties of A also need to be included;
   - 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}}

## 6.9 <*semanticValidation*> Operations

### 6.9.1 Introduction

The <*semanticValidation*> resource, as a virtual resource of <*ontologyRepository*> resource, is used for validating an input <*semanticDescriptor*> resource sent from an authorized originator.

### 6.9.2 Create <*semanticValidation*>

The <*semanticValidation*> resource shall be created when the parent <*ontologyRepository*> resource is created by the hosting CSE. The Create operation is not applicable via Mca, Mcc or Mcc'.

### 6.9.3 Retrieve <*semanticValidation*>

The Retrieve operation is not applicable for <*semanticValidation*>.

-

### 6.9.4 Update <*semanticValidation*>

This procedure shall be used for validating a <*semanticDescriptor*> resource contained in the Update request against its referenced ontology. The semantic validation process shall also take into account linked <*semanticDescriptor*> resources (if any) of the <*semanticDescriptor*> resource in the request.

**Table 6.9.4-1: <*semanticValidation*> UPDATE**

| **<*semanticValidation*> UPDATE** | |
|---|---|
| Associated Reference Point | Mca, Mcc and Mcc'. |
| Information in Request message | All parameters defined in table 8.1.2-3 in oneM2M TS-0001 [1]. Besides, the resource representation in the ***Content*** parameter of the request message shall be set as the <semanticDescriptor> resource to be validated. |
| Processing at Originator before sending Request | According to clause 10.1.4 in oneM2M TS-0001 [1]. |
| Processing at Receiver | The Recevier shall follow the basic procedure according to clause 10.1.4 in oneM2M TS-0001 [1], with the following specific details:<br>• retrieve the semantic description (i.e. the triples in *descriptor* attribute of), the URI to the referenced ontology in *ontologyRef* attribute and potential links to other linked <*semanticDescriptor*> resources from the <*semanticDescriptor*> resource to be validated;<br>• retrieve the referenced ontology, any linked <*semanticDescriptor*> resources and the referenced ontologies of the linked <*semanticDescriptor*> resources*;*<br>• perform semantic validation according to clause 7.10. |
| Information in Response message | According to clause 10.1.4 in oneM2M TS-0001 [1]. |
| Processing at Originator after receiving Response | According to clause 10.1.4 in oneM2M TS-0001 [1].<br>In case the Originator is the hosting CSE of the <*semanticDescriptor*> resource being validated, the Originator shall update the *semanticValidated* attribute (true or false) of the hosted <*semanticDescriptor*> resource according to the received response code accordingly. |
| Exceptions | According to clause 10.1.4 in oneM2M TS-0001 [1]. |

### 6.9.5 Delete <*semanticValidation*>

The <*semanticValidation*> resource shall be deleted when the parent <*ontologyRepository*> resource is deleted by the hosting CSE. The Delete operation is not applicable via Mca, Mcc or Mcc'.

## 6.10 <*ontologyMapping*> Operations

### 6.10.1 Introduction

The ontology mapping task shall be performed by the Create or Update operation against an <*ontologyMapping*> resource on a Hosting CSE. A Retrieve operation against the same <*ontologyMapping*> resource shall be used to get the result of ontology mapping. A Delete operation against a <*ontologyMapping*> resource shall follow the basic procedure as specified in clause [1].

### 6.10.2 Create <*ontologyMapping*> (Ontology Mapping)

This procedure shall be used for performing the ontology mapping task by creating a <*ontologyMapping*> resource as described in Table 6.10.2-1. Detailed message flows are described in Figure 6.10.2-1.

-

**Table 6.10.2-1: <*ontologyMapping*> CREATE**

| <*ontologyMapping*> CREATE | |
|---|---|
| Associated Reference Point | Mca, Mcc and Mcc'. |
| Information in Request message | All parameters defined in [1] table 8.1.2-2 apply with the specific details for:<br>**Content**: The resource content shall provide the information about an <*ontologyMapping*> resource (e.g. attribute values) as described in [1]. |
| Processing at Originator before sending Request | According to clause 10.1.2 in [1]. |
| Processing at Receiver | The receiver shall follow the basic procedure according to clause 10.1.2 of [1], with the following specific details:<br>● Determine the source <*ontology*> and target <*ontology*> resources to be mapped according to the *sourceOntology* and *targetOntology* attributes provided in the request;<br>● Determine the ontology mapping method according to the mapping method description including the *mappingPolicy* and *mappingAlgorithmLinks* attributes provided in the request;<br>● Retrieve the source <*ontology*> and target <*ontology*> resources from a remote CSE if needed;<br>● Retrieve the <*ontologyMappingAlgorithm*> resources from a remote CSE if needed;<br>● Create the ontology mapping relationships between the source <*ontology*> and the target <*ontology*>;<br>● Store the mapping result in the <*ontologyMapping*> resource in the successful case. |
| Information in Response message | All parameters defined in table 8.1.3-1 in [1] shall apply with the specific details for:<br>**Content:** Address of the created <*ontologyMapping*> resource, according to clause 10.1.2 in [1]. |
| Processing at Originator after receiving Response | According to clause 10.1.2 in [1]. |
| Exceptions | According to clause 10.1.2 in [1]. |

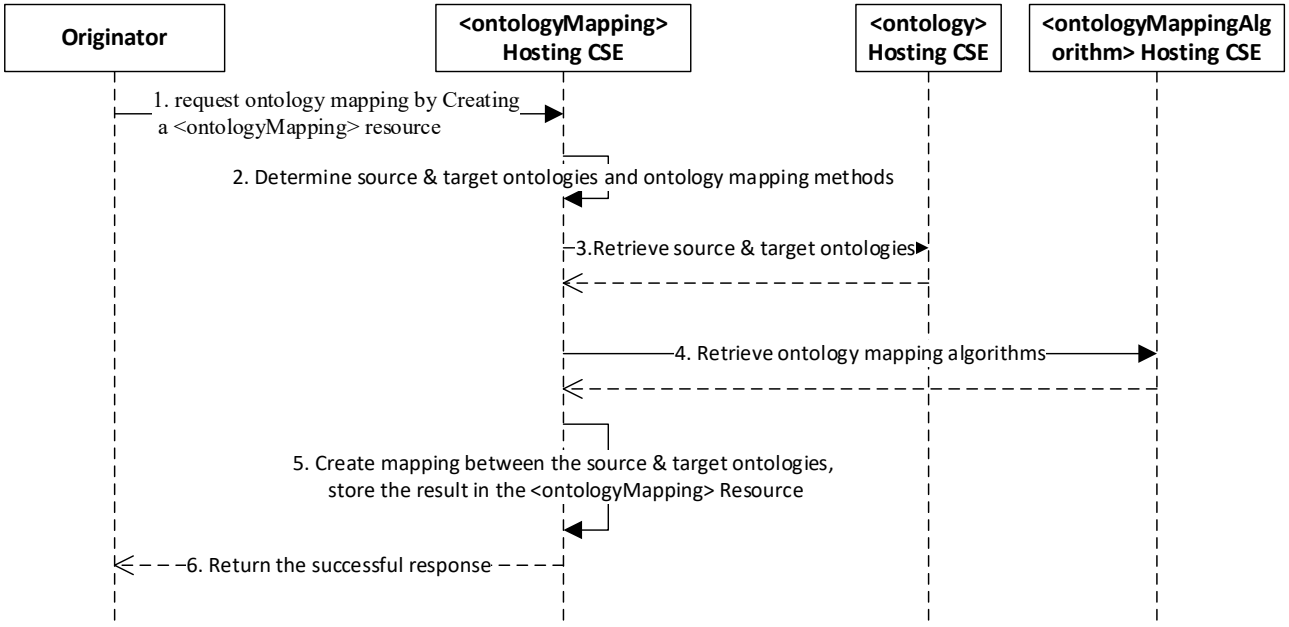


**Figure 6.10.2-1: The ontology mapping procedure by Create/Update a <*ontologyMapping*> resource**

The normal message flow Create <*ontologyMapping*> procedure is described as follows:

1. The hosting CSE (e.g. an oneM2M platform) receives an ontology mapping request from an Originator (e.g. an oneM2M application) in the form of a Create operation against an <*ontologyMapping*> resource. The request shall contain the *resourceID*s of the source and target <*ontology*> resources indicated by the *sourceOntology* and *targetOntology* attributes in the <*ontologyMapping*> resource. It shall also contain the information of mapping method description including the mapping policy and the mapping algorithms indicated by the attributes of *mappingPolicy* and *mappingAlgorithmLinks* respectively.

-

2. The hosting CSE shall determine the source and target ontologies according to the *resourceID*s of the source and target <*ontology*> resources provided in the request. It shall also determine the ontology mapping method according to the information of mapping method description provided in the request. Specifically, the hosting CSE shall first determine the mapping algorithm policy (single, multiple or traversal) according to the *mappingPolicy* attribute provided in the request. Then it shall determine the mapping algorithm(s) to be used according to the determined mapping algorithm policy and the available mapping algorithm(s) provided by the *mappingAlgorithmLinks* attribute, and shall perform the ontology mapping process according to Table 6.10.2-2.

**Table 6.10.2-2: ontology mapping process according to different mapping policies and the provided mapping algorithms**

| ***mappingPolicy*** | ***mappingAlgorithmLinks*** | ***ontology mapping process by the hosting CSE*** |
|---|---|---|
| single | contains the *resourceID*(s) of one or more existing <*ontologyMappingAlgorithm*> resources, or the *resourceID* of an existing <*ontologyMappingAlgorithmRepository*> resource. | Decide to use a single ontology mapping algorithm for the ontology mapping between the source and target ontologies.<br><br>If more than one <*ontologyMappingAlgorithm*> resource is provided by the *mappingAlgorithmLinks*, or contained in the referenced <*ontologyMappingAlgorithmRepository*> resource, the hosting CSE may decide to use one of the provided algorithms according to its local policy. |
| multiple | contains the *resourceID*s of two or more existing <*ontologyMappingAlgorithm*> resources of different types, or the *resourceID* of an existing <*ontologyMappingAlgorithmRepository*> resource which contains two or more <*ontologyMappingAlgorithm*> resources of different types. | Decide to use two or more different types of mapping algorithms (based on the *algorithmType* attribute of the <*ontologyMappingAlgorithm*> resource) for the ontology mapping between the source and target ontologies.<br><br>The hosting CSE may decide to use a subset (at least two types) of the provided algorithms according to its local policy.<br><br>If the number of the types of the provided algorithms is less than two, the hosting CSE shall reject the request with an error. |
| traversal | contains the *resourceID*(s) of one or more existing <*ontologyMappingAlgorithm*> resources, or the *resourceID* of an existing <*ontologyMappingAlgorithmRepository*> resource. | Decide to use all the provided ontology mapping algorithms in a traversal way for the ontology mapping between the source and target ontologies. |
| Note1: Any combination of *mappingPolicy* and *mappingAlgorithmLinks* not covered by this table shall be considered as an exceptional case, and the hosting CSE shall reject the request with an error.<br>Note2: For a pre-configured algorithm already stored in the system, the <*ontologyMappingAlgorithm*> resource may not contain the executable of the algorithm. In this case, the hosting CSE may invoke the algorithm from the system locally according to the *resourceName* or *resourceID* attribute.<br>Note3: If more than one algorithms are used, the final ontology mapping result shall be a union of all the results from each algorithm. | | |

3. The hosting CSE may need to retrieve the source and/or target <*ontology*> resources from a remote CSE the *sourceOntology* and *targetOntology* attributes if they are not hosted locally.

4. The hosting CSE may need to retrieve the used <*mappingAlgorithm*> resources from a remote CSE according to the *mappingAlgorithmLinks* attribute if they are not hosted locally.

5. The hosting CSE shall create the ontology mapping relationships between the source and target ontologies according to the determined mapping method, and shall store the resulted mapping relationships in the <*ontologyMapping*> resource.

Figure 6.10.2-2 shows an example of the ontology mapping between the source ontology (Ontology-A) and the target ontology (Ontology-B). Assuming the *mappingPolicy=multiple* and the *mappingAlgorithmLinks* points to two <*ontologyMappingAlgorithm*> resources which are a "linguistic-feature extraction algorithm" and an "external resource acquisition algorithm" respectively. The hosting CSE first performs the "linguistic-feature extraction algorithm" against Ontology-A and Ontology-B, and generates three mapping relationships (as formatted by

-

Triple#1, Triple#2 and Triple#3 below). The hosting CSE then performs the "external resource acquisition algorithm" with the support from external resources (e.g. the WordNet dictionary), and generates a fourth mapping relationship (as Triple#4 below).

- RDF Triple #1: Ontology-A:Thing *owl:equivalentClass* Ontology-B:Thing
- RDF Triple #2: Ontology-A:Devices *owl:equivalentClass* Ontology-B:Device
- RDF Triple #3: Ontology-A:LightSensor *owl:equivalentClass* Ontology-B:Light_Sensor
- RDF Triple #4: Ontology-A:Switch_off *owl:equivalentProperty* Ontology-B:Turn_off

The mapping relationships (RDF Triple #1/2/3/4) are stored in the *mappingResult* attribute of the <*ontologyMapping*> resource to be created.

6. The hosting CSE shall return the successful response to the Originator with the *resourceID* of the created <*ontologyMapping*> resource.

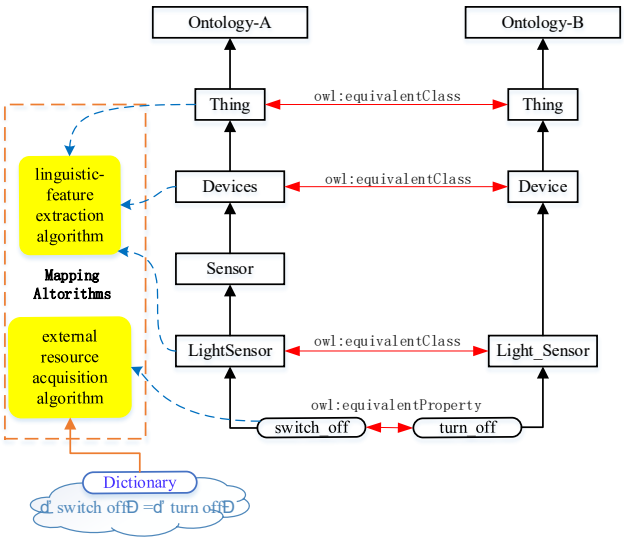


**Figure 6.10.2-2: Example of the mapping result between ontology A and ontology B**

## 6.10.3 Retrieve <*ontologyMapping*> (Get the ontology mapping result)

The ontology mapping result can be retrieved by the Retrieve operation against an <*ontologyMapping*> resource as described in Table 6.10.3-1. The mapping result is contained in the *mappingResult* attribute. No resource specific process is required.

**Table 6.10.3-1: <*ontologyMapping*> RETRIEVE**

| **<*ontologyMapping*> RETRIEVE** | |
|---|---|
| Associated Reference Point | Mca, Mcc and Mcc'. |
| Information in Request message | All parameters defined in [1] table 8.1.2-2. |
| Processing at Originator before sending Request | According to clause 10.1.3 in [1]. |
| Processing at Receiver | The receiver shall follow the basic procedure according to clause 10.1.2 of [1]. |
| Information in Response message | All parameters defined in table 8.1.3-1 apply with the specific details for:<br>***Content*:** attributes of the <*ontologyMapping*> resource as specified in [1]. The resulted mapping relationships are contained in the *mappingResult* attribute of the <*ontologyMapping*> resource. |
| Processing at Originator after receiving Response | According to clause 10.1.3 in [1]. |
| Exceptions | According to clause 10.1.3 in [1]. |

-

## 6.10.4 Update <*ontologyMapping*> (Ontology Mapping)

The ontology mapping task may also be performed by the Update operation against an <*ontologyMapping*> resource. This operation shall be used to generate new mapping results based upon an existing configuration of the <*ontologyMapping*> with only necessary modifications. The procedure is similar to the Create operation described in clause 6.10.2.

**Table 6.10.4-1: <*ontologyMapping*> UPDATE**

| **<*ontologyMapping*> UPDATE** | |
|---|---|
| Associated Reference Point | Mca, Mcc and Mcc'. |
| Information in Request message | All parameters defined in [1] table 8.1.2-2 apply with the specific details for:<br>**Content**: The resource content shall provide the information about an <*ontologyMapping*> resource (e.g. attribute values) as described in [1]. |
| Processing at Originator before sending Request | According to clause 10.1.4 in [1]. |
| Processing at Receiver | The receiver shall follow the basic procedure according to clause 10.1.4 of [1], with the following specific details:<br>● Determine the source <*ontology*> and target <*ontology*> resources to be mapped according to the *sourceOntology* and *targetOntology* attributes provided in the request;<br>● Determine the ontology mapping method according to the mapping method description including the *mappingPolicy* and *mappingAlgorithmLinks* attributes provided in the request;<br>● Retrieve the source <*ontology*> and target <*ontology*> resources from a remote CSE if needed;<br>● Retrieve the <*mappingAlgorithm*> resources from a remote CSE if needed;<br>● Create the ontology mapping relationships between the source <*ontology*> and the target <*ontology*>;<br>● Store the mapping result in the <*ontologyMapping*> resource in the successful case. |
| Information in Response message | According to clause 10.1.4 in [1]. |
| Processing at Originator after receiving Response | According to clause 10.1.4 in [1]. |
| Exceptions | According to clause 10.1.4 in [1]. |

## 6.10.5 Delete <*ontologyMapping*>

This procedure shall be used for deleting a <*ontologyMapping*> resource.

**Table 6.10.5-1: <*ontologyMapping*> DELETE**

| **<*ontologyMapping*> DELETE** | |
|---|---|
| Associated Reference Point | Mca, Mcc and Mcc' |
| Information in Request message | All parameters defined in table 8.1.2-2 in [1] apply |
| Processing at Originator before sending Request | According to clause 10.1.5 in [1] |
| Processing at Receiver | According to clause 10.1.5 in [1] |
| Information in Response message | According to clause 10.1.5 in [1] |
| Processing at Originator after receiving Response | According to clause 10.1.5 in [1] |
| Exceptions | According to clause 10.1.5 in [1] |

# 6.11 <*ontologyMappingAlgorithm*> Procedure

Ontology mapping algorithms are represented as <*ontologyMappingAlgorithm*> resources under an <*ontologyMappingAlgorithmRepository*> resource. They can be added, updated, retrieved and deleted by the CRUD

-

operation against a <*ontologyMappingAlgorithm*> resource following the basic procedures as specified in clause 10.1 in [1]. There is no resource-specific process to be defined.

# 6.12 <*ontologyMappingAlgorithmRepository*> Procedure

The CRUD operation against a <*ontologyMappingAlgorithmRepository*> resource following the basic procedures as specified in clause 10.1 in [1]. There is no resource-specific process to be defined.

# 6.13 <*semanticRuleRepository*> Operations

## 6.13.1 Introduction

A <*semanticRuleRepository*> resource is a child resource of the <*CSEBase*> resource. The <*semanticRuleRepository*> resource may have one or multiple <*reasoningRules*> child resources to represent different sets of reasoning rules in the oneM2M system. A reasoning initiator can create <*reasoningJobInstance*> child resources of a <*semanticRuleRepository*> resource to initiate desired reasoning operations.

## 6.13.2 Create <*semanticRuleRepository*>

This procedure is used for creating a <*semanticRuleRepository*> resource as described in Table 6.13.2-1.

**Table 6.13.2-1: <*semanticRuleRepository*> CREATE**

| **<*semanticRuleRepository*> CREATE** | |
|---|---|
| Associated Reference Point | Mca, Mcc and Mcc' |
| Information in Request message | All parameters defined in TS-0001 [1] table 8.1.2-3 apply with the specific details for:<br>***Content:*** The resource content provides the information as defined in the resource definition of <*semanticRuleRepository*> resource. |
| Processing at Originator before sending Request | According to clause 10.1.2 in oneM2M TS-0001 in [1]. |
| Processing at Receiver | According to clause 10.1.2 in oneM2M TS-0001 in [1]. |
| Information in Response message | All parameters defined in table 8.1.3-1 in [1] apply with the specific details for:<br>**Content**: Address of the created <*semanticRuleRepository*> resource, according to clause 10.1.2 in [1]. |
| Processing at Originator after receiving Response | According to clause 10.1.2 in oneM2M TS-0001 in [1]. |
| Exceptions | According to clause 10.1.2 in oneM2M TS-0001 in [1]. |

## 6.13.3 Retrieve <*semanticRuleRepository*>

This procedure is used for retrieving the attributes of a <*semanticRuleRepository*> resource as described in Table 6.13.3-1.

-

**Table 6.13.3-1: <*semanticRuleRepository*> RETRIEVE**

| ***<semanticRuleRepository> RETRIEVE*** | |
|---|---|
| Associated Reference Point | Mca, Mcc and Mcc'. |
| Information in Request message | All parameters defined in Table 8.1.2-3 in [1] apply. |
| Processing at Originator before sending Request | According to clause 10.1.3 in oneM2M TS-0001 in [1]. |
| Processing at Receiver | According to clause 10.1.3 in oneM2M TS-0001 in [1]. |
| Information in Response message | All parameters defined in Table 8.1.3-1 in [1] apply with the specific details for:<br>**Content:** Attributes of the <*semanticRuleRepository*> resource. |
| Processing at Originator after receiving Response | According to clause 10.1.3 in oneM2M TS-0001 in [1]. |
| Exceptions | According to clause 10.1.3 in oneM2M TS-0001 in [1]. |

## 6.13.4 Update <*semanticRuleRepository*>

This procedure is used for updating the attributes of a <*semanticRuleRepository*> resource as described in Table 6.X.4-1.

**Table 6.13.4-1: <*semanticRuleRepository*> UPDATE**

| ***<semanticRuleRepository> UPDATE*** | |
|---|---|
| Associated Reference Point | Mca, Mcc and Mcc' |
| Information in Request message | All parameters defined in Table 8.1.2-3 in [1] apply with the specific details for:<br>**Content**: Attributes of the <*semanticRuleRepository*> resource to be updated. |
| Processing at Originator before sending Request | According to clause 10.1.4 in oneM2M TS-0001 in [1]. |
| Processing at Receiver | According to clause 10.1.4 in oneM2M TS-0001 in [1]. |
| Information in Response message | According to clause 10.1.4 in oneM2M TS-0001 in [1]. |
| Processing at Originator after receiving Response | According to clause 10.1.4 in oneM2M TS-0001 in [1]. |
| Exceptions | According to clause 10.1.4 in oneM2M TS-0001 in [1]. |

## 6.13.5 Delete <*semanticRuleRepository*>

This procedure is used for deleting a <*semanticRuleRepository*> resource as described in Table 6.13.5-1.

-

**Table 6.13.5-1: <*semanticRuleRepository*> DELETE**

| ***<semanticRuleRepository> DELETE*** | |
|---|---|
| Associated Reference Point | Mca, Mcc and Mcc' |
| Information in Request message | All parameters defined in table 8.1.2-3 in [1] apply. |
| Processing at Originator before sending Request | According to clause 10.1.5 in oneM2M TS-0001 in [1]. |
| Processing at Receiver | According to clause 10.1.5 in oneM2M TS-0001 in [1]. |
| Information in Response message | According to clause 10.1.5 in oneM2M TS-0001 in [1]. |
| Processing at Originator after receiving Response | According to clause 10.1.5 in oneM2M TS-0001 in [1]. |
| Exceptions | According to clause 10.1.5 in oneM2M TS-0001 in [1]. |

# 6.14 <*reasoningRules*> Operations

## 6.14.1 Introduction

A <*reasoningRules*> resource can be used to store a set of related reasoning rules (e.g. for supporting a particular application). A <*reasoningRules*> resource is a child resource of the <*semanticRuleRepository*> resource. By performing the CRUD operations on the <*reasoningRules*> resources, various reasoning rules (e.g., user-defined reasoning rules based on business logic) can be created, discovered, retrieved, updated and deleted inside the oneM2M system.

## 6.14.2 Create <*reasoningRules*>

This procedure is used for creating a <*reasoningRules*> resource as described in Table 6.14.2-1.

**Table 6.14.2-1: <*reasoningRules*> CREATE**

| ***<reasoningRules>* CREATE** | |
|---|---|
| Associated Reference Point | Mca, Mcc and Mcc' |
| Information in Request message | All parameters defined in Table 8.1.2-3 in TS-0001 [1] apply with the specific details for:<br>***Content:*** The resource content provides the information as defined in the resource definition of <*reasoningRules*> resource. |
| Processing at Originator before sending Request | According to clause 10.1.2 in oneM2M TS-0001 in [1]. |
| Processing at Receiver | According to clause 10.1.2 in oneM2M TS-0001 in [1]. |
| Information in Response message | All parameters defined in table 8.1.3-1 in [i.3] apply with the specific details for:<br>**Content**: Address of the created <*reasoningRules*> resource, according to clause 10.1.2 in [i.3]. |
| Processing at Originator after receiving Response | According to clause 10.1.2 in oneM2M TS-0001 in [1]. |
| Exceptions | According to clause 10.1.2 in oneM2M TS-0001 in [1]. |

## 6.14.3 Retrieve <*reasoningRules*>

This procedure is used for retrieving the attributes of a <*reasoningRules*> resource as described in Table 6.14.3-1.

-

**Table 6.14.3-1: <*reasoningRules*> RETRIEVE**

| ***<reasoningRules> RETRIEVE*** | |
|---|---|
| Associated Reference Point | Mca, Mcc and Mcc'. |
| Information in Request message | All parameters defined in table 8.1.2-3 in [1] apply. |
| Processing at Originator before sending Request | According to clause 10.1.3 in oneM2M TS-0001 in [1]. |
| Processing at Receiver | According to clause 10.1.3 in oneM2M TS-0001 in [1]. |
| Information in Response message | All parameters defined in Table 8.1.3-1 in [1] apply with the specific details for:<br>**Content:** Attributes of the <*reasoningRules*> resource. |
| Processing at Originator after receiving Response | According to clause 10.1.3 in oneM2M TS-0001 in [1]. |
| Exceptions | According to clause 10.1.3 in oneM2M TS-0001 in [1]. |

## 6.14.4 Update <*reasoningRules*>

This procedure is used for updating the attributes of a <*reasoningRules*> resource as described in Table 6.14.4-1.

**Table 6.14.4-1: <*reasoningRules*> UPDATE**

| ***<reasoningRules> UPDATE*** | |
|---|---|
| Associated Reference Point | Mca, Mcc and Mcc' |
| Information in Request message | All parameters defined in Table 8.1.2-3 in [1] apply with the specific details for:<br>**Content**: Attributes of the <*reasoningRules*> resource to be updated. |
| Processing at Originator before sending Request | According to clause 10.1.4 in oneM2M TS-0001 in [1]. |
| Processing at Receiver | According to clause 10.1.4 in oneM2M TS-0001 in [1]. |
| Information in Response message | According to clause 10.1.4 in oneM2M TS-0001 in [1]. |
| Processing at Originator after receiving Response | According to clause 10.1.4 in oneM2M TS-0001 in [1]. |
| Exceptions | According to clause 10.1.4 in oneM2M TS-0001 in [1]. |

## 6.14.5 Delete <*reasoningRules*>

This procedure is used for deleting a <*reasoningRules*> resource as described in table 6.14.5-1.

**Table 6.14.5-1: <*reasoningRules*> DELETE**

| ***<reasoningRules> DELETE*** | |
|---|---|
| Associated Reference Point | Mca, Mcc and Mcc' |
| Information in Request message | All parameters defined in Table 8.1.2-3 in [1] apply. |
| Processing at Originator before sending Request | According to clause 10.1.5 in oneM2M TS-0001 in [1]. |
| Processing at Receiver | According to clause 10.1.5 in oneM2M TS-0001 in [1]. |
| Information in Response message | According to clause 10.1.5 in oneM2M TS-0001 in [1]. |
| Processing at Originator after receiving Response | According to clause 10.1.5 in oneM2M TS-0001 in [1]. |
| Exceptions | According to clause 10.1.5 in oneM2M TS-0001 in [1]. |

-

## 6.15 <*reasoningJobInstance*> Operations

### 6.15.1 Introduction

A Reasoning Initiator (RI), such as an AE or CSE, may trigger two types of reasoning operations. One type is a "one-time" reasoning operation. This is applicable to the case where a reasoning operation can be executed over a Fact Set (FS) and a Rule Set (RS) that may not change over time. In comparison, the other type is a "continuous" reasoning operation. The second type is applicable to the cases where the input FS and RS for reasoning may change over time, and accordingly the previously inferred knowledge may not be valid anymore. Therefore, new reasoning is executed over the latest version of FS and RS in order to generate up-to-date inferred knowledge.

A <*reasoningJobInstance*> resource represents a specific reasoning job instance for enabling the two types of reasoning operations. A RI initiates a desired reasoning operation by creating a <*reasoningJobInstance*> resource as a child resource of a <*semanticRuleRepository*> resource.

### 6.15.2 Create <*reasoningJobInstance*>

This procedure is used for creating a <*reasoningJobInstance*> resource as described in Table 6.15.2-1.

-

**Table 6.15.2-1: <*reasoningJobInstance*> CREATE**

| <*reasoningJobInstance*> CREATE | |
|---|---|
| Associated Reference Point | Mca, Mcc and Mcc' |
| Information in Request message | All parameters defined in [1] table 8.1.2-3 apply with the specific details for:<br>***Content:*** The resource content provides the information as defined in the resource definition of <*reasoningJobInstance*> resource. |
| Processing at Originator before sending Request | According to clause 10.1.2 in oneM2M TS-0001 in [1]. |
| Processing at Receiver | The Receiver follows the basic procedure according to clause 10.1.4 in oneM2M TS-0001 [1], with the following specific details:<br>1. The receiver first retrieves the facts from the resources referred to by the *factSet* attribute. For example,<br>o If a referred resource is a type of <*semanticDescriptor*> resource, the RDF triples included in the *descriptor* attribute will be collected.<br>o If a referred resource is a type of <*ontology*> resource, the data included in the *ontologyContent* attribute will be collected.<br>2. The receiver retrieves all the related reasoning rules for the resources referred to by the *ruleSet* attribute. For example,<br>o If a referred resource is a <*reasoningRules*> resource, the rules included in the *ruleRepresentation* attribute will be collected.<br>3. The receiver includes the retrieved facts and rules from the previous steps, as well as optional facts/rules based on local policies, as inputs for the semantic reasoning operation. The receiver performs semantic reasoning processing using these inputs and produces the reasoning result and stores it in the *resultRepresentation* attribute of the created <*reasoningJobInstance*> resource.<br>4. If the created <*reasoningJobInstance*> resource represents a continuous reasoning operation (i.e., the *reasoningType* attribute is set to “continuous”), subsequent reasoning processing will be automatically triggered and performed according to the values of *reasoningMode* and *reasoningPeriod* attributes and the *resultRepresentation* attribute will be overwritten with the latest reasoning result. |
| Information in Response message | All parameters defined in table 8.1.3-1 in [1] apply with the specific details for:<br>***Content***: Address of the created <*reasoningJobInstance*> resource, according to clause 10.1.2 in [1]. |
| Processing at Originator after receiving Response | According to clause 10.1.2 in oneM2M TS-0001 in [1]. |
| Exceptions | According to clause 10.1.2 in oneM2M TS-0001 in [1]. |

## 6.15.3 Retrieve <*reasoningJobInstance*>

This procedure is used for retrieving the attributes of a <*reasoningJobInstance*> resource as described in Table 6.15.3-1.

-

**Table 6.15.3 -1: <*reasoningJobInstance*> RETRIEVE**

| ***<reasoningJobInstance> RETRIEVE*** | |
|---|---|
| Associated Reference Point | Mca, Mcc and Mcc'. |
| Information in Request message | All parameters defined in table 8.1.2-3 in [1] apply. |
| Processing at Originator before sending Request | According to clause 10.1.3 in oneM2M TS-0001 in [1]. |
| Processing at Receiver | According to clause 10.1.3 in oneM2M TS-0001 in [1]. |
| Information in Response message | All parameters defined in table 8.1.3-1 in [i.3] apply with the specific details for:<br>***Content*:** Attributes of the <*reasoningJobInstance*> resource. |
| Processing at Originator after receiving Response | According to clause 10.1.3 in oneM2M TS-0001 in [1]. |
| Exceptions | According to clause 10.1.3 in oneM2M TS-0001 in [1]. |

## 6.15.4 Update <*reasoningJobInstance*>

This procedure is used for updating the attributes of a <*reasoningJobInstance*> resource as described in Table 6.15.4-1.

**Table 6.15.4-1: <*reasoningJobInstance*> UPDATE**

| ***<reasoningJobInstance> UPDATE*** | |
|---|---|
| Associated Reference Point | Mca, Mcc and Mcc' |
| Information in Request message | All parameters defined in table 8.1.2-3 in [1] apply with the specific details for:<br>***Content***: Attributes of the <*reasoningJobInstance*> to be updated. |
| Processing at Originator before sending Request | According to clause 10.1.4 in oneM2M TS-0001 in [1]. |
| Processing at Receiver | According to clause 10.1.4 in oneM2M TS-0001 in [1]. |
| Information in Response message | According to clause 10.1.4 in oneM2M TS-0001 in [1]. |
| Processing at Originator after receiving Response | According to clause 10.1.4 in oneM2M TS-0001 in [1]. |
| Exceptions | According to clause 10.1.4 in oneM2M TS-0001 in [1]. |

## 6.15.5 Delete <*reasoningJobInstance*>

This procedure is used for deleting a <*reasoningJobInstance*> resource as described in Table 6.15.5-1.

**Table 6.15.5-1: <*reasoningJobInstance*> DELETE**

| ***<reasoningJobInstance> DELETE*** | |
|---|---|
| Associated Reference Point | Mca, Mcc and Mcc' |
| Information in Request message | All parameters defined in table 8.1.2-3 in [1] apply. |
| Processing at Originator before sending Request | According to clause 10.1.5 in oneM2M TS-0001 in [1]. |
| Processing at Receiver | According to clause 10.1.5 in oneM2M TS-0001 in [1]. |
| Information in Response message | According to clause 10.1.5 in oneM2M TS-0001 in [1]. |
| Processing at Originator after receiving Response | According to clause 10.1.5 in oneM2M TS-0001 in [1]. |
| Exceptions | According to clause 10.1.5 in oneM2M TS-0001 in [1]. |

-

# 7 Functional Descriptions

## 7.1 Overview

This clause specifies functional operations of semantic functions including access control, semantics annotation, semantic filtering and discovery, semantic queries and query scope, semantics reasoning, semantics mashup, semantics-based data analytics, ontology management, and semantics validation. Some of these functional operations are based on the basic resource procedures as specified in clause 6.

## 7.2 Access Control

### 7.2.1 Direct ACP control via semantic graph store

#### 7.2.1.1 Introduction

When realizing semantic functionalities and operations (e.g. semantic resource discovery or semantic query), a centralized Semantic Graph Store (SGS) can be used in the system to store RDF triples which are collected from the <*semanticDescriptor*> resources distributed in the resource tree. However, oneM2M uses <*accessControlPolicy*> resources to define Access Control Policies (ACP) and those resources are hosted in the resource tree and referred/used by other resources through *accessControlPolicyIDs* attribute. In other words, any resource accesses (e.g. CRUD or discovery) to a specific resource shall be compliant to certain access control policies as specified by the *accessControlPolicyIDs* attribute of this resource. Since <*semanticDescriptor*> resources have their own access control policies as defined in the *accesscontrolPolicyIDs* attribute, semantic operations to be executed directly on the RDF triples stored at the SGS shall follow those access control policies in the sense that RDF triples from certain <*semanticDescriptor*> resources shall not be used or involved in a semantic operation processing if it is not allowed by the corresponding access control policies.

Figure 7.2.1.1-1 gives an example of an 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*>.

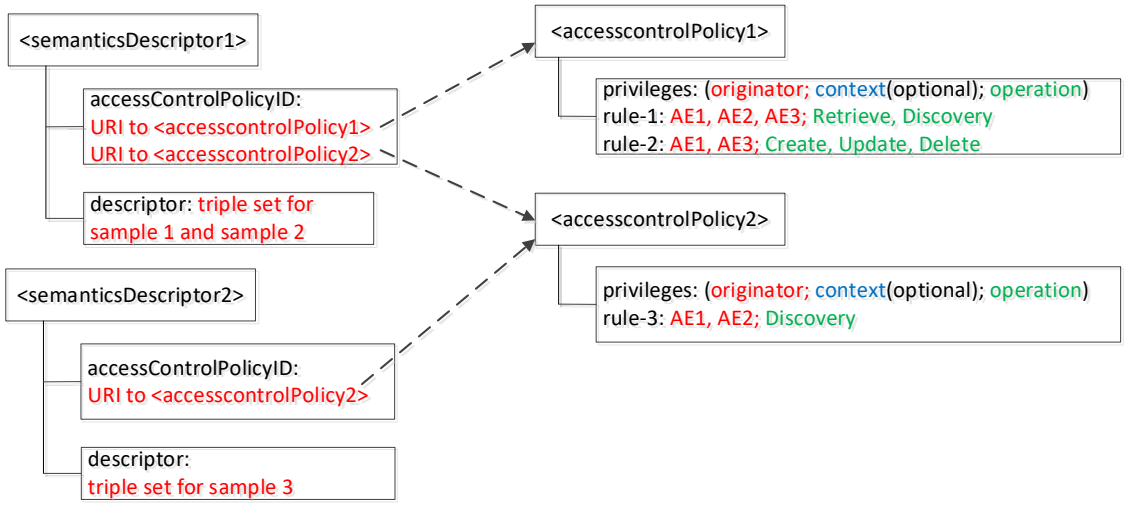


**Figure 7.2.1.1-1: Example of access control policy for <*semanticDescriptor*>**

In direct ACP control via semantic graph store, access control for any semantic operation (e.g. semantic resource discovery, semantic query, etc.) shall be directly enforced in the SGS. For this purpose, the following types of semantic triples shall be generated according to the oneM2M resource tree (i.e. <*semanticDescriptor*> and <*accessControlPolicy*> resources) and added to the SGS before the semantic operation is executed at the SGS; in

-

addition, those four types of semantic triples shall be synchronized with the oneM2M resource tree (i.e. changes on or related to <*semanticDescriptor*> and <*accessControlPolicy*> resources):

- SD Original Triples: RDF triples or other semantic representation contained in the *descriptor* attribute of a <*semanticDescriptor*> resource. In addition, the relationship between a <semanticDescriptor> resource and its *accessControlPolicyIDs* shall be represented in a semantic form and stored in the SGS.

- SD Relationship Triples: RDF triples or other semantic representation used to describe the belonging relationship between each SD Original Triple (i.e. contained in the *descriptor* attribute of a <*semanticDescriptor*> resource) and the corresponding <*semanticDescriptor*> resource.

- ACP-SD Binding Triples: RDF triples or other semantic representation used to describe which <*accessControlPolicy*> shall be applied to which <*semanticDescriptor*> (i.e. the binding relationship between a <*semanticDescriptor*> resource and its *accessControlPolicyIDs* attribute). For oneM2M, such binding relationship shall be obtained from the resource <*semanticDescriptor*>'s *accessControlPolicyIDs* attribute.

- ACP Triples: RDF triple or other semantic representation used to describe ACP policies/rules (as defined in <*accessControlPolicy*> resources) for semantic resources (e.g. <*semanticDescriptor*>).

Overall, direct ACP control via the SGS shall consist of the following tasks:

- **Task 1:** Store SD Original Triples in the SGS. Generate SD Relationship Triples, store them in the SGS. This task is detailed in clause 7.2.1.2.

- **Task 2:** Generate ACP-SD Binding Triples and ACP Triples; store them in the SGS. This task is detailed in clause 7.2.1.3.

- **Task 3:** Conduct semantic operations with direct ACP control in the SGS. Semantic operations are conducted with the selected semantic triples which are associated with the *Access Control Rules* allowing the Originator to operate (which is based on the work of Task 1 and Task 2). This task is detailed in clause 7.2.1.4.

- **Task 4:** Synchronize ACP Triples, ACP-SD Binding Triples, SD Relationship Triples, and SD Original Triples as stored in the SGS with any updated <*semanticDescriptor*> and/or <*accessControlPolicies*> resources in the oneM2M resource tree. This task is detailed in clause 7.2.1.5.

### 7.2.1.2 Create SD relationship triples

Access control policies for a <*semanticDescriptor*> resource have been defined in its *accesscontrolPolicyIDs* attribute. In other words, the granularity for defining ACP is on a resource-level in the sense that all the RDF triples stored in the *descriptor* attribute of the <*semanticDescriptor*> resource should be compliant to the same ACP as specified by the *accessControlPolicyIDs* attribute of this <*semanticDescriptor*> resource. However, when those RDF triples (i.e. SD Original Triples) are copied to the SGS, they are not stored under any resource/attribute anymore which is different than the way oneM2M resource tree works. Therefore, a way is needed to re-represent the association relationship between a SD Original Triple and its <*semanticDescriptor*> resource in SGS in order to perform the same ACP enforcement directly in the SGS. In other words, SD Relationship Triples shall be generated and stored in the SGS.

In order to do so, an internal ontology (referred to as Semantic Descriptor Ontology) with two classes *semanticDescriptor* and *atomDescription,* and several properties *describedIn*, *hasSubject hasObject* and *hasProperty* shall be used (see Figure 7.2.1.2-1). Note that the class *semanticDescriptor* is the concept to model a <*semanticDescriptor*> resource, while *atomDescription* is used to model a SD Original Triple; the *atomDescription* has four properties *describedIn*, *hasSubject hasObject* and *hasProperty*. For example, for a triple like "*classX propertyY classZ*" stored in a <*semanticDescriptor*> resource (which is termed as SD Original Triple), the following association triples shall be created for building the association and stored in the SGS; those association triples are termed as SD Relationship Triples.

| | | |
|---|---|---|
| *atomDescriptionA* | *hasSubject* | *classX* |
| *atomDescriptionA* | *hasObject* | *classZ* |
| *atomDescriptionA* | *hasproperty* | *propertyY* |
| *atomDescriptionA* | *describedIn* | *semanticDescriptorA* |

-

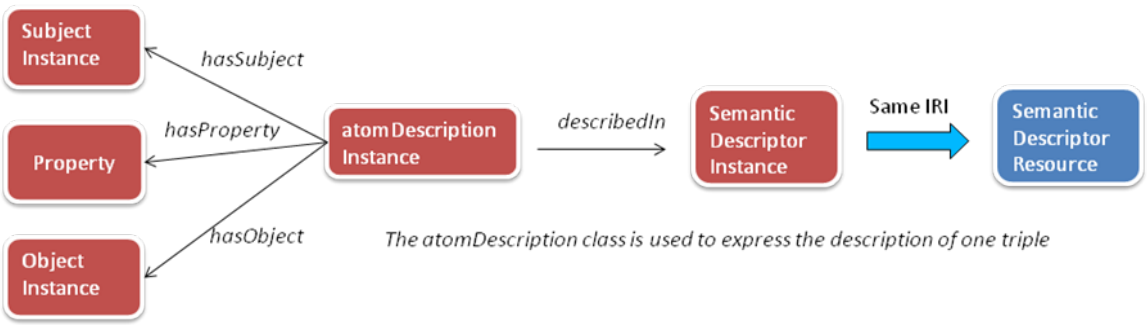


**Figure 7.2.1.2-1: Association between a SD original triple and the semanticDescriptor instance**

An illustration of such a process is shown in Figure 7.2.1.2-1. As an example, consider a <semanticDescriptor> resource called <SD-1>, which include 4 SD Original Triples:

| | | |
|---|---|---|
| HomeA | rdf:type | ex:Home. |
| HomeA | ex:hasLocation | LocationA. |
| LocationA | ex:hasLatitude | "300". |
| LocationA | ex:hasLongitude | "200". |

When those four SD Original Triple are copied to SGS, the following SD Relationship Triple shall be shall be generated:

| | | |
|---|---|---|
| @PREFIX | sd: | <http://semanticDescriptor.org>. |
| atomDescription1 | rdf:type | sd:atomDescription. |
| <SD-1> | rdf:type | sd:semanticDescriptor. |
| atomDescription1 | sd:hasSubject | HomeA. |
| atomDescription1 | sd:hasObject | ex:Home. |
| atomDescription1 | sdhasProperty | rdf:type. |
| atomDescription1 | sd:describedIn | <SD-1>. |
| atomDescription2 | sd:hasSubject | HomeA. |
| atomDescription2 | sd:hasObject | LocationA. |
| atomDescription2 | sd:hasProperty | ex:hasLocation. |
| atomDescription2 | sd:describedIn | <SD-1>. |
| atomDescription3 | sd:hasSubject | LocationA. |
| atomDescription3 | sd:hasObject | "300". |
| atomDescription3 | sd:hasProperty | ex:Latitude. |
| atomDescription3 | sd:describedIn | <SD-1>. |
| atomDescription4 | sd:hasSubject | LocationA. |
| atomDescription4 | sd:hasObject | "200". |
| atomDescription4 | sd:hasProperty | ex:hasLongtitude. |
| atomDescription4 | sd:describedIn | <SD-1>. |

-

### 7.2.1.3 Create ACP triples and ACP binding triples

#### 7.2.1.3.1 Access Control Ontology

In order to represent an ACP in a semantic form, the **Access Control Ontology** is introduced, which is shown in Figure 7.2.1.3.1-1. This ontology is defined by following how an oneM2M <*accessControlPolicy*> resource is specified in oneM2M TS-0001 [1], where an access-control-rule-tuple consists of parameters such as accessControlOriginators, accessControlOperations, and accessControlContexts. Accordingly, this ontology defines two new classes:

- accessControlPolicy; and
- accessControlRule.

In addition, five new properties (i.e. hasACPRule, hasACOriginator, hasACOperations, hasACContexts and appliedTo) are defined. More details about those terms are introduced as follows:

- The property 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 shall issue what operations under which conditions. As these triples describe the ACP themselves, they are referred to as ACP Triples.
- The property appliedTo is used to describe which <*semanticDescriptor*> resource an accessControlPolicy instance shall be applied to. As these triples bind <*accessControlPolicy*> and <*semanticDescriptor*>, they are referred to as ACP-SD Binding Triples.

**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 m 2m: <http://oneM 2M.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  ;
                          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  .
```

Access Control Triples

**Figure 7.2.1.3.1-1: Access control ontology model**

-

### 7.2.1.3.2 Example of Using Access Control Ontology

Figure 7.2.1.3.2-1 shows an example of the eHealth Ontology Reference Model, which will be used to develop the SGS example in Figure 7.2.1.3.2-2.

```
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 7.2.1.3.2-1: eHealth ontology reference model**

Figure 7.2.1.3.2-2 describes an example of ACP Triples and ACP-SD Binding Triples in the SGS, based on the <*semanticDescriptor*> resource example shown in Figure 7.2.1.1-1 and the Access Control Ontology defined in Figure 7.2.1.3.1-1. In this example, 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 7.2.1.3.2-2, which are generated based on the eHealth Ontology Reference Model in Figure 7.2.1.3.2-1.

The triples in red text in Figure 7.2.1.3.2-2 are added for access control purpose according to the proposed Access Control Ontology model in Figure 7.2.1.3.1-1, when new ACPs are created or updated. In this example, it is assumed two access control polices be created. First, two <*semanticDescriptor*> resources 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 detailed 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 7.2.1.3.2-2: eHealth triples in the SGS**

### 7.2.1.4 Conduct semantic operations with direct ACP control

This clause is to introduce more details on implementing Task-3 as discussed in clause 7.2.1.1. This clause uses semantic query as an example of semantic operatoins to be excuted with direct ACP control in the SGS.

When the Hosting CSE receives a SPARQL query from the Originator, it shall:

1) add the access control related patterns according to the ID of the Originator and the request operation to be conducted (e.g. semantic discovery) into the received SPARQL statement;

2) add ACP-SD binding related patterns into the received SPARQL statement (i.e. constraints on ACP-SD Binding Triples). For each triple pattern contained in the original SPARQL query statement, a new ACP-SD binding triple pattern shall be added;

-

3) add SD relationship related patterns (i.e. constraints on SD Relationship Triples) to the received SPARQL statement. For each triple in the original SPARQL query statement, four new triple patterns shall be added to describe the SD relationship; and

4) execute the revised SPARQL statement to make query on the SGS.

For example, in the scenario of the example in Figure 7.2.1.3.2-2, when AE-ID-3 sends the following SPARQL query request to the Hosting CSE:

```
select distinct ?sample ?sValue ?dValue
where
{
   ?sample      rdf:type        ex:BPMeasurementSample .
   ?sample      ex:sValue       ?sValue .
   ?sample      ex:dValue       ?dValue .
}
```

The Hosting CSE shall 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 is given as below (red text for ACP constraints, blue text for ACP-SD binding constraints, and orange text for SD relationship constraints):

```
select distinct ?sample ?sValue ?dValue
where
{
   ?accessControlRule       acp:hasACOriginator     "AE-ID-3" .                #---
   ?accessControlRule       acp:hasACOperations     "DISCOVERY" .              #   |---> ACP
Triples
   ?accessControlPolicy     acp:hasACPRule          ?accessControlRule .       #---

   ?accessControlPolicy     acp:appliedTo           ?semanticDescriptor1 .     #---
   ?accessControlPolicy     acp:appliedTo           ?semanticDescriptor2 .     #   |---> ACP-SD
Binding Triples
   ?accessControlPolicy     acp:appliedTo           ?semanticDescriptor3 .     #---

   ?atomDescription1        sd:describedIn          ?semanticDescriptor1 .  #---
   ?atomDescription1        sd:hasSubject           ?sample .                  #   |
   ?atomDescription1        sd:hasObject            ex:BPMeasurementSample .   #   |
   ?atomDescription1        sd:hasProperty          rdf:type .                 #   |
   ?atomDescription2        sd:describedIn          ?semanticDescriptor2 .  #   |
   ?atomDescription2        sd:hasSubject           ?sample .                  #   |---> SD
Relationship Triples
   ?atomDescription2        sd:hasObject            ?sValue .                  #   |
   ?atomDescription2        sd:hasProperty          ex:sValue .                #   |
   ?atomDescription3        sd:describedIn          ?semanticDescriptor3 .  #   |
   ?atomDescription3        sd:hasSubject           ?sample .                  #   |
   ?atomDescription3        sd:hasObject            ?dValue .                  #   |
   ?atomDescription3        sd:hasProperty          ex:dValue .                #---
  ?sample                   rdf:type                ex:BPMeasurementSample .
   ?sample                  ex:sValue               ?sValue .
   ?sample                  ex:dValue               ?dValue .
}
```

Next, the revised SPARQL query statement is excuted within the SGS. Since ACP have already been reprensented in a semantical form, the query result of the revised SPARQL query is the desried result with enforced direct acceess control. Figure 7.2.1.4-1 shows the SPARQL query result in the above example over the eHealth SGS in Figure 7.2.1.3.2-2. According to the access control triples added to the SGS (i.e. red text in Figure 7.2.1.3.2-2), AE-ID-3 is only allowed to DISCOVER samples included in semanticDescriptor1 (i.e. Sample1 and Sample2). As a result, the returned result for SPARQL query in Figure 7.2.1.4-1 presents the selected content of Sample1 and Sample2.

-

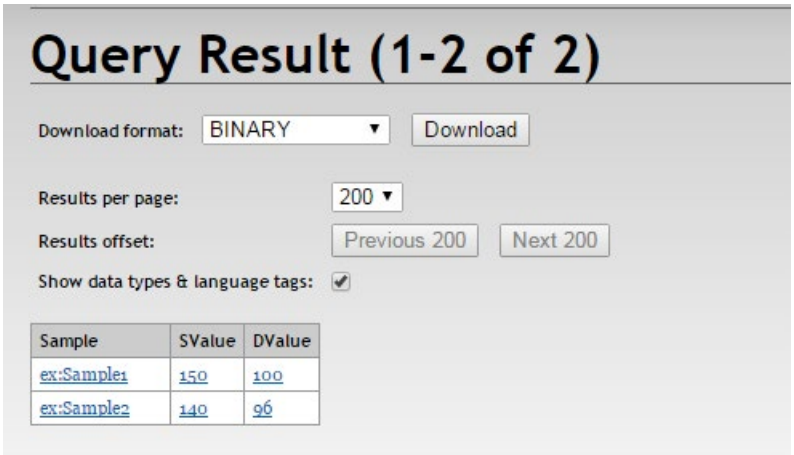


**Figure 7.2.1.4-1: Example for eHealth semantic query result with access control**

## 7.2.1.5 Synchronization ACP triples and SD-related triples in the SGS with the resource tree

### 7.2.1.5.1 Introduction

When making ACP policies/rules be also available in a semantical form (i.e. ACP Triples) in the SGS for supporting direct access control, synchronization between <*accessControlPolicy*> resources at the Hosting CSE and ACP Triples at the SGS is required. Depending on different cases, the Hosting CSE shall perform the following tasks in order to maintain such synchronization:

- When a new <*accessControlPolicy*> resource is created, the Hosting CSE shall generate new ACP Triples according to the ACP ontology and stores these new ACP Triples in the SGS (see clause 7.2.1.5.2).
- When the *privileges* attribute of an existing <*accessControlPolicy*> resource is updated, the Hosting CSE shall generate new ACP Triples and update corresponding old ACP Triples at the SGS accordingly (see clause 7.2.1.5.3).
- When an existing <*accessControlPolicy*> resource is deleted, the Hosting CSE shall remove the corresponding ACP Triples at the SGS (see clause 7.2.1.5.4).

Similar to ACP Triples, others such as SD Original Triples, SD Relationship Triples, and ACP-SD Binding Triples shall be synchronized. Depending on different cases, the Hosting CSE shall perform the following tasks in order to maintain such synchronization:

- When a <*semanticDescriptor*> is created:
  - In this case, the Hosting CSE shall generate SD Relationship Triples and ACP-SD Binding Triples and then store them in the SGS (see clause 7.2.1.5.5).
- When the *accessControlPolicyIDs* attribute of a <*semanticDescriptor*> resource changes:
  - In this case, the Hosting CSE shall generate new ACP-SD Binding Triples and use them to update corresponding old ACP-SD Binding Triples in the SGS. This case may apply also when the *accessControlPolicyIDs* attribute of the parent changes (see clause 7.2.1.5.6).
- When the descriptor attribute of a <*semanticDescriptor*> resource changes:
  - In this case, the Hosting CSE shall generate new SD Relationship Triples and use them to update old SD Relationship Triples in the SGS (see clause 7.2.1.5.7).
- When a <*semanticDescriptor*> resource is deleted:
  - In this case, the Hosting CSE shall delete all corresponding SD Original Triples, SD Relationship Triples and ACP-SD Binding Triples from the SGS (see clause 7.2.1.5.8).

-

#### 7.2.1.5.2 Procedure for creating ACP triples when a new *<accessControlPolicy>* resource is created

Figure 7.2.1.5.2-1 illustrates the procedure for creating ACP Triples in SGS, which is triggered when an Originator requests to create a new <*accessControlPolicy*> resource at the Hosting CSE.

The following steps shall be performed:

- **Step 1:** The Originator sends a request to create a new <*accessControlPolicy*> resource to the Hosting CSE. This message contains the representation of <*accessControlPolicy*> to be created (e.g. the value of privileges attribute).
- **Step 2:** The Hosting CSE receives the request in Step 1 and, subject to the Originator access rights verification, shall create the requested <*accessControlPolicy*> resource.

EXAMPLE 1: Assume <*acp1*> be the newly created ACP resource and its URI "acp1URI". Assuming <*acp1*> has one access control rule (e.g. acr11) and the URI of the corresponding privileges attribute is "acr11URI". For exemplification, assume also that acr11 allows an AE ("AE-ID-1") to perform DISCOVERY operations.

- **Step 3:** The Hosting CSE sends a response to the Originator.

EXAMPLE 2: If Step 1 was successful, "acp1URI" will be contained in this response message.

- **Step 4:** The Hosting CSE generates corresponding ACP Triples based on the content of <*acp1*> and the ACP ontology.

EXAMPLE 3: An example of ACP Triples for <*acp1*> resource created in Step 1 is illustrated in Figure 7.2.1.5.2-2.

- In Figure 7.2.1.5.2-2:
  - line#1 defines prefix "acp" which will used in lines #2-#6.
  - line#2 defines a new acp:accessControlPolicy class instance for <*acp1*> resource. The subject value of this triple (i.e. acp:acp1) is "acp1URI", therefore the subject value of this triple makes it possible to locate the corresponding resource <*acp1*>. The Hosting CSE shall also use "acp1URI" to locate corresponding triples in the SGS (e.g. when updating existing ACP Triples).
  - line#3 defines that acp:acp1 instance has an associated access control rule acr11. The object value of this triple (i.e. acp:acr11) is "acr11URI", therefore the object value of this triple, makes it possible to locate the corresponding *privileges* attribute of <acp1> resource. The Hosting CSE shall use "acr11URI" to locate the corresponding triples in the SGS (e.g. when updating existing ACP Triples).
  - line#4 defines that acp:acr11 (i.e. the object on line#3) is an acp:accessControlRule class instance.
  - line#5 and line#6 give the values of two properties of acp:acr11 based on the assumptions in this example.

NOTE: The triples on lines #4-#6 define the access control rule acr11. If <*acp1*> has more access control rules, additional access control rules will be defined similarly to those on lines #4-#6.

- Optionally: The Hosting CSE may add the address of the SGS to the <*accessControlPolicy*> resource created in Step 2 in a new attribute, to enable direct addressing of the triples.

- **Step 5:** The Hosting CSE sends a SPARQL request to store the ACP Triples created in Step 4 to the selected SGS.

EXAMPLE 4: The ACP Triples shown in Figure 7.2.1.5.2-2 will be contained in the SPARQL request.

- **Step 6:** The SGS receives the SPARQL request, processes it and saves the ACP Triples into its graph store.
- **Step 7:** The SGS sends a response back to the Hosting CSE to confirm the request in Step 6 is successfully executed.

-

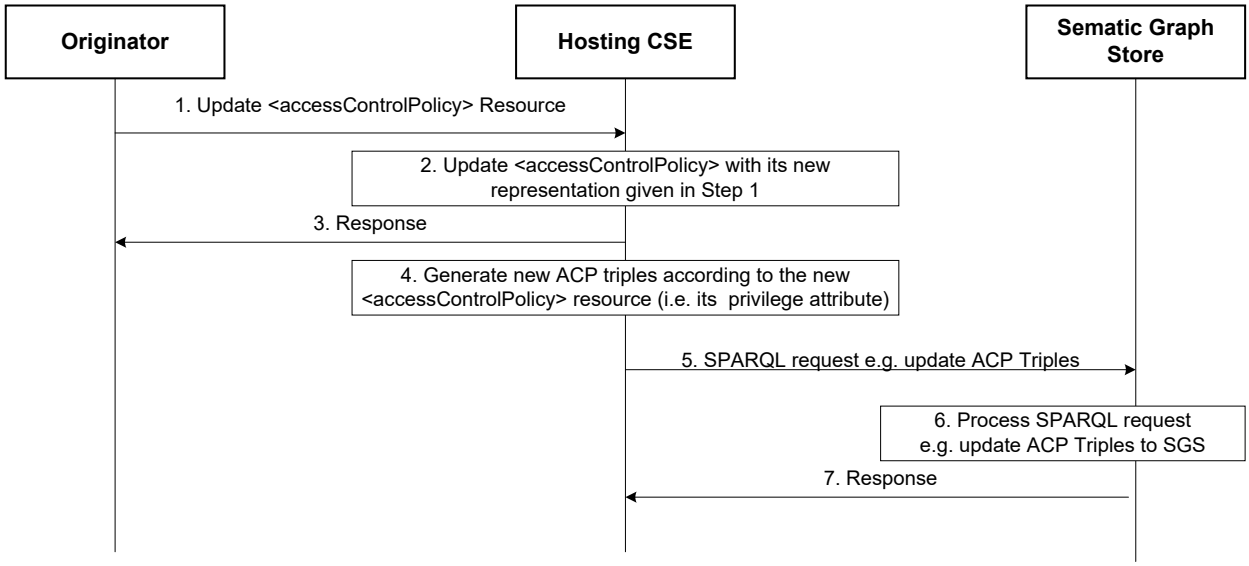


**Figure 7.2.1.5.2-1: Procedure for creating ACP triples in the SGS**

| | | | |
|---|---|---|---|
| Line#1 | @PREFIX | acp: | <http://accessControlPolicy.org> . |
| Line#2 | acp:acp1 | rdf:type | acp:accessControlPolicy . |
| Line#3 | acp:acp1 | acp:hasACPRule | acp:acr11 . |
| Line#4 | acp:acr11 | rdf:type | acp:accessControlRule . |
| Line#5 | acp:acr11 | acp:hasACOriginator | "AE-ID-1" . |
| Line#6 | acp:acr11 | acp:hasACOperations | "DISCOVERY" . |

**Figure 7.2.1.5.2-2: Example ACP triples corresponding to <acp1> resource**

### 7.2.1.5.3 Procedure for updating ACP triples when an existing *<accessControlPolicy>* resource is updated

The procedure for updating ACP Triples in a SGS follows a similar flow to the procedure used when a new *<accessControlPolicy>* resource is created. In this case the Originator requests to update the *privileges* attribute of an existing *<accessControlPolicy>* resource, as shown in Figure 7.2.1.5.3-1.

NOTE: This procedure applies also for updates of the *accessControlPolicyIDs* attribute of the *<semanticDecriptor>* resource.

The following steps shall be performed:

- **Steps 1 - 3:** Similar to those describing Figure 7.2.1.5.2-1, but reflecting normal processing of an UPDATE operation. In this case the Originator triggers an update of the *privileges* attribute of an existing *<accessControlPolicy>*.

- **Step 4:** Based on the new value of the *privileges* attribute the Hosting CSE generates new ACP Triples

EXAMPLE 1: Assume the Originator aims to update the *privileges* attribute of *<acp1>* resource from "DISCOVERY" to "DISCOVERY" and "RETRIEVE" as the new *accessControlOperations*. To implement these changes in Figure 7.2.1.5.2-2 the Hosting CSE can simply add a new triple e.g. "acp:acr11 acp:hasACOperations "RETRIEVE". " Alternatively, the Hosting CSE can replace the triple on Line#6 to the new triple "acp:acr11 acp:hasACOperations "DISCOVERY", "RETRIEVE". ".

- **Step 5:** The Hosting CSE sends a SPARQL request to the SGS to update existing ACP Triples related to <acp1> resource to reflect the update being requested:

-

**EXAMPLE 2:** As described in Step 4, there are two options to implement this.

- The Hosting CSE adds a new triple with the following SPARQL request:

```
@PREFIX      acp:                    http://accessControlPolicy.org>.
INSERT DATA
{ acp:acr11      acp:hasACOperations    "RETRIEVE". }
```

- The Hosting CSE replaces Line#6 in Figure 7.2.1.5.1-2 with the SPARQL request:

```
@PREFIX      acp:                    http://accessControlPolicy.org>.
DELETE
{?acr            acp:hasACOperations    ?operation }
WHERE
{
 ?acr                acp:hasACOperations    ?operation
 FILTER (?acr=acp:acr11)
}
INSERT DATA
{ acp:acr11      acp:hasACOperations    "DISCOVERY", "RETRIEVE". }
```

- **Step 6:** The SGS processes the received SPARQL request and updates the corresponding ACP Triples.
- **Step 7:** The SGS sends a response to the Hosting CSE to inform it whether the request has successfully executed.

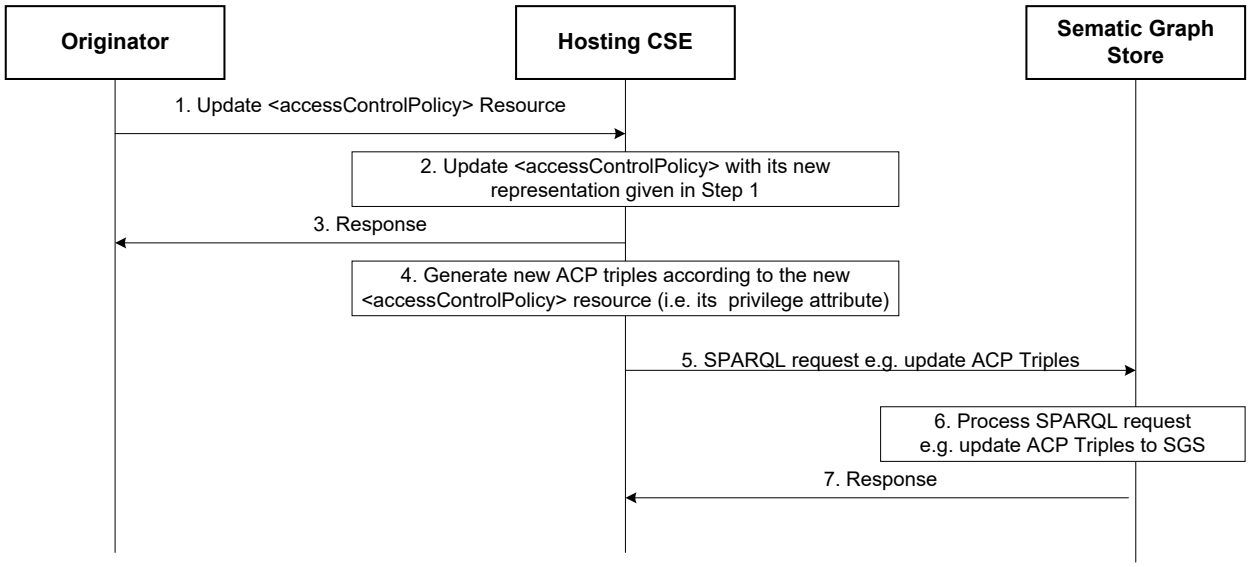


**Figure 7.2.1.5.3-1: Procedure for updating ACP triples in SGS**

### 7.2.1.5.4 Procedure for deleting ACP triples when an existing <*accessControlPolicy*> resource is deleted

The procedure for deleting ACP Triples in a SGS follows a similar flow to the procedure used when a new <*accessControlPolicy*> resource is created. In this case, the Originator requests to delete an existing <*accessControlPolicy*> resource, as shown in Figure 7.2.1.5.4-1.

The following steps shall be performed:

- **Steps 1 - 3:** Similar to those describing Figure 7.2.1.5.2-1, but reflecting normal processing of a DELETE operation. In this case the Originator triggers the deletion of an existing <*accessControlPolicy*> resource.
- **Step 4:** The Hosting CSE shall send a SPARQL request to the SGS to delete existing ACP Triples

EXAMPLE: The following SPARQL request implements this request:

-

```
@#PREFIX   acp:    http://accessControlPolicy.org>.
DELETE
{
  ?acp     ?p                  ?o
  ?s       ?p2                 ?acp
  ?acr     ?p1                 ?o1
}
WHERE
{
 ?acp      ?p                  ?o
 ?s        ?p2                 ?acp
 ?acp      acp:hasACPRule      ?acr
 ?acr      ?p1                 ?o1
 FILTER (?acp=acp:acp1)
}
```

- **Step 5:** The SGS processes the received SPARQL request and removes all requested ACP Triples.
- **Step 6:** The SGS sends a response to the Hosting CSE to inform it whether the request was successfully executed.

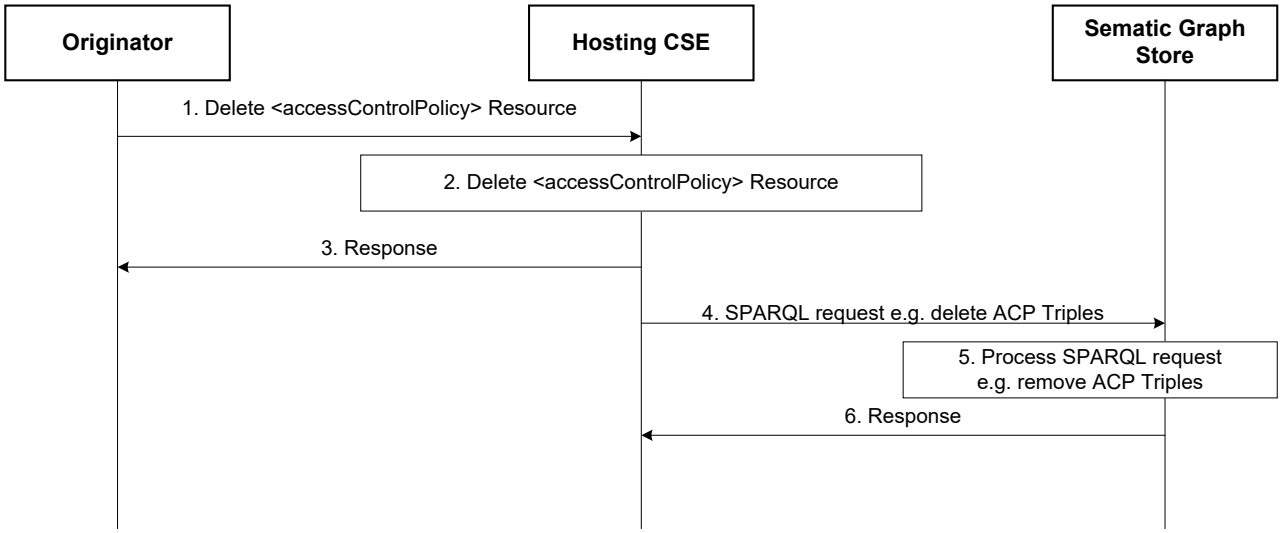


**Figure 7.2.1.5.4-1: Procedure for Deleting ACP Triples in the SGS**

### 7.2.1.5.5 Procedure for creating ACP-SD binding triples and SD relationship triples in SGS

Figure 7.2.1.5.5-1 illustrates the procedure for creating ACP-SD Binding Triples and SD Relationship Triples in SGS, which shall be triggered when an Originator requests to create a new <*semanticDescriptor*> resource.

After checking the access rights and other related security functions, the Hosting CSE shall create the <*semanticDescriptor*> resource locally (referred to as sd1 and its URI assumed to be sd1URI). Then, the Hosting CSE shall store all semantic triples as described in the descriptor attribute of SD1 resource to the SGS. More importantly, the Hosting CSE shall generate new SD Relationship Triples and ACP-SD Binding Triples and shall store them to the SGS as well. Note that if sd1 has no *accessControlPolicyIDs* attribute, ACP-SD Binding Triples shall not be generated.

The following steps shall be performed:

- **Step 1:** The Originator sends "Create <*semanticDescriptor*> Resource" request to the Hosting CSE. It is assumed that the value of descriptor attribute and *accessControlPolicyIDs* attribute of <*semanticDescriptor*> resource will be given in this request message:
  - Assume the descriptor attribute contains only one SD Original Triple "S1 P1 O1".
  - Assume the value of *accessControlPolicyIDs* is "acp1URI" i.e. the access control policy acp1 will be applied.

-

- **Step 2:** The Hosting CSE accordingly creates the <*semanticDescriptor*> resource (referred to as sd1):
  - Assume its URI is sd1URI.
- **Step 3:** The Hosting CSE sends a response to Originator to inform it if Step 2 is successfully completed.
- **Step 4:** Bases don the SD Original Triple contained in the descriptor attribute of sd1, the Hosting CSE generates SD Relationship Triples.
  - In our example, there is only one SD Original Triple, as shown below:

| | | |
|---|---|---|
| @PREFIX | sd: | <http://semanticDescriptor.org>. |
| sd:sd1 | rdf:type | sd:semanticDescriptor. |
| sd:tripleInstance11 | rdf:type | sd:atomDescription. |
| sd:tripleInstance11 | sd:describedIn | sd:sd1. |
| sd:tripleInstnace11 | sd:hasSubject | sd:S1. |
| sd:tripleInstnace11 | sd:hasProperty | sd:P1. |
| sd:tripleInstnace11 | sd:hasObject | sd:O1. |

- **Step 5:** The Hosting CSE will generate the ACP-SD Binding Triples:
  - In our example, since sd1's accessControlPolicyIDs attribute points to acp1 resource as shown below:

| | | |
|---|---|---|
| @PREFIX | sd: | <http://semanticDescriptor.org>. |
| @PREFIX | acp: | <http://accessControlPolicy.org>. |
| acp:acp1 | rdf:type | acp:accessControlPolicy. |
| sd:sd1 | rdf:type | sd:semanticDescriptor. |
| acp:acp1 | acp:appliedTo | sd:sd1. |

- **Step 6:** The Hosting CSE sends a SPARQL request to the SGS to store these SD Relationship Triples and ACP-SD Binding Triples to the SGS.
- **Step 7:** The SGS processes the SPARQL request and store corresponding SD Relationship Triples and ACP-SD Binding Triples in the SGS.
- **Step 8:** The SGS sends a response message to the Hosting CSE to inform it if the SPARQL request in Step 6 is successfully executed.

NOTE: If the <*semanticDescriptor*> resource being created in Step 2 does not have *accessControlPolicyIDs* attribute, the *accessControlPolicyIDs* attribute of the parent resource may be used or system default access privileges may be applied. The new ACP-SD Binding Triples will also be generated using either the parent resource's *accessControlPolicyIDs* attribute or based on the default privileges.

-

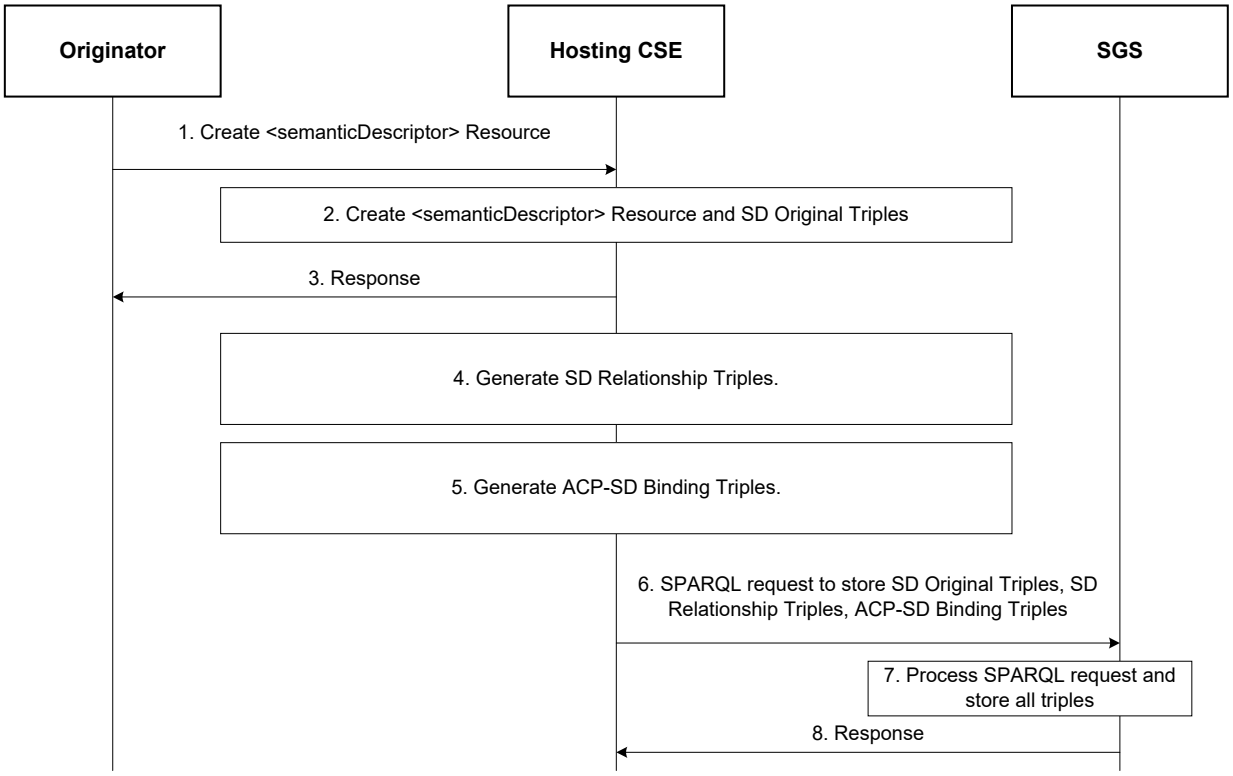


**Figure 7.2.1.5.5-1: Procedure for creating SD relationship triples and ACP-SD binding triples**

### 7.2.1.5.6 Procedure for updating ACP-SD binding triples in SGS

Figure 7.2.1.5.6-1 shows the procedure for updating ACP-SD Binding Triples when the accessControlPolicyIDs attribute of a <semanticDescriptor> resource is updated. For example, assume the sd1 resource created earlier have its *accessControlPolicyIDs* changed from acp1 to acp2; with the ACP Triples for the resource acp2 as follows:

| | | |
|---|---|---|
| @PREFIX | acp: | <http://accessControlPolicy.org>. |
| acp:acp2 | rdf:type | acp:accessControlPolicy. |
| acp:acp2 | acp:hasACPRule | acp:acr21. |
| acp:acr21 | rdf:type | acp:accessControlRule. |
| acp:acr21 | acp:hasACOriginator | "AE-ID-2". |
| acp:acr21 | acp:hasACOperations | "RETRIEVE". |

The following steps shall be performed:

- **Step 1:** The Originator sends a request to update the resource sd1's *accessControlPolicyIDs* from the URI of the resource acp1 to the URI of the resource acp2. The URI of the resource acp2 (i.e. acp2URI) is contained in this request. The URI of the resource sd1 (i.e. sd1URI) is also contained in this request.
- **Step 2:** The Hosting CSE checks access rights. If it is allowed, the Hosting CSE updates sd1's *accessControlPolicyIDs* with acp2's URI given in Step 1.
- **Step 3:** The Hosting CSE sends a response back to the Originator to inform it if the request in Step 1 is successful or not.
- **Step 4:** Since the sd1's *accessControlPolicyIDs* is changed, the Hosting CSE generates a new ACP-SD Binding Triple ("acp:acp2 acp:appliedTo sd:sd1") to reflect this change. This new ACP-SD Binding Triple will replace the old ACP-SD Binding Triple (i.e. "acp:acp1 acp:appliedTo sd:sd1"):

  | | | | |
  |---|---|---|---|
  | (new ACP-SD Binding Triple) | acp:acp2 | acp:appliedTo | sd:sd1 |
  | (old ACP-SD Binding Triple) | acp:acp1 | acp:appliedTo | sd:sd1 |

- **Step 5:** The Hosting CSE sends an SPARQL request to replace the old ACP-SD Binding Triple in the SGS with the new ACP-SD Binding Triple as shown in above Step 4. This SPARQL request for this example is shown below:

-

```
@PREFIX    acp:              <http://accessControlPolicy.org>.
@PREFIX    sd:               <http:semanticDescriptor.org>.
DELETE
{ ?acp          acp:appliedTo          sd:sd1 }
WHERE
{
?acp        acp:appliedTo              sd:sd1
}
INSERT DATA
{ acp:acp2          acp:appliedTo          sd:sd1 . }
```

- **Step 6:** The SGS processes the SPARQL request and updates the specified ACP-SD Binding Triples in Step 5.
- **Step 7:** The SGS sends a response to the Hosting CSE to inform it if the SPARQL request in Step 5 is successfully performed.

NOTE: If the *accessControlPolicyIDs* attribute of the <*semanticDescriptor*> resource was empty to start with, its parent resource's *accessControlPolicyIDs* may be enforced. The hosting CSE will apply this step based on updates to the *accessControlPolicyIDs* attribute of the parent resource.

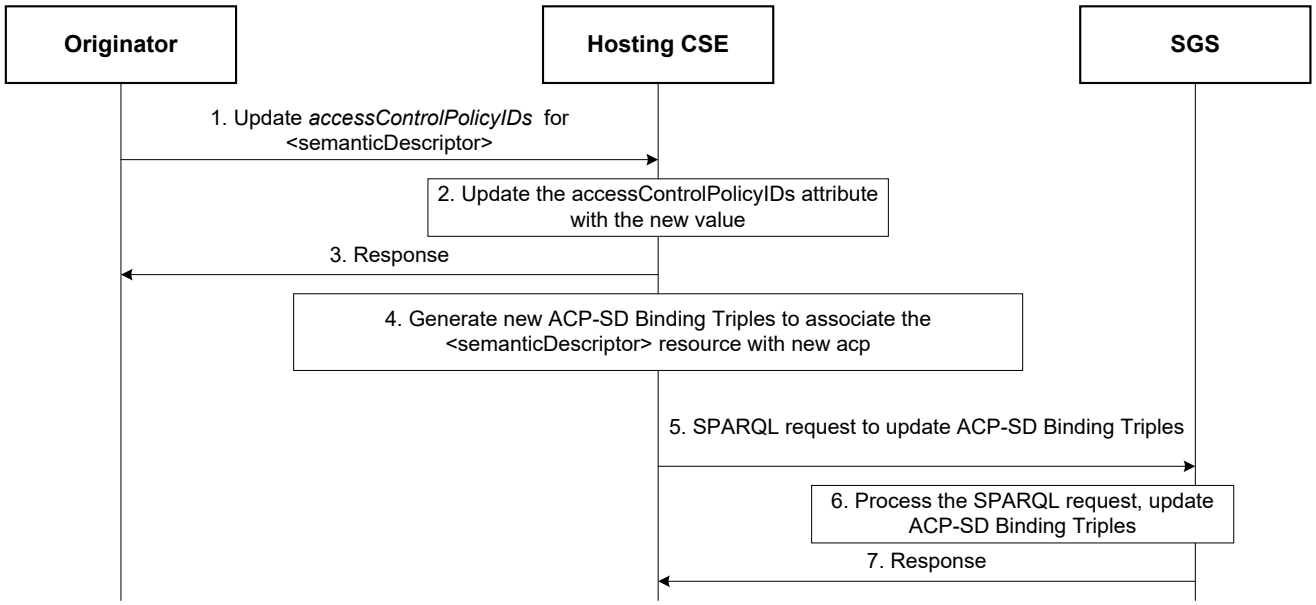


**Figure 7.2.1.5.6-1: Procedure for updating ACP-SD binding triples in the SGS**

### 7.2.1.5.7 Procedure for updating SD relationship triples in SGS

Figure 7.2.1.5.7-1 shows the procedure for updating SD Relationship Triples when the descriptor attribute of a <semanticDescriptor> resource is changed. For example, the descriptor of the sd1 resource created earlier is changed to have two SD Original Triples (Old one - "S1 P1 O1"; New one - "S2 P2 O2").

The following steps shall be performed:

- **Step 1:** The Originator sends a request to update the resource sd1's descriptor to include one new SD Original Triple (i.e. "S2 P2 O2"). The URI of the resource sd1 (i.e. sd1URI) is also contained in this request.
- **Step 2:** The Hosting CSE checks access rights. If it is allowed, the Hosting CSE updates sd1's descriptor attribute by adding one new SD Original Triple (i.e. "S2 P2 O2").
- **Step 3:** The Hosting CSE sends a response back to the Originator to inform it if the request in Step 1 is successful or not.
- **Step 4:** The Hosting CSE generates new SD Relationship Triples below to reflect this change:
  - In our example:

-

```
sd:tripleInstance12    rdf:type          sd:atomDescription.
sd:tripleInstance12    sd:describedIn    sd:sd1.
sd:tripleInstance12    sd:hasSubject     sd:S2.
sd:tripleInstance12    sd:hasProperty    sd:P2.
sd:tripleInstance12    sd:hasObject      sd:O2.
```

- **Step 5:** The Hosting CSE sends an SPARQL request to replace old SD Relationship Triples and/or add new SD Relationship Triple in the SGS with the new SD Relationship Triple generated in above Step 4. This SPARQL request for this example is shown below:

```
@PREFIX    acp:    <http://accessControlPolicy.org>.
@PREFIX    sd:     <http://semanticDescriptor.org>.

INSERT DATA
{ sd:tripleInstance12    rdf:type          sd:atomDescription.
  sd:tripleInstance12    sd:describedIn    sd:sd1 .
  sd:tripleInstance12    sd:hasSubject     sd:S2 .
  sd:tripleInstance12    sd:hasProperty    sd:P2 .
  sd:tripleInstance12    sd:hasObject      sd:O2 .
}
```

- **Step 6:** The SGS processes the SPARQL request and adds new SD Relationship Triples.
- **Step 7:** The SGS sends a response to the Hosting CSE to inform it if the SPARQL request in Step 5 is successfully performed.

NOTE: If an old SD Original Triple is removed or updated by a new SD Original Triple, the corresponding SD Relationship Triples related to this old SD Original Triple will be removed from the SGS.

The update of triples in the descriptor attribute may be performed also by targeting the *semanticOpExec* attribute of the <*semanticDescriptor*> parent resource with a SPARQL query; when this SPARQL query is executed, new SD Original Triples may be added to the *descriptor* attribute of the <*semanticDescriptor*> resource. In this case Steps 4 - 7 will be performed. More specifically, SPARQL Update consists of DELETE and ADD operations, so the SD relationship triples associated with the old original triples will be deleted, and the new ones stored.

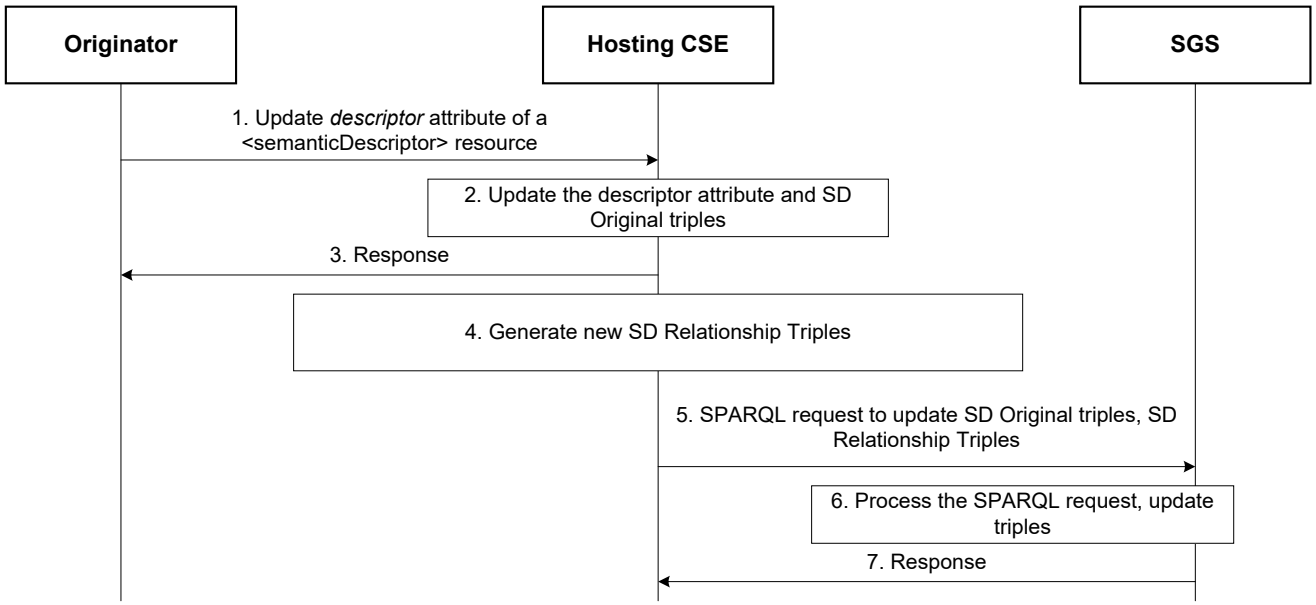


**Figure 7.2.1.5.7-1: Procedure for updating SD relationship triples in the SGS**

### 7.2.1.5.8 Procedure for deleting SD relationship triples and ACP-SD binding triples in SGS

Figure 7.2.1.5.8-1 shows a procedure for deleting SD Relationship Triples and ACP-SD Binding Triples from the SGS, which could be triggered by the Initiating AE/CSE or the Hosting CSE to delete a <*semanticDescriptor*> resource. For example, the sd1 resource created earlier is removed.

-

The following steps shall be performed.

- **Step 1:** The Originator sends "Delete <*semanticDescriptor*> Resource" to the Hosting CSE to delete sd1 resource. The URI of sd1 resource (i.e. sd1URI) is contained in this request.
- **Step 2:** The Hosting CSE deletes sd1 resource locally.
- **Step 3:** The Hosting CSE sends a response to the Originator to inform it if the deletion request in Step 1 is successful.
- **Step 4:** The Hosting CSE sends an SPARQL request to the SGS to remove SD Relationship Triples and ACP-SD Binding Triples related to sd1 resource. The SPARQL will look like:

```
@PREFIX      acp:          <http://accessControlPolicy.org>.
@PREFIX      sd:           <http:semanticDescriptor.org>.

DELETE
{ ?sd        ?p            ?o
  ?tripleInstance  ?p1     ?o1
  ?acp       acp:AppliedTo ?sd
}

WHERE
{
?sd          ?p            ?o.
?tripleInstance  ?p1       ?o1.
?tripleInstance  sd:describedIn  ?sd .
?acp         acp:AppliedTo ?sd
FILTER ( ?sd = sd:sd1)
}
```

- **Step 5:** The SGS processes the SPARQL request in Step 4 and removes corresponding SD Relationship Triples and ACP-SD Binding Triples.
- **Step 6:** The SGS sends a response to the Hosting CSE to inform it if the SPARQL request in Step 4 is successfully performed.

NOTE: Steps 4 - 6 will also be performed if a SPARQL query targeting the *semanticOpExec* attribute of a <*semanticDescriptor*> resource results in the deletion of existing SD Original Triples.

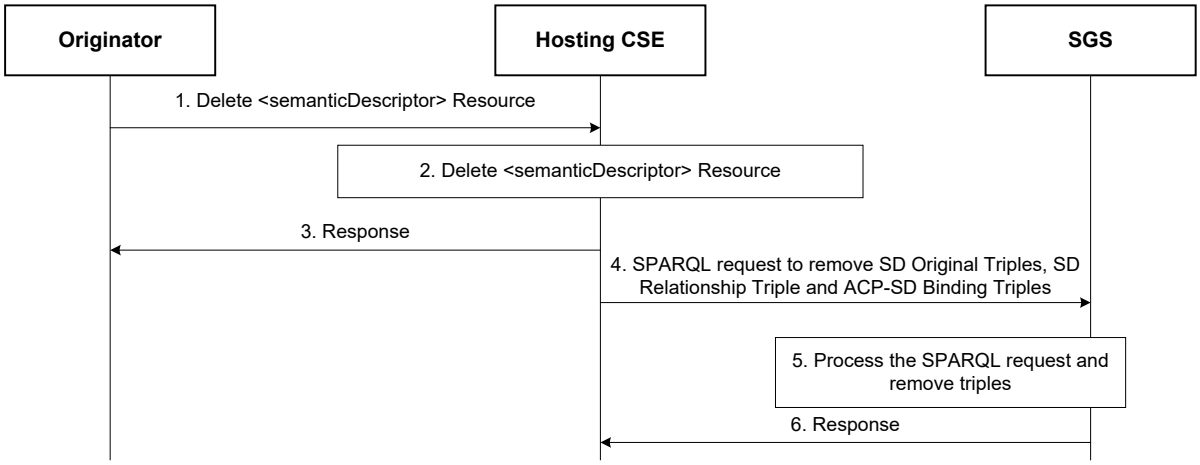


**Figure 7.2.1.5.8-1: Procedure for deleting SD relationship triples and ACP-SD binding triples from the SGS**

-

## 7.3 Semantics Annotation

Semantics annotation is defined as the process to add semantic content (i.e. <semanticDescriptor>) to an oneM2M resource (not a <semanticDescriptor> resource) as its child resource. This <semanticDescriptor> child resource provides additional semantic information about the oneM2M resource. An AE or a CSE shall use the procedures specified in clause 6.1.2 "Create <semanticDescriptor>" to add <semanticDescriptor> child resource to an oneM2M resource to fulfill the semantics annotation. In addition, the AE or the CSE can also use the procedures specified in clause 6.1.4 "Update <semanticDescriptor>" to update an existing <semanticDescriptor> and in turn update semantics annotation. Semantics annotation can be conducted for a single data item (e.g. create a <semanticDescriptor> child resource for a <contentInstance> resource); it can be also conducted for multiple data items or a data flow (e.g. create <semanticDescriptor> child resource for a <container> resource).

## 7.4 Semantic Filtering and Discovery

### 7.4.1 Introduction

NOTE: In the following descriptions, the general term *semantic resource* is used to refer to <*semanticDescriptor*> resources and <*contentInstance*> resources containing semantic information.

This clause describes semantic discovery procedures on semantic descriptions represented as RDF triples, given that an overall semantic description (logical tree) may be distributed across several semantic resources.

Semantic discovery procedures may be performed using RETRIEVE operations as follows:

**Using <*semanticFanOutPoint*> resource**

Targeting any resource other than <*semanticFanOutPoint*>:

- The receiver begins processing the request by retrieving the <*semanticDescriptor*> resource of the request target and its *descriptor* attribute. Related semantic resources are discovered and accessed according to clause 7.4.2 or clause 7.4.3. The content of related *descriptor* attributes in the case of <*semanticDescriptor*> resources or *content* attributes in the case of <*contentInstance*> resources are added to the content on which the SPARQL request is being executed. Depending on which of the options described in clauses 7.4.2 or 7.4.3 is chosen, all potentially relevant semantic content is added before executing the SPARQL request or they are added when needed during the execution of the SPARQL request.

- The resulting content subject to the SPARQL request is provided to the SPARQL engine for processing.

Targeting a <*semanticFanOutPoint*> resource (see also clause 10.2.7.12 in oneM2M TS-0001 [1]):

- In this case the related semantic resources are the members of the <*group*> resource parent of the targeted <*semanticFanOutPoint*>. Based on the *memberID* attribute of the parent <group> resource all the related descriptors are discovered, and those on the <*group*> hosting CSE are retrieved together.

- If there are semantic resources stored on a different CSE, individual RETRIEVE requests are sent to each CSE for retrieving the external resources.

- All semantic resources are retrieved based on the respective access control policies.

- Once all of the related semantic resources have been accessed, the content of each semantic attribute 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.

**Not using <*semanticFanOutPoint*> resource**

Given that an overall semantic description (logical tree) may be distributed across the semantic resources, there are two methods of constructing the logical tree in the scope of a semantic discovery targeting any resource other than <*semanticFanOutPoint*>*:*

- If the attribute *relatedSemantics* is empty or does not exist, the "Annotation-based method" (using *resourceDescriptorLink*) detailed in clause 7.4.2 shall be used.

-

- If the attribute *relatedSemantics* is not empty the "Resource link-based method" (using *relatedSemantics*) detailed in clause 7.4.3 shall be used.

## 7.4.2 Annotation-based semantic discovery method

In this option, the links to related <*semanticDescriptor*> semantic resources are encoded in the semantic description itself, which is encoded as RDF triples [6] logically structured as <subject> <predicate> <object>. For this purpose, an annotation property called *onem2m:resourceDescriptorLink* is introduced. It is formally specified as part of the oneM2M Base Ontology defined in [7] and can be used as a predicate in any RDF triple with any subject and without further relation to the oneM2M Base Ontology. Only the use of the onem2m namespace is required to uniquely identify the annotation property.

Whenever further information about a semantic instance <X> is stored in another semantic resource, a new RDF triple <X> *onem2m:resourceDescriptorLink* <ResourceURL> may be added to this semantic description, where <ResourceURL> is the URL of the other semantic resource containing additional information related to <X>. If multiple <*semanticDescriptor*> resources contain relevant further information, these can be added to a <*group*> resource and the <ResourceURL> then refers to the virtual <*fanOutPoint*> resource of this group, which will be used for retrieving the aggregated information.

NOTE: The RDF triple syntax in this paragraph is only used for illustration purposes. The actual encoding of the RDF triples used in oneM2M is defined in oneM2M TS-0004 [3].

To make use of the *onem2m:resourceDescriptorLink* property, the evaluation of semantic queries formulated as SPARQL requests by the SPARQL engine has to be adapted in the following way:

- The SPARQL request is executed on the content of the semantic description in the *descriptor* attribute of the *semanticDescriptor* resource.
- For each semantic instance matched in the SPARQL request, it is checked whether one or more *onem2m:resourceDescriptorLink* annotations exist.
- If this is the case, the execution of the SPARQL request is halted.
- The semantic content of the semantic resource referenced by the *onem2m:semanticDescriptorLink* annotations is added to the content on which the SPARQL request is being executed. If the *onem2m:semanticDescriptorLink* annotation references a group, the additional semantic content is accessed by performing a retrieve request to the virtual <*fanOutPoint*> resource referenced.
- The execution of the SPARQL request is continued on the enlarged content.

## 7.4.3 Resource link-based method

In this option, the links to related semantic resources are specified in the *relatedSemantics* attribute.

Processing of the SPARQL engine procedures at the receiver :

- The receiver retrieves the <*semanticDescriptor*> resource of the request target.
- Based on the *relatedSemantics* attribute of the <*semanticDescriptor*> resource targeted, all the related semantic resources are discovered, as follows:
    1) If the *relatedSemantics* attribute includes a list of links, each of the linked semantic resources are accessed based on the respective access control policies.
    2) If the *relatedSemantics* points to a <*group*> resource, the group members from the *memberID* attribute are used and each of them is accessed based on the respective access control policies.
- Once all of the related semantic resources have been accessed, the content of each of the descriptor attribute 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.

-

## 7.5 Semantic Queries and Query Scope

NOTE: In the following descriptions, the general term *semantic resource* is used to refer to *<semanticDescriptor>, <ontology>* resources, *<contentInstance>* resource containing semantic triples, and any other future resources containing semantic information (e.g. semantic content resources, etc.).

This clause describes semantic query procedures on semantic descriptions represented as RDF triples, given that an overall semantic description (i.e. a logical tree) may be distributed across several semantic resources.

In general, semantic queries enable the retrieval of both explicitly and implicitly derived information based on syntactic, semantic and structural information contained in data (such as RDF data). The result of a semantic query is the semantic information/knowledge for answering/matching the query. By comparison, the result of a semantic resource discovery is a list of identified resource URIs. Detailed comparison aspects between semantic query and semantic resource discovery are listed in table 7.5-1.

**Table 7.5-1: Comparison between semantic query and semantic resource discovery**

| Aspects | Semantic Query | Semantic Resource Discovery |
|---|---|---|
| Objective | The objective of Semantic Query is extracting "useful knowledge" over a set of "RDF data basis". | Semantic resource discovery is targeted to discovery of resources for further resource use (e.g. CRUD operations). |
| Technical Focus | Semantic Query is a more advanced feature leveraging semantics to derive knowledge from distributed semantic descriptors, based on a query statement. | Semantic resource discovery is a resource-oriented feature to leveraging semantics to enable sophisticated resource discovery. |
| Result | The semantic query result (representing the derived "knowledge") is provided as semantic information to answer the query not limited to resources URIs. | The processed result of a semantic resource discovery is mainly to include a list of identified resource URIs. |

A complete semantic query operation shall include the following steps:

- **Step 1:** The Originator shall be given or form a semantic query statement (i.e. using SPARQL) based on its needs.
- **Step 2:** The Originator shall form a RETRIEVE request including the semantic query statement in the *semanticsFilter* condition and shall set the "Semantic Query Indicator" parameter to "TRUE". The Originator shall send the RETRIEVE request to a Receiver.
- **Step 3:** The Receiver shall execute the semantic query statement contained in the received semantic query request, for which the following information shall be required: a) the semantic query statement which is received from the Originator; and b) the RDF data basis. The RDF data basis is composed of all the RDF triples in scope of the semantic query. The RDF data basis may be distributed in the resource tree and stored in different semantic resources. Therefore, the Receiver shall perform Semantic Graph Scoping (SGS) which is the process of establishing the "query scope", i.e. RDF data basis. An illustration of SGS is shown in Figure 7.5-1 and with two approaches described later.
- **Step 4:** Once the RDF data basis is determined through the SGS process, the Receiver shall apply the semantic query statement to the RDF data basis, yielding the semantic query result.
- **Step 5:** The semantic query result shall be included in a response message and returned to the Originator.

-

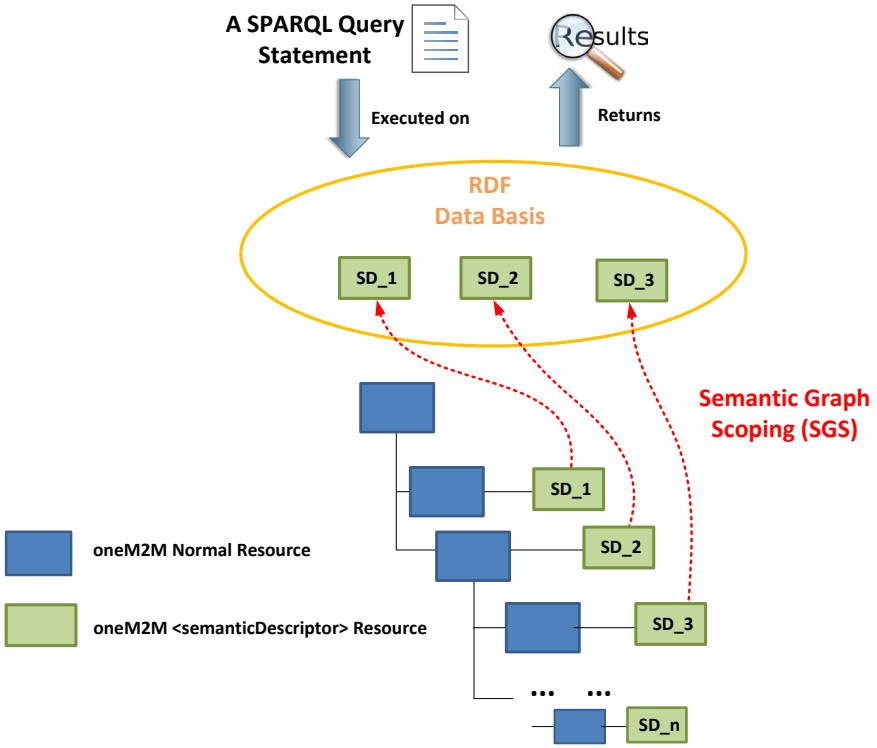


**Figure 7.5-1: An Illustration of SGS in oneM2M Architecture**

The following two approaches may be used for the SGS process in Step 3 above, in order to decide the semantic query scope of the semantic query:

**Approach-1: The scope of the semantic query is provided *implicitly*.**

In Approach-1, a semantic query request message targets any resource (i.e. as specified by the "**To**" parameter) and the semantic query shall be executed relative to this target resource, similarly to other request messages. The scope of the semantic query is formed through the aggregation of the semantic contents of the target resource's descendants. All the contents of semantic resource descendants of the target resource shall form the RDF data basis for this semantic query to be executed on. Thus, by targeting a oneM2M regular resource in the resource tree, the scope of the semantic query is implicitly decided as discussed above.

**Approach-2: The scope of the semantic query is provided *explicitly*.**

In Approach-2 the relevant semantic resources are the members of a <*group*> resource. The scope of the semantic query is formed through the aggregation of the semantic contents of all the group members. In this approach, the request targets the <*semanticFanOutPoint*> (as specified by the "**To**" parameter), i.e., the child resources of the <*group*> resource. As a result, this <*group*> resource explicitly specifies the RDF data basis of the semantic query (i.e. the scope is explicitly defined by the semantic resources which are the members of the <*group*> resource).

When the semantic query scope is explicitly defined by the <*group*> resource, the processing stage can be decoupled from the SGS process. For example, without processing any semantic query, the Receiver (e.g. a CSE) may proactively aggregate relevant semantic resources together using a <*group*> resource. The Originator may first discover various <*group*> resources and select the one with the desired RDF data basis, before launching a semantic query request. For example, the <*semanticDescriptor*> child resource of <*group*-1> resource may indicate that this group resource includes all the devices deployed in Building-1. The Originator, whose query is to be limited to Building-1, may then send its semantic query request to the <*semanticFanOutPoint*> child resource of the <*group*-1> resource.

In Approach-2, the SGS processing (included in step 3 above of the sematic query flow) shall include the following steps:

- The Receiver of the semantic query request targeting a <*semanticFanOutPoint*> resource shall use the memberIDs attribute of the parent <*group*> resource to retrieve all the related semantic information. If there are descriptors stored on different CSEs, individual RETRIEVE requests are sent to each CSE for retrieving the semantic information from the external resources.

-

- All semantic resources are accessed based on the respective access control policies. The <*semanticFanOutPoint*> resource uses *membersAccessControlPolicyIDs* attribute in the parent <*group*> resource for access control policy validation.
- Once all of the related semantic information has been retrieved (which forms the RDF data basis for this semantic query), the SPARQL query statement will be executed on the collected RDF data basis in order to provide the semantic query result.

The RETRIVE operation targeting a <*semanticFanOutPoint*> for semantic queries is detailed in clause 6.2.2.

## 7.6 Content-related Semantic Resource Discovery and Semantic Query

This clause describes the functionality supporting content-related semantic resource discovery and semantic query operations, where the SPARQL query statements pertain also to data content stored in <contentInstance> resources. For example:

• A semantic resource discovery with the content constraint: “Return URIs of sensors whose current temperature is greater than 20”.

• A semantic query with the content constraint: “Return the locations of sensors whose current temperature is greater than 20”.

These examples show that semantic resource discovery and semantic query need semantic representations of actual content which enable a variety of entities in a system, including:

• Semantic-capable data creators who can directly describe its data in a semantic form such as RDF triples;

• Semantic-incapable data creators who only can produce raw data stored as opaque content in the content attribute of the <*contentInstance*> resource and rely on other entities to add semantic annotations to the raw data;

• Semantic-capable data consumers who have the semantic resource discovery and/or semantic query capability;

• Semantic-incapable data consumers who only can retrieve raw data contents stored in the <*contentInstance*> resource through pre-configurations.

In order to enable this capability, any information that is originally stored in the content attribute of a <*contentInstance*> resource can also be represented as RDF triples and stored in certain <*semanticDescriptor*> resources (see [1] clause 9.6.7 for details). The opaque data in the content attribute of a <*contentInstance*> supports functionality for semantic-incapable data creators and consumers. The semantic formats are provided to enable semantic-capable data creators and consumers and their semantic functionality.

## 7.7 Semantics Mashup

### 7.7.1 Introduction

Existing semantic resource discovery in oneM2M can help in discovering various IoT devices and their data. However, in many application scenarios, the discovered data needs to be further processed (e.g. integrated/orchestrated/combined) based on a certain application business logic. For example, users may just be interested in a metric called "weather comfortability index", which cannot be directly provided by physical sensors, and in fact can be calculated based on the original sensory data collected from multiple types of physical sensors (e.g. temperature and humility sensors).

In general, the above process is called "**Semantic Mashup**", which is defined as a process to discover and collect data from **more than one source** as inputs, conduct a kind of **business logic-related mashup function** over the collected data, and eventually generate meaningful **mashup results**. In particular, semantic mashup emphasizes on **leveraging semantic-related technologies** during the entire mashup process. For example, in the oneM2M context, an normal resource (e.g. a <AE> resource representing a temperature sensor) may be annotated by semantic descriptions and then they could be discovered and identified as a potential data source for a specific mashup application through the semantic resource discovery.

-

The above definition also indicates a fact that a complete semantic mashup process may involve multiple stages and multiple entities for each stage. Those entities include:

- **Mashup Requestor (MR):** The entity which initiates a mashup request to Semantic Mashup Function for a certain need. In the context of oneM2M, an AE or a CSE can be an MR.
- **Resource Host (RH):** The entity which hosts data source(s) for a given mashup process. In the context of oneM2M, a data source is typically represented by a oneM2M resource (e.g. a temperature <AE> resource) and a RH will be a CSE that hosts oneM2M resources.
- **Semantic Mashup Function (SMF):** The entity which is responsible for collecting the data inputs from data sources hosted on RHs and mashing them up to generate the mashup result based on a certain business logic. In the context of oneM2M, SMF is a Common Service Function.

## 7.7.2 Semantic Mashup Function (SMF) Description

### 7.7.2.1 Introduction

Semantic mashup function including high-level architecture and high-level operations will be described in this clause.

### 7.7.2.2 High-level architecture

The high-level architecture of an SMF is shown in Figure 7.7.2.2-1, which shall contain the following components:

- **Semantic Mashup Job Profile (SMJP):** Each specific semantic mashup application has a corresponding SMJP, which not only provides functionality/interaction details for external entities to discover (e.g. MRs), but also defines the internal working details regarding how to realize this mashup application (e.g. the criteria of how to select the qualified data sources as well as the definition of mashup function). The content of an SMJP has been defined in the clause 9.6.53 in oneM2M TS-0001 [1].
- **Semantic Mashup Instance (SMI):** Once an MR identifies a desired SMJP (which can be analogous to a "job description", but not a real job), it can ask SMF to initialize a real mashup process, which corresponds to a "working instance" of this SMJP and is referred to as a Semantic Mashup Instance (SMI). In order to do so, the SMF will inject the corresponding SMJP into the Mashup Engine of SMF for the SMI instantiation, during which the engine may be involved in: 1) Identifying the qualified data sources according to the data source criteria as defined in the SMJP; 2) Collecting data inputs from those identified data sources; 3) Mashing up the collected inputs by applying mashup functions as defined in the SMJP, and finally deriving the mashup result. The content of an SMI has been defined in the clause 9.6.54 in oneM2M TS-0001 [1].

-

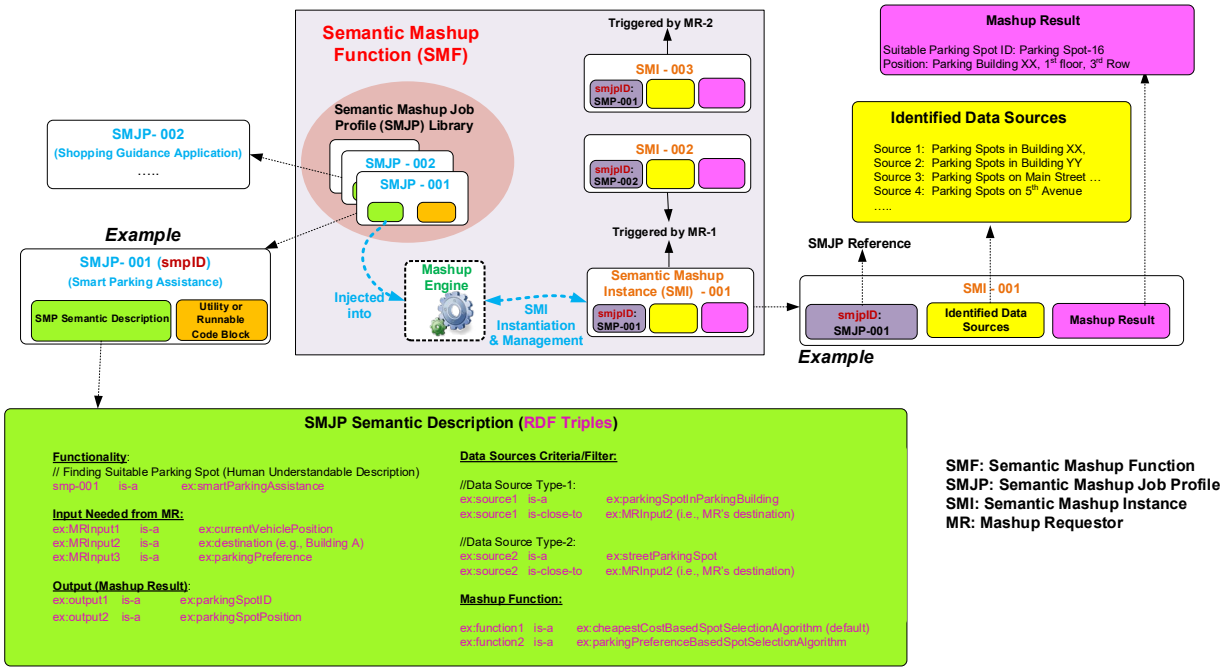


**Figure 7.7.2.2-1: High-level architecture of Semantic Mashup Function**

## 7.7.2.3 High-level operations

An SMF as introduced in clause 7.7.2.2 may involve in different tasks/operations for realizing a complete semantic mashup process. This clause is to introduce those major SMF operations. The high-level SMF operations are shown in Figure 7.7.2.3-1, where each operation shall be realized using CRUD operations as specified in the clauses 6.2.2, 6.3.2, 6.4.2 and 6.5.2, respectively:

- **Operation 1 - SMJP Discovery:** This process is needed when an MR (e.g. MR-1 in Figure 7.7.2.3-1) tries to discover a desired SMJP for its need. The procedure defined in the clause 6.3.3 for retrieving a <*semanticMashupJobProfile*> shall be leveraged for discovering <*semanticMashupJobProfile*> resources based on resource discovery procedures as defined in oneM2M TS-0001 [1].

- **Operation 2 - SMI Creation:** This process is needed when an MR already identified a desired <*semanticMashupJobProfile*> resource, but there is no corresponding SMI available for use. To implement this operation, an MR shall leverage the procedure defined in the clause 6.4.2 to send an SMI creation request to the CSE hosting SMF in order to instantiate a new SMI (i.e. <*semanticMashupInstance*> resource) for the desired SMJP. Alternatively, the SMF can also create a new SMI by itself instead of being triggered by the SMI creation request from the MR.

- **Operation 3 - Mashup Member Identification:** This process is needed when an SMF tries to identify the qualified mashup members (i.e., data sources) for a given SMI, by referring to the criteria as defined in the corresponding SMJP of this SMI (i.e. the *memberFilter* attribute of a <*semanticMashupJobProfile*> resource). Since in the oneM2M context, data sources (such as sensors) are normally represented as oneM2M resources hosted by RHs, this operation shall be implemented using semantic resource discovery mechanism as defined in clause 7.4.

-

- **Operation 4 - Mashup Result Retrieval**: This process is needed when an MR tries to retrieve the mashup result from a specific SMI. For a given SMI, it may involve in multiple rounds for mashup result generation especially when the mashup result needs to be refreshed periodically. For each round, the SMF shall collect new data inputs from identified mashup members (via Operation 5) and generate new mashup result which will be stored in the child resource <*semanticMashupResult*> of corresponding <*semanticMashupInstance*> resource. There are several alternatives for generating semantic mashup results as defined by the *resultGenType* attribute of an <*semanticMashupInstance*> resource in the clause 9.6.54 in oneM2M TS-0001 [1], for example:

  - **Option 1:** After an SMI is created, the SMF proactively and periodically runs the mashup result generation; each time before generating new mashup result, the SMF shall use Operation 5 to collect data inputs from mashup members. Whenever a new mashup result becomes available, it shall be stored in a <*semanticMashupResult*> resource and be exposed to MRs for access.

  - **Option 2:** The SMF shall generate mashup result only after receiving an explicit request from an MR (i.e. using the procedure defined in the clause 6.5.2). The benefit of this approach is that SMF works in an on-demand way, which may reduce overhead as compared to Option 1. However, the downside is that it leads to longer waiting time for an MR before the up-to-date mashup result becomes available because data re-collection and mashup result generation will not be triggered until the SMF receives a request from the MR.

- **Operation 5 - Data Input Collection and Mashup Result Generation:** This process is needed when an SMF tries to generate a mashup result for a given SMI. Note that Operation 3 focuses on how to identify the mashup members while Operation 5 focuses on how to collect data inputs from those identified/qualified mashup members. Operation 5 shall be implemented using resource retrieval mechanism as defined in oneM2M TS-0001 [1]. In addition, the working mechanism used for Operation 4 as mentioned above will affect how Operation 5 is conducted by the SMF.

- **Operation 6 - SMI Discovery and Re-use:** An SMI can be discovered, re-used and shared among different MRs. For example, the same SMI of a weather reporting mashup application for New York City Area can be shared by different users asking weather information for this area. Accordingly, Operation 6 is needed when an MR (e.g. MR-2 in Figure 7.7.2.3-1) tries to discover whether there is already an available/desired SMI ready for use. Since a given SMI is exposed as a <*semanticMashupInstance*> resource, existing resource discovery mechanism in oneM2M TS-0001 [1] shall be leveraged to discover a desired SMI from the Hosting CSE. This approach leads to less processing overhead, since other MRs do not need to require the SMF to generate a new SMI (therefore Operation 2 and 3 are not needed).

-

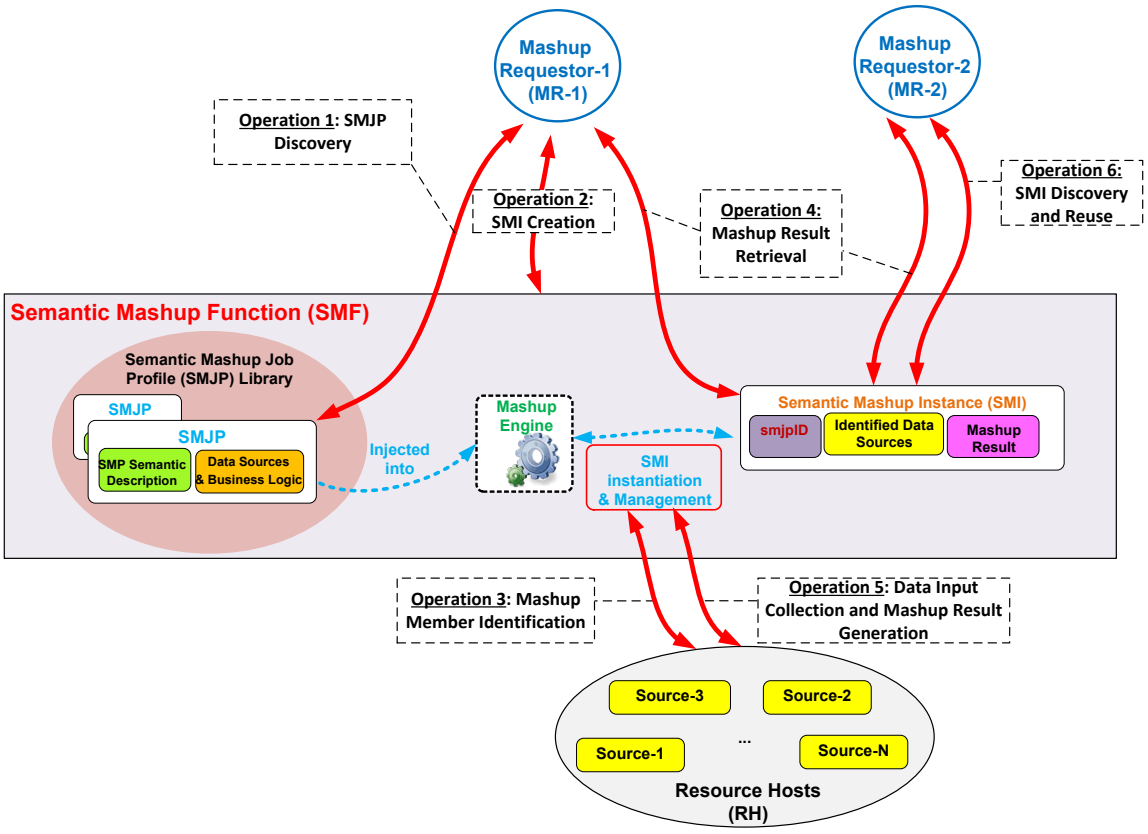


**Figure 7.7.2.3-1: High-level operations for Semantic Mashup Function**

## 7.8 Semantics-based Data Analytics

The procedures are not fully defined in this release.

## 7.9 Ontology Management

In general, the oneM2M system needs to represent knowledge as a hierarchy of concepts (ontologies), either external or internal to the oneM2M domain, using a shared vocabulary to denote the classes, properties and interrelationships of those concepts. Storage, discovery and management of ontologies (including both oneM2M Base Ontology and external ontologies e.g. SSN [i.1], SAREF [i.2]) within the oneM2M platform are key for supporting basic and advanced semantic functionalities within the oneM2M platform.

An ontology repository as represented by a <*ontologyRepository*> resource is capable of storing multiple ontologies in the unified languages adopted by the oneM2M system, e.g. RDFS/OWL. Each of the ontology under management is represented as an <*ontology*> resource in the oneM2M system. An <*ontology*> resource may contain the full representation of an ontology or the IRI reference to it. SPARQL queries can be applied directly on the <*ontology*> resource to perform semantic query and triple-level update.

An ontology repository may also provide the semantic validation service (see more in clause 7.10) via the <*semanticValidation*> child virtual resource. The service is triggered by sending a UPDATE request that contains the <*semanticDescriptor*> resource to be validated to the <*sematnicValidation*> virtual resource.

The resource type definitions of <*ontologyRepository*>, <*ontology*> and <*semanticValidation*> are specified in oneM2M TS-0001 [1], while the corresponding resource procedures are specified in clause 6 of the present document.

-

## 7.10 Semantic Validation

### 7.10.1 Introduction

The <*semanticDescriptor*> resource contains a *descriptor* attribute which can store any RDF triples as the semantic description (i.e. annotation) of the associated resource (usually the parent resource of the <*semanticDescriptor*>). In the same time, <*semanticDescriptor*> resource may also contain an *ontologyRef* attribute, which is a reference (URI) of the ontology used to represent the information that is stored in the *descriptor* attribute. Normally, the triples stored in the *descriptor* attribute should be compliant with the ontology referenced by the *ontologyRef* attribute. However, there is no guarantee that an issuer (e.g. an AE) which creates or updates the <*semanticDescriptor*> will always provide the consistent information. In case the semantic description (as triples in *descriptor* attribute) is not compliant with the referenced ontology, it basically means the <*semanticDescriptor*> is not valid and cannot be used by the AE and/or CSE properly e.g. for semantic query or reasoning.

To solve the potential inconsistency between the <*semanticDescriptor*> resources and the referenced ontology, two message flows of semantic validation are specified in the following clauses.

### 7.10.2 Semantic validation independent of <semanticDescriptor> resource operation

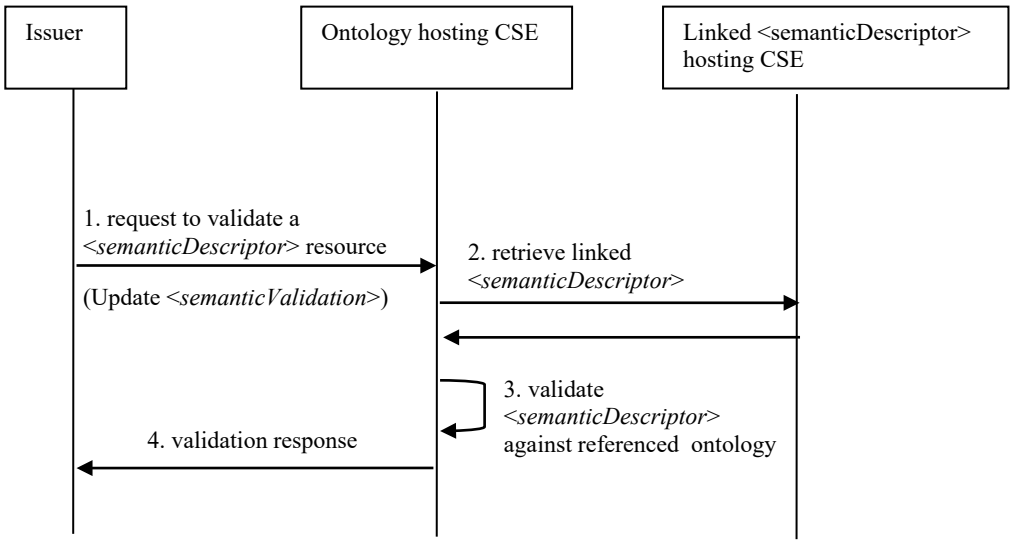


**Figure 7.10.2-1: Message flow for semantic description validation independent of <*semanticDescriptor*> resource operation**

This flow can be used independent of <*semanticDescriptor*> resource operation. For example, an AE can validate a <*semanticDescriptor*> resource after retrieving it from a hosting CSE, so as to ensure the validity of the RDF triples in the retrieved resource before using it in the application layer process (e.g. reasoning). An AE or a CSE may also choose to validate a <*semanticDescriptor*> resource representation before actually creating it in the oneM2M system.

This flow can also be used as a part of the semantic validation procedure during a <*semanticDescriptor*> resource Create or Update operation as specified in clause 7.10.3.

**Step 1.** The Issuer (e.g. an AE or CSE) shall send a semantic validation request to the ontology hosting CSE of the referenced ontology according to *ontologyRef* attribute of the <*semanticDescriptor*> resource to be validated. The request shall be an Update request addressing the <*semanticValidation*> virtual resource of the ontology hosting CSE as specified in oneM2M TS-0001 [1]. It shall contain the <*semanticDescriptor*> resource representation to be validated, which includes the semantic description (*descriptor* attribute), the URI of the referenced ontology (*ontologyRef* attribute) against which to validate, and potentially URIs (*relatedSemantics* attribute, or triples with annotation property *m2m:resourceDescriptorLink* in the *descriptor* attribute) to other linked <*semanticDescriptor*> resources that are also incorporated for validation.

-

**Step 2.** After receiving the semantic validation request, the ontology hosting CSE shall retrieve any linked <*semanticDescriptor*> resources (including the semantic description - *descriptor* and the URI of the referenced ontology - *ontologyRef*) according to the *relatedSemantics* attribute and triples with annotation property *m2m:resourceDescriptorLink* in the *descriptor* attribute of the <*semanticDescriptor*> resource in the request. In case the linked <*semanticDescriptor*> resources are further linked to more <*semanticDescriptor*> resources, the ontology hosting CSE shall repeat this step iteratively to retrieve all linked <*semanticDescriptor*> resources. In case the ontology hosting CSE cannot retrieve the linked <*semanticDescriptor*> resources (due to access right control or other exceptional reasons) within a reasonable time (according to local policy), skip **Step 3.**

**Step 3.** The ontology hosting CSE shall use the referenced ontologies (indicated by the *ontologyRef* attribute) of the received <*semanticDescriptor*> resource and the linked <*semanticDescriptor*> resources to validate the semantic description of the received <*semanticDescriptor*> resource and the linked <*semanticDescriptor*> resources all together. The aspects to be checked in semantic validation is specified in clause 7.10.4.

**Step 4.** The ontology hosting CSE shall return the validation response to the Issuer. In case **Step 3** succeeds, the response code shall indicate success of validation, otherwise (including **Step 3** is skipped due to **Step 2** fails), the response shall indicate failure of validation.

## 7.10.3 Semantic validation triggered when Create or Update a <semanticDescriptor> resource

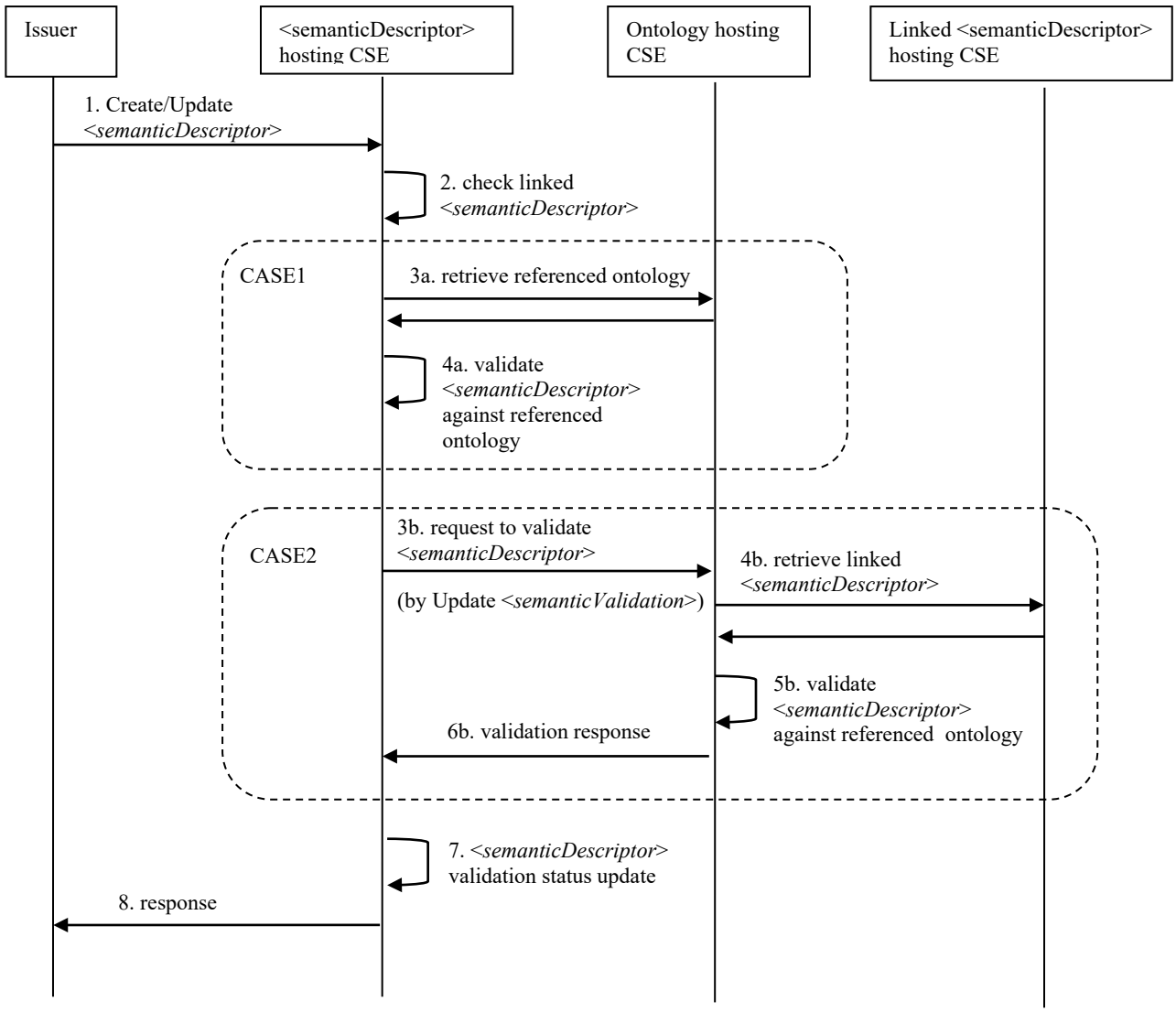


**Figure 7.10.3-1: Message flow for semantic description validation triggered by <*semanticDescriptor*> resource Create/Update**

-

**Step 1.** The issuer shall send a Create or Update request to the hosting CSE of a <*semanticDescriptor*> resource (called <*semanticDescriptor*> hosting CSE). The request shall contain the <*semanticDescriptor*> resource representation, which includes a *validationEnable* attribute (set to 'true') to trigger the semantic validation process, the semantic description (*descriptor* attribute), the URI of the referenced ontology (*ontologyRef* attribute) against which to validate, and potentially URIs (*relatedSemantics* attribute, or triples with annotation property *m2m:resourceDescriptorLink* in the *descriptor* attribute) to other linked <*semanticDescriptor*> resources that are also incorporated for validation.

**Step 2.** After receiving the request, the <*semanticDescriptor*> hosting CSE shall firstly check if semantic validation is needed according to the value of the *validationEnable* attribute. If true, it shall further check if the addressed <*semanticDescriptor*> resource is linked to any other remote <*semanticDescriptor*> resources according to the URIs in the *relatedSemantics* attribute or triples with annotation property *m2m:resourceDescriptorLink* in *descriptor* attribute. If no, the procedure goes to **Case 1** (**Step 3a** to **4a**), otherwise, goes to **Case 2** (**Step 3b** to **6b**).

NOTE: The <*semanticDescriptor*> hosting CSE may override the value of the *validationEnable* attribute according to its local policy so as to enforce or disable the following semantic validation procedures regardless of the requested value from the issuer.

**Case 1: stand-alone *<semanticDescriptor>***

**Step 3a.** The <*semanticDescriptor*> hosting CSE shall retrieve the referenced ontology representation according to the URI in the *ontologyRef* attribute of the addressed <*semanticDescriptor*> resource from the ontology hosting CSE (which hosts the referenced ontology). In case the ontology representation cannot be retrieved (due to access right control or other exceptional reasons), skip **Step 4a.**

**Step 4a.** The <*semanticDescriptor*> hosting CSE shall use the retrieved referenced ontology to validation the semantic description (the triples in *descriptor* attribute) of the addressed <*semanticDescriptor*> resource. The aspects to be checked in semantic validation is specified in clause 7.10.4.

**Case 2: linked *<semanticDescriptor>***

**Step 3b.** This step shall follow **Step 1** of figure 7.10.2-1, wherein the <*semanticDescriptor*> hosting CSE shall act as the Issuer and the <*semanticDescriptor*> resource to be validated is the addressed <*semanticDescriptor*> resource in the received Create or Update request.

**Step 4b.** This step shall follow **Step 2** of figure 7.10.2-1.

**Step 5b.** This step shall follow **Step 3** of figure 7.10.2-1.

**Step 6b.** This step shall follow **Step 4** of figure 7.10.2-1, wherein the response is sent to the <*semanticDescriptor*> hosting CSE.

**Step 7.** The <*semanticDescriptor*> hosting CSE shall perform the normal operation (Create or Update) on the addressed <*semanticDescriptor*> resource according to the original request from the issuer. In addition, based on the validation result of **Step 4a** (in **Case 1**) or the validation response received in **Step 6b** (in **Case 2**), The <*semanticDescriptor*> hosting CSE shall update the *semanticValidated* attribute properly to reflect the validation status (validated or not) of the addressed <*semanticDescriptor*> resource accordingly. If **Step 4a** is skipped due to **Step 3a** fails, it's also considered as not validated.

**Step 8.** The <*semanticDescriptor*> hosting CSE shall return the operation (Create or Update) response to the issuer.

## 7.10.4 Aspects to be checked in semantic validation

Several aspects shall to be checked in order to make sure that the content of *descriptor* attribute of <*semanticDescriptor*> resource consists of valid RDF triples and they are indeed capable of interoperating semantically with other oneM2M resources. Taking into account the nature of semantically annotated data, three levels of validation can be distinguished:

1) **Lexical check.** This level of check consists of verifying the correctness of RDF serialization regarding to the declared type. For example, the <*semanticDescriptor*> resource is marked in XML representation (according to the *descriptorRepresentation* attribute) whereas the semantic annotation (in the *descriptor* attribute) is indeed serialized in JSON, or the XML document contains some error that causes parse error, the lexical check fails.

-

2) **Syntactic checks.** After the basic lexical checks, the syntactic check consists of verifying the correctness of the "syntax" of the RDF triples represented by the underlined serialization format, more specifically:

   a) **Untyped of resources and literals**. Here resource refers to instances of a class, and literal refers to a textual or numerical value. The type of resource or literal is the link of an annotation back to the ontology which enables the semantic capabilities. Any un-typed element presented in an annotation is problematic towards the semantic interoperability.

   b) **Ill-formed URIs**. URI is essential and critical for identification of a resource. They shall be checked against RFC3968 which defines the generic syntax of URI.

   c) **Problematic prefix and namespaces**. Namespaces play the role of linking the annotation to the reference ontologies and vocabularies, and it shall be consistent with *ontologyRef* attribute. If the URI of the namespace is problematic (e.g. wrong URI, URI contains illegal character), it may cause others to mis-interpret the data semantics and types. Prefix is a unique reference to replace the namespaces in the local file. A one-to-one mapping between the prefix and namespace is essential and shall be checked to ensure a correct reference.

   d) **Unknown classes and properties**. A prerequisite of semantic interoperability is that all the resources use a common and agreed vocabulary. As consequence, if any resource uses in its annotation a class or property that is not defined in the reference ontology(ies), other resources would have no way to understand it, so that the semantic interoperability is impossible.

3) **Semantic checks.** Following a successful syntactic validation, the semantic check consists of verifying the logical consistence of the semantic annotation regarding to the reference ontology(ies):

   a) **Cardinality inconsistency:**

      i) **Inconsistency of object properties**. If the ontology defines that class A has an object property that can have one and only one instance of class B, and in the annotation, there are two instances of B related to one instance of A, there is a problem.

      ii) **Inconsistency of data properties.** If the ontology defines that class A has a data property that can have one and only one data value, and in the annotation, there are two instances of the data properties of different value, there is a problem.

   b) **Problematic relationship or inheritance**. Following the relationship defined in the reference ontology, if an instance of a class A is wrongly annotated to be at same time an instance of class B which is disjoint from class A, there is a conflict and the instance cannot be resolved by the semantic engine. A concrete example in detailed in clause 8.3.1 in oneM2M TR-0033 [i.3].

   c) **Remaining dependencies**. If deleting a property of an instance of a class for which this property is mandatory, there is a problem.

The validation response returned to the issuer depends on the result of each of the above tests. To conclude that an annotation is validated, a complete check of all the above checks shall to be performed and passed. However, as several tests are independent from others (for example, 3.a and 3.b do not have an impact on each other), several "validated profiles" may be defined as a subset of all the aspects to be checked.

## 7.11 Semantics Reasoning

Semantic reasoning is a mechanism to derive implicit facts that are not explicitly expressed in the existing knowledge/facts (such as RDF triples) by leveraging a set of reasoning rules. A Semantic Reasoning Function (SRF) is defined in this clause in order to support semantic reasoning functionality in the oneM2M system. The key features of a SRF are shown in Figure 7.11-1:

-

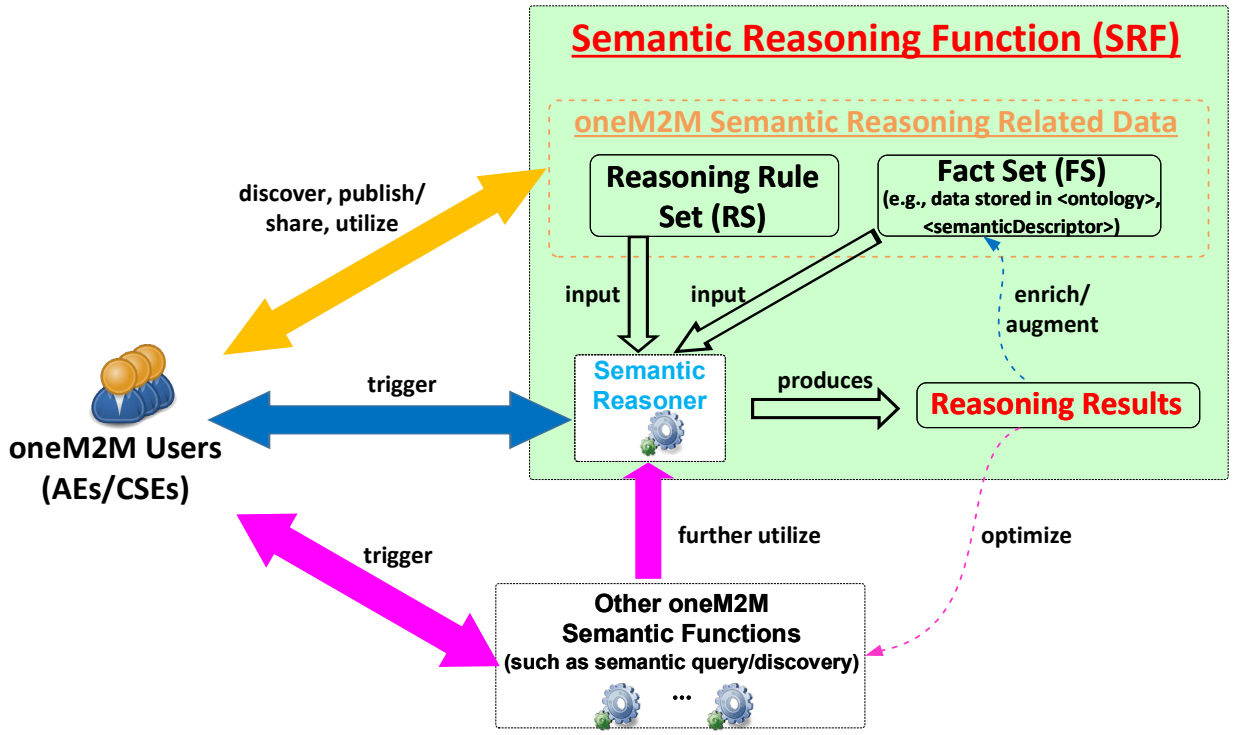


**Figure 7.11-1: Key Features of Semantic Reasoning Function (SRF)**

**Feature-1: Enabling semantic reasoning related data**

The major functionality of Feature-1 is to enable the semantic reasoning related data (referring to facts and reasoning rules) by be discoverable and publishable/sharable across different entities in the oneM2M system (which is illustrated as the dark yellow arrow in the Figure 7.11-1). The semantic reasoning related data can be a Fact Set (FS) and/or a Rule Set (RS):

- A FS refers to a set of facts. For example, a set of RDF triples stored in a <semanticDescriptor> resource can be regarded as a FS. In general, a FS can be used as an input for a semantic reasoning process (i.e. an input FS) or it can be a set of inferred facts as the result of a semantic reasoning process (i.e. an inferred FS).
- A RS refers to a set of semantic reasoning rules. For example, oneM2M applications may define their own reasoning rules (user-defined reasoning rules) for different application needs.

Overall, Feature-1 involves the publishing/discovering/sharing of semantic reasoning related data (including both FSs and RSs) through appropriate oneM2M resources. The general flow of Feature-1 is that oneM2M users (as originators) can send requests to certain receiver CSEs in order to publish/discover/update/delete the FS/RS-related resources through the corresponding CRUD operations. Once the processing is done, the receiver CSE will send the response back to the originator.

**Feature-2: Optimizing other semantic operations with background semantic reasoning support**

The existing semantic operations supported in the oneM2M system (e.g., semantic resource discovery and semantic query) may not yield desired results without semantic reasoning support. The major functionality of Feature-2 of SRF is to leverage semantic reasoning as a "background support" function to optimize other semantic operations (which are illustrated by the pink arrows in the Figure 7.11-1). In this case, users trigger/initiate specific semantic operations (e.g., a semantic query). During the processing of this operation, semantic reasoning may be further triggered in the background, which is however fully transparent to the user.

Overall, the general flow of Feature-2 is that oneM2M users (as originators) can send requests to certain receiver CSEs for the desired semantic operations (such as semantic resource discovery, semantic query, etc.). During the request processing, the receiver CSE, assuming it supports SRF, can further leverage the reasoning capability. In general, the reasoning capability of the SRF is realized by an underlying semantic reasoner. By leveraging the outputs of semantic reasoning (i.e., reasoning result), the receiver CSE will further produce the optimal result for the semantic operation as

-

requested by the originator (e.g., the semantic query result, or semantic discovery result) and then send the response back to the originator.

**Feature-3: Enabling individual semantic reasoning process**

oneM2M users (as originators) may also directly interact with the SRF by triggering an individual semantic reasoning process, which is Feature-3 of the SRF. When using this feature, a oneM2M user shall first identify the interested facts (as input FS) as well as the desired reasoning rules based on their application needs. When the input FS and RS are identified, the oneM2M user shall send a request to the SRF for triggering a specific semantic reasoning process by specifying the inputs (i.e. the input FS and RS). The SRF will then initiate a desired semantic reasoning process. Once the SRF works out the semantic reasoning result, it will be returned to the oneM2M users for further usage.

# 7.12 Ontology Mapping

## 7.12.1 Introduction

There are already many standardized or proprietary ontologies defined for various vertical domains or cross-domain scenarios. Each ontology specifies the common vocabulary and relationships between concepts within its own namespace, but may sometimes overlap conceptually with other ontologies due to the independent design. This is often true if two ontologies are designed for the same knowledge domain or under a common high level domain. Different terminologies may mean the same or similar concept (e.g. lamp vs. light), or one is the actually the sub-class of another (e.g. device vs. thing).

To enable the semantic interoperability between different ontologies, ontology mapping is a prerequisite. It's an important ontology management method to identify the commonality, similarity as well as inclusion relationships between ontologies, so that the data described in one ontology can be consumed meaningfully by the application who understand only another ontology. Ontology mapping can also help to build a global knowledge base and enhance the system intelligence by linking together a collection of ontologies via the anchors of equal/similar/inclusive concepts.

Ontology mapping can be implemented by either manual approaches or automatic approaches. For example, in Annex B.1 of oneM2M TS-0012 [5], the ontology mapping between Base Ontology and SAREF is specified by manually configured mapping rules (in the format of mapping tables) according to the experts' common understanding on both ontologies.

However, discovering proper mapping relationships manually is often too labour-intensive, error-prone, and impractical for large ontologies, especially for non-standardized and unstable ones. Therefore, oneM2M provides automatic means of ontology mapping to discover, create and save the mapped relationships between semantically related ontologies by using industry-proven mapping algorithms, e.g. the edit distance, language-based similarity, structural-based similarity, or external- resources-based similarity etc.

The solution is based on the <*ontologyMapping*> resource to configure the input parameters for executing the ontology mapping task and to store the mapping result. Meanwhile, the <*ontologyMappingAlgorithmRepository*> resource and its child resources <*ontologyMappingAlgorithm*> are used to host a collection of algorithms for automatic ontology mapping that can be selected for individual ontology mapping tasks. The detailed procedures related to ontology mapping are specified in clause 6.10, 6.11 and 6.12.

Based on the generated ontology mapping result contained in the <ontologyMapping> resource, semantically equivalent operations such as semantic resource discovery and semantic query can be realized in multi-ontology scenarios as specified in clause 7.12.2.

## 7.12.2 Semantic query and semantic resource discovery based on the ontology mapping result

### 7.12.2.1 Introduction

Semantic query and semantic resource discovery operations can be enhanced by leveraging the ontology mapping result stored in the <*ontologyMapping*> resource, so that an application which understands only one ontology (e.g. Ontology-A) can get the resulting content or resource described in another (e.g. Ontology-B).

-

This feature requires the issuer to provide an additional request parameter - ***Ontology Mapping Resources*** as specified in [1] in a normal semantic query and semantic resource discovery request. This request parameter contains a list of resource identifiers of existing <*ontologyMapping*> resources that are used as the base of converting the query statement or the semantic description of the target resources into their equivalents.

The following clause describes the detailed procedures using the example ontology mapping result in clause 6.10.2.

Note: There is no difference between the semantic query operation and the semantic resource discovery operation in terms of applying the feature of ontology mapping in addition to the respective original procedures. So the following procedures combine the two operations in to one message flow.

### 7.12.2.2 Procedures

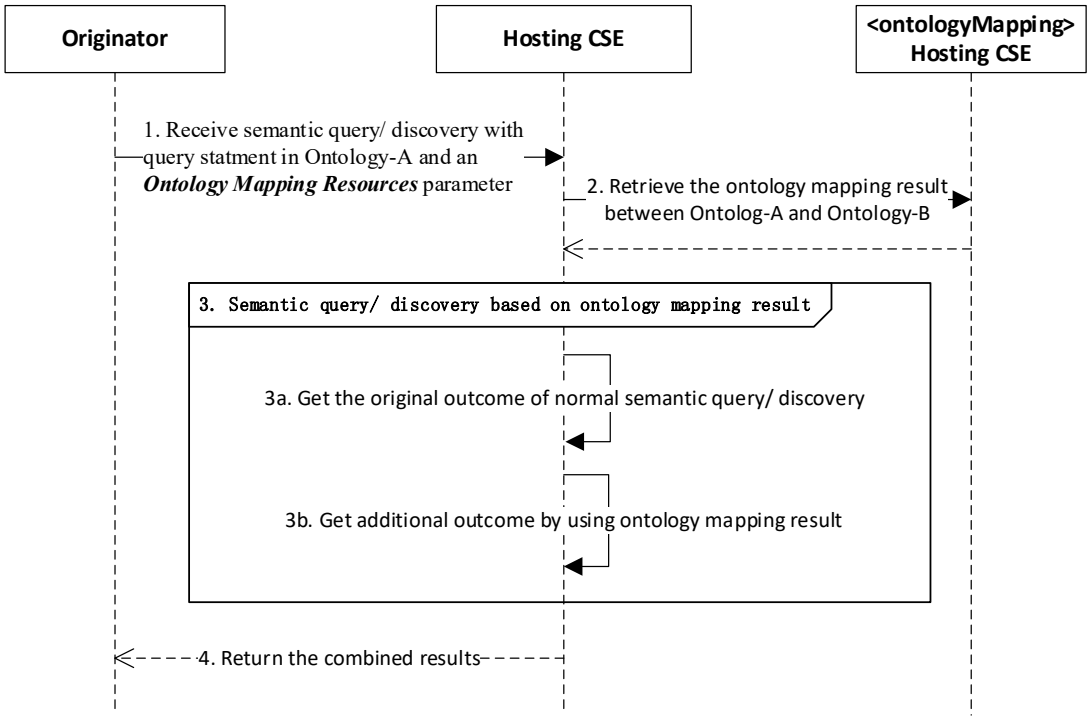


**Figure 7.12.2.2-1: The semantic query (or semantic resource discovery) procedure with ontology mapping**

The detailed message flow is depicted in Figure 7.12.2.2-1 and explained as follows:

1. The hosting CSE (e.g. an oneM2M platform) receives a semantic query (or semantic resource discovery) request from an Originator (e.g. an oneM2M application). The request shall carry a ***semanticsFilter*** request parameter that contains the original query statement described in the first ontology (e.g. Ontology-A), as well as an ***Ontology Mapping Resources*** request parameter that contains the resource identifiers of one or multiple <*ontologyMapping*> resources.
   Note that the originator may or may not be the one who created the <*ontologyMapping*> resources. The use of the <*ontologyMapping*> resources is subject to the associated access control policy against the originator.

   For example, the original query statement may be:

```
SELECT ?device
WHERE {
?device rdf:type Ontology-A:LightSensor.
}
```

2. The hosting CSE shall locate the <*ontologyMapping*> resources according to the resource identifiers in the ***Ontology Mapping Resources*** request parameter, and retrieves the mapping results between the first ontology (Ontology-A) and the second ontology (Ontology-B) from the *mappingResult* attribute of the <*ontologyMapping*> resources.

   For example, the *mappingResult* may contain the following triple (mapping relationship):

-

```
Ontology-A:LightSensor  owl:equivalentClass Ontology-B:Light_Sensor
```

3. The hosting CSE shall perform the semantic query (or semantic resource discovery) upon the semantic description of the target resources which reference the first ontology (Ontology-A) or the second ontology (Ontology-B), according to the ontology mapping result and the original query statement. The result shall contain both the original outcome of a normal semantic query (or semantic resource discovery) operation as well as additional outcome of the semantic query (or semantic resource discovery) by using the ontology mapping result.

   This step comprises the following sub-steps:

   a) The hosting CSE shall perform the semantic query (or semantic resource discovery) procedure upon the semantic description of the target resources that reference the first ontology (Ontology-A) using the original query statement, and collect the outcome.

      For example, a <container-x> resource may have a <*semanticDescriptor*> child resource that references Ontology-A (by the *ontologyRef* attribute) and contains the following triple that matches the original query statement:

      ```
      dev-x rdf:type Ontology-A:LightSensor.
      ```

   b) The hosting CSE shall perform the semantic query (or semantic resource discovery) procedure upon the semantic description of the target resources that reference the second ontology (Ontology-B) using the ontology mapping result (provided in the <ontologyMapping> resources), and get the additional outcome.

      This step may be performed in two different approaches that are implementation specific (non-normative):

      - Approach-1: converting the semantic query statement using the ontology mapping result.

        The hosting CSE can determine the equivalent query statement described in the second ontology (Ontology-B) by converting from the original query statement described in the first ontology (Ontology-A), according to the ontology mapping result between the first ontology (Ontology-A) and the second ontology (Ontology-B). This can be done by replacing all the ontology class and properties of the first ontology (Ontology-A) in the query statement with the corresponding equivalents of the second ontology (Ontology-B).

        According to the examples above, the equivalent query statement becomes:

        ```
        SELECT ?device
        WHERE {
        ?device rdf:type Ontology-B:Light_Sensor.
        }
        ```

        The hosting CSE then performs the semantic query (or semantic resource discovery) procedure upon the semantic description of the target resources that reference the second ontology (Ontology-B) using the equivalent query statement, and collect the second query (or semantic resource discovery) result.

        For example, a <container-y> resource may have a <*semanticDescriptor*> child resource that references Ontology-B (by the *ontologyRef* attribute) and contains the following triple that matches the converted equivalent query statement:

        ```
        dev-y rdf:type Ontology-B:Light_Sensor.
        ```

      - Approach-2: converting the semantic description of the target resources according to the <ontologyMapping> resources.

        The hosting CSE can determine the equivalent semantic description in the first ontology (Ontology-A) for the target resources which reference the second ontology (Ontology-B), by converting from the original semantic description in the second ontology (Ontology-B) according to the ontology mapping result. This can be done by replacing all the ontology class and properties of the second ontology (Ontology-B) in the related <*semanticDescriptor*> resources of the target resources into the equivalents of the first ontology (Ontology-A).

-

For example, a <container-y> resource may have a <*semanticDescriptor*> child resource that references Ontology-B (by the ontologyRef attribute) and contains the following triple:

```
dev-y rdf:type Ontology-B:Light_Sensor.
```

This triple is then converted into the equivalent triple in Ontology-A that matches the original query statement:

```
dev-y rdf:type Ontology-A:LightSensor.
```

The hosting CSE then performs the semantic query (or semantic resource discovery) procedure upon the converted equivalent semantic description using the original query statement, and collect the second query (or semantic resource discovery) result.

4. The hosting CSE shall combine the query (or semantic resource discovery) results from both step 3a and step 3b, and return the combined results to the originator.

   For example, in the case of semantic query, the results are the IRIs of `dev-x` and `dev-y`. In the case of semantic resource discovery, the results are the resource identifiers of <container-x> and <container-y>.

-

# History

This clause shall be the last one in the document and list the main phases (all additional information will be removed at the publication stage).

| Publication history | | |
|---|---|---|
| V4.0.0 | June 2019 | Release-4 Development<br>Full copied from and with the same content as TS-0034 V3.0.2 |
| V4.1.0 | October 2019 | Incorporated the following two contributions towards TS-0034 as agreed in oneM2M TP42:<br>• SDS-2019-0461R02<br>• SDS-2019-0484R02 |
| V4.1.1 | Jan. 2020 | Incorporated the following one contribution towards TS-0034 as agreed in oneM2M TP43:<br>• SDS-2019-0673 |