ETSI GR CIM 022 V1.2.1 (2024-01)

Context Information Management (CIM); NGSI-LD/oneM2M interworking proxy proposal

Reference

RGR/CIM-0022v121

Keywords

interworking, IoT, NGSI-LD, oneM2M

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - APE 7112B
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° w061004871

Content

Intellectual Property Rights

Foreword

This Group Report (GR) has been produced by ETSI Industry Specification Group (ISG) cross-cutting Context Information Management (CIM).

Modal verbs terminology

In the present document "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

Executive summary

The present document presents several NGSI-LD/oneM2M interworking proxy solutions using existing oneM2M and NGSI-LD features and analyses those to find which one is suitable for a certain condition.

Introduction

The present document aims to find candidate interworking solutions between oneM2M and NGSI-LD technologies.

oneM2M is one of the most widely used middleware platform standard for IoT/M2M devices/gateways and servers. oneM2M, as the name stands for, is the standards for communicating IoT data among IoT things and IoT applications so oneM2M provides IoT data and things related functions such as collecting IoT data, to actuating IoT things, managing IoT things. NGSI-LD is the standard focusing on exchanging data from data providers and consumers. Therefore, in NGSI-LD standpoint, what kind of data to be exchanged is out of scope and it does not provide any IoT specific functions.

As stated above, two technologies have its own speciality so the present document investigates the complementary use of the two technologies as follows:

1) NGSI-LD context broker can make use of oneM2M for actuating IoT things.

2) NGSI-LD context broker can make use of oneM2M for collecting IoT data.

3) oneM2M can make use of NGSI-LD for collecting data from external systems so oneM2M applications can access that with oneM2M protocols.

4) oneM2M can make use of NGSI-LD for collecting linked data which can be leveraged with its semantic features.

In the end of 2022, there are 4 versions of oneM2M standards and compliant products in the deployments. In the case of NGSI-LD, there are release 1 compatible deployments. One important requirement for the candidate solutions in the present document is the version compatibility of any of the oneM2M and NGSI-LD standards. Therefore, the solutions do not propose any changes of features or APIs of both NGSI-LD and oneM2M specifications and use as-is features.

The scope of the present document is limited to data synchronization between oneM2M and NGSI-LD. Other interworking for example querying information of NGSI-LD context broker from oneM2M system or querying information of oneM2M system from NGSI-LD context broker is out-of-scope of the present document.

To be able to synchronize new data between NGSI-LD and oneM2M, the interworking solutions relies on the oneM2M Subscription/Notification and NGSI-LD Subscription/Notification as well as retrieval of oneM2M resources and Temporal API of NGSI-LD Entities.

1 Scope

The present document provides several NGSI-LD/oneM2M interworking proxy solutions using existing oneM2M features and analyses those to find which one is suitable for a certain condition.

2 References

2.1 Normative references

Normative references are not applicable in the present document.

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.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

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

[i.1] ETSI TS 118 101: "oneM2M; Functional Architecture (oneM2M TS-0001)".

[i.2] ETSI GS CIM 009 (V1.6.1): "cross-cutting Context Information Management (CIM); NGSI-LD API".

NOTE: The interworking scheme uses the NGSI-LD ontology (e.g. ngsi-ld:Entity) in ETSI GS CIM 009.

[i.3] FED4IoT project.

[i.4] ETSI TS 118 123 (V3.7.3): "oneM2M; Home Appliances Information Model and Mapping (oneM2M TS-0023 version 3.7.3 Release 3)".

[i.5] ETSI TS 118 104: "oneM2M; Service Layer Core Protocol Specification (oneM2M TS-0004)".

[i.6] oneM2M TR-0007 (V2.11.1): "Study of Abstraction and Semantics Enablements".

[i.7] Smart Data Models About Web Page.

[i.8] JSONPath: Query expressions for JSON.

3 Definition of terms, symbols and abbreviations

3.1 Terms

Void.

3.2 Symbols

Void

3.3 Abbreviations

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

AE	Application Entity
API	Application Programming Interface
ASN	Application Service Node
CSE	Common Services Entity
DAS	Dynamic Authorization System
ID	Identifier
IN	Infrastructure Node
IoT	Internet Of Things
IPE	Interworking Proxy Entity
JSON	JavaScript Object Notation
JWT	JSON Web Token
M2M	Machine to Machine
MN	Middle Node
NGSI-LD	Next Generation Service Interface Linked Data
OWL	Web Ontology Language
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
SAREF	Smart Applications REFerence ontology
SDT	Smart Device Template
SPARQL	SPARQL Protocol And RDF Query Language
TS	Technical Specifications
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
XSD	XML Schema Definition

4 Architecture Model

4.1 Information Model

The notion of data synchronization between oneM2M and NGSI-LD systems can be realized by data mapping as well as interface interworking. The two standards have a different view on information and data modelling and it is important to understand this difference.

oneM2M takes two approaches regarding information models. One is the modelling and corresponding models' standardization with Smart Device Template (SDT) (ETSI TS 118 123 [i.4]). Figure 4.1-1 shows the hierarchy of SDT in oneM2M standard. It is a template to represent device and functional modules. It is a good fit for oneM2M as a model for M2M/IoT device management and data collection from devices. On the other hand, any service-specific or user-defined data can be exchanged by oneM2M platforms with <*container*> and <*contentInstance*> resources, for example. It allows flexibility to use any preferred data models for IoT services, but the downside of this is lack of data interoperability to 3rd party application providers, because there is no common, cross-domain data model.

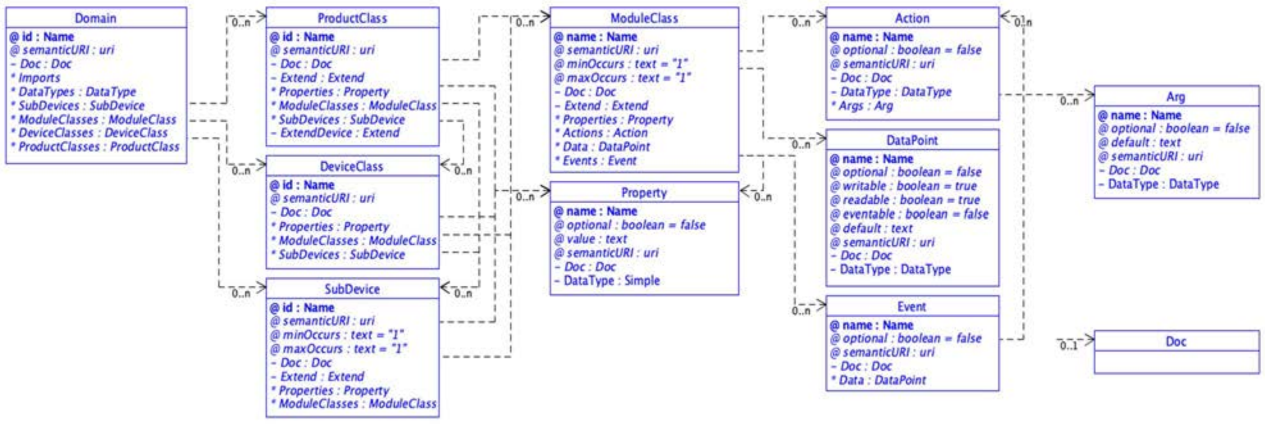


Figure 4.1-1: oneM2M Smart Device Template (SDT) [i.4]

In NGSI-LD systems, it is up to service or data providers to define their domain-specific information models. Indeed, NGSI-LD standard does not define any domain-specific information models but, instead, defines a common, cross-domain data model based on the Entity, Property and Relationship concepts that form a Property Graph. Even though there is no schemas for service implementations, by means of the Linked Data concept, each NGSI-LD Entity instance and its attributes (Properties and Relationships) can be identified uniquely, so that 3rd party application providers, willing to conform to the underlying cross-domain model, can use data from other data providers.

On top of the cross-domain model, NGSI-LD advocates for higher level models that are domain-specific, to be constructed and adopted on top of the cross-domain one, to enable full interoperability (oneM2M TR-0007 [i.6]).

4.2 Reference Model

To be able to interwork between two technologies, the application called Interworking Proxy Entity (IPE) which understands the technologies need existing.

IPE on oneM2M side, acts as Application Entity (AE) and interacts with a Common Services Entity (CSE). The CSE can be Infrastructure Node (IN), Middle Node (MN), or Application Service Node (ASN). Even though IPE interact with the CSE, depending on address of oneM2M resource, IPE can get or manipulate resources in other CSEs via Mcc or Mcc'.

IPE on NGSI-LD side, acts as NGSI-LD application and interacts with a NGSI-LD context broker. Even though IPE interacts with the NGSI-LD context broker, IPE can get or manipulate entities in other NGSI-LD context brokers using CSourceRegistration information.

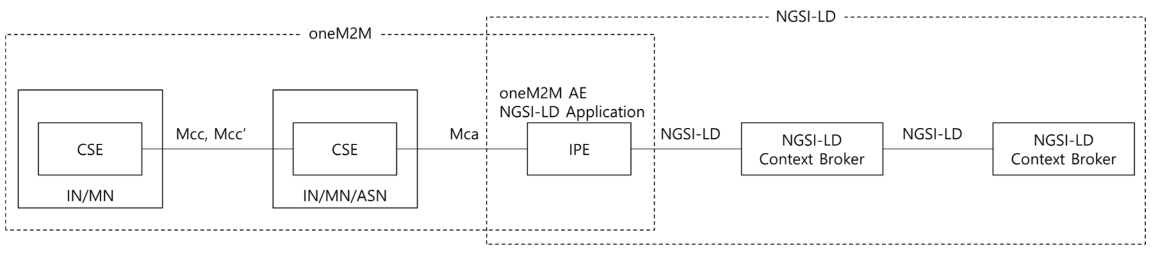


Figure 4.2-1: NGSI-LD - oneM2M Interworking Reference Model

4.3 Types of Interworking Proxy Entity

Depending on the use case, oneM2M data needs synchronizing to NGSI-LD context broker or NGSI-LD data needs synchronizing to oneM2M. Therefore, in the present document, each solution specifies direction of data synchronization:

- oneM2M to NGSI-LD: This can be used when services are already implemented and data are already being stored in oneM2M platform. The goal is for users to get data of oneM2M services using the NGSI-LD API.

- NGSI-LD to oneM2M: This can be used when device management or actuation is required from NGSI-LD enabled system (e.g. city operation centre) as well as when oneM2M applications make use of NGSI-LD data.

5 Solution 1 - labels based dynamic interworking

5.1 Introduction

Interworking between oneM2M and NGSI-LD systems requires interfaces and data mapping between the two. It is realized by the Interworking Proxy Entity (IPE) which understand two standard interfaces and application logics including data models.

The proposed interworking solution retrieves information on the interworking via labels attribute, which is the oneM2M resource universal attribute, because all oneM2M resource can have labels. This means that any of the oneM2M resource in any resource type, can be mapped with NGSI-LD Entity instances. Since NGSI-LD Entity represents a logical entity, while oneM2M resource can represent a measurement of a device, which is a small portion of a device model, NGSI-LD Entity can be mapped with one or more oneM2M resource instances.

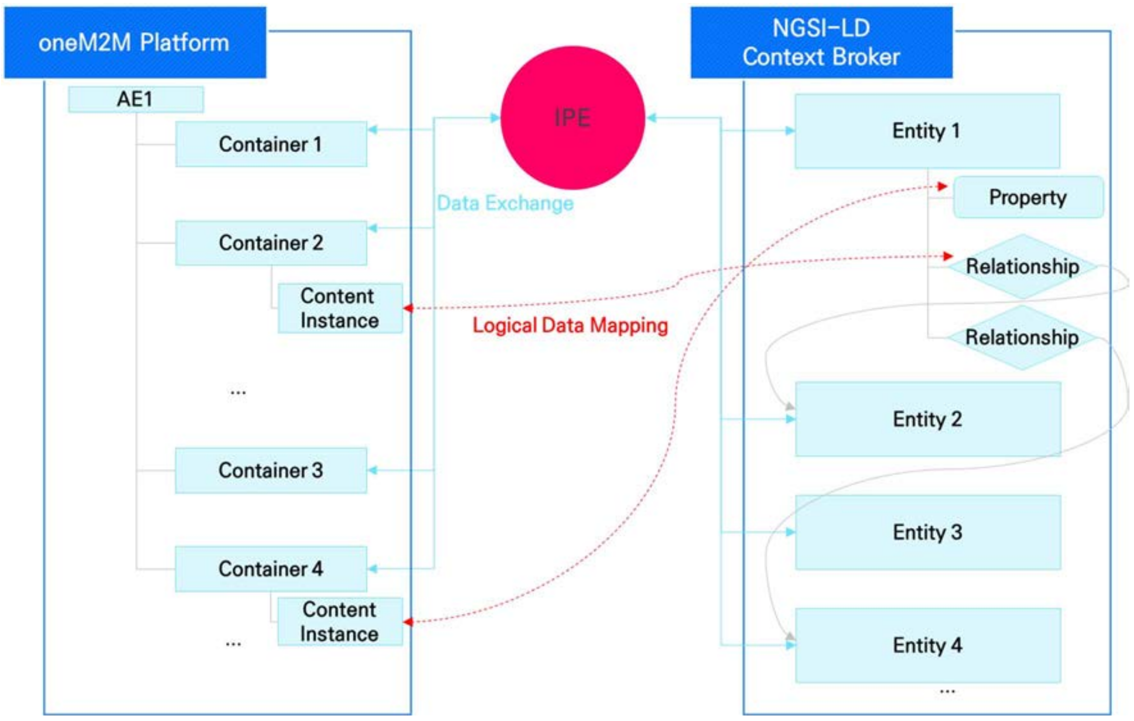


Figure 5.1-1: Example of Resource mapping between oneM2M and NGSI-LD systems

The given examples in the clause 5.1 only use <*container*> resource type from the oneM2M standard, but <*flexContainer*> or <*timeSeries*> resource types can also be applied for this label-based interworking mechanism.

5.2 oneM2M features

This solution uses labels attribute, one of the common attributes in all resource types. Normally in oneM2M labels attribute is used for discovery purposes like hash tag, but this solution uses for storing mapping rules.

Labels attribute is a list of individual labels and each of them is either a standalone label-key, used as a simple "tag", that can be used for example for discovery purposes when looking for particular resources that one can "tag" using that label-key or a composite element made of a label-key and a label-value, separated by a special character defined in ETSI TS 118 101 [i.1].

5.3 Resource mapping

In this interworking scheme, the labels attribute contains mapping information between oneM2M resources and NGSI-LD Entities. The labels attribute has string array data type in ETSI TS 118 104 [i.5], so it is proposed to use key-value pair in a single token string and there are one or more pairs in the labels attribute on the oneM2M resource instance. The keys that represent interworking information are defined in Table 5.3-1.

Table 5.3-1: Interworking mapping labels keys

Label Key Name	Data Type	Limitation	Multiplicity	Description
Iwked-Technology	String	"NGSI-LD"	1	"NGSI-LD"
Iwked-Entity-Type	String	NGSI-LD Entity Type	0..1	Type of the NGSI-LD Entity
Iwked-mapping-rule	List of MappingRule	Mapping Rule	0..1	Data mapping rules for oneM2M resource and NGSI-LD Entities
Iwked-Entity-ID	String	n/a	0..1	The id attribute of NGSI-LD Entity instance. If not exists, it is decided by "oneM2M platform ID" + ":" + "ri of oneM2M resource"
Iwked-Related-Resources	List of String	n/a	0..1	Other oneM2M resources instances to be combined for the NGSI-LD Entity instance
NGSI-LD-Context	List of String	n/a	1	@context attribute of NGSI-LD Entity instance

Table 5.3-2 defines the Mapping Rule data type that is used by Iwked-mapping-rule.

Table 5.3-2: Data type of Mapping Rule

Name	Data Type	Limitation	Multiplicity	Description
oneM2MResourceId	String	n/a	0..1	Identifier of a oneM2M resource instance
oneM2MChildResourceType	Integer	n/a	0..1	Applied child resource type of the resources to be interwoked. m2m:resourceType is used [i.5]
oneM2MAttribute Name	String	n/a	1	Attribute name of the instance
oneM2MJsonPath	String	n/a	0..1	JSONPath [i.8], this is valid when the resource is serialized in JSON
oneM2MDataType	String	n/a	0..1	Data type for the attribute or the element referred by JSONPath. Allowed values are: "String", "Integer", "Double", "Boolean", "Date", "ArrayString", "ArrayInteger", "ArrayDouble", or "ArrayBoolean"
ngsi-ldQueryTermAttributePath	String	n/a	0..1	Attribute or element of Attribute referred by QueryTerm in Query Language [i.2]
ngsi-ldAttributeType	List of String	"Property" or "Relationship"	1	Type of NGSI-LD attribute
ngsi-ldAttribute DataType	List of String	n/a	1	Data type of ngsi-ldQueryTermAttributePath. Allowed values are: "String", "Integer", "Double", "Boolean", "Date", "ArrayString", "ArrayInteger", "ArrayDouble", or "ArrayBoolean"
ngsi-ldAttribute ParentInformation	Array of ParentInformation	n/a	0..1	Since there is no schema information of the NGSI-LD Entity, information on parent attributes is required

Name	Data Type	Limitation	Multiplicity	Description
oneM2MArrayIndex	Integer	n/a	0..1	Indicates an index of an array item. This is valid when oneM2MDataType is in array
ngsi-ldArrayIndex	Integer	n/a	0..1	Indicates an index of an array item. This is valid when Ngsi-ldAttributeDataType is in array

Table 5.3-3 defines the ParentInformation data type introduced in Table 5.3-2.

Table 5.3-3: Data type of ParentInformation

Name	Data Type	Limitation	Multiplicity	Description
ngsi-ldQueryTermAttributePath	String	n/a	1	Attribute of QueryTerm in Query Language
ngsi-ldAttributeType	String	"Property" or "Relationship"	1	Type of NGSI-LD attribute

5.4 New data synchronization procedures

5.4.1 Pre-configuration

An NGSI-LD system user updates the labels attribute with mapping information as specified in clause 5.3. It is iterated for all the oneM2M resources for NGSI-LD Entity mapping.

5.4.2 IPE initialization

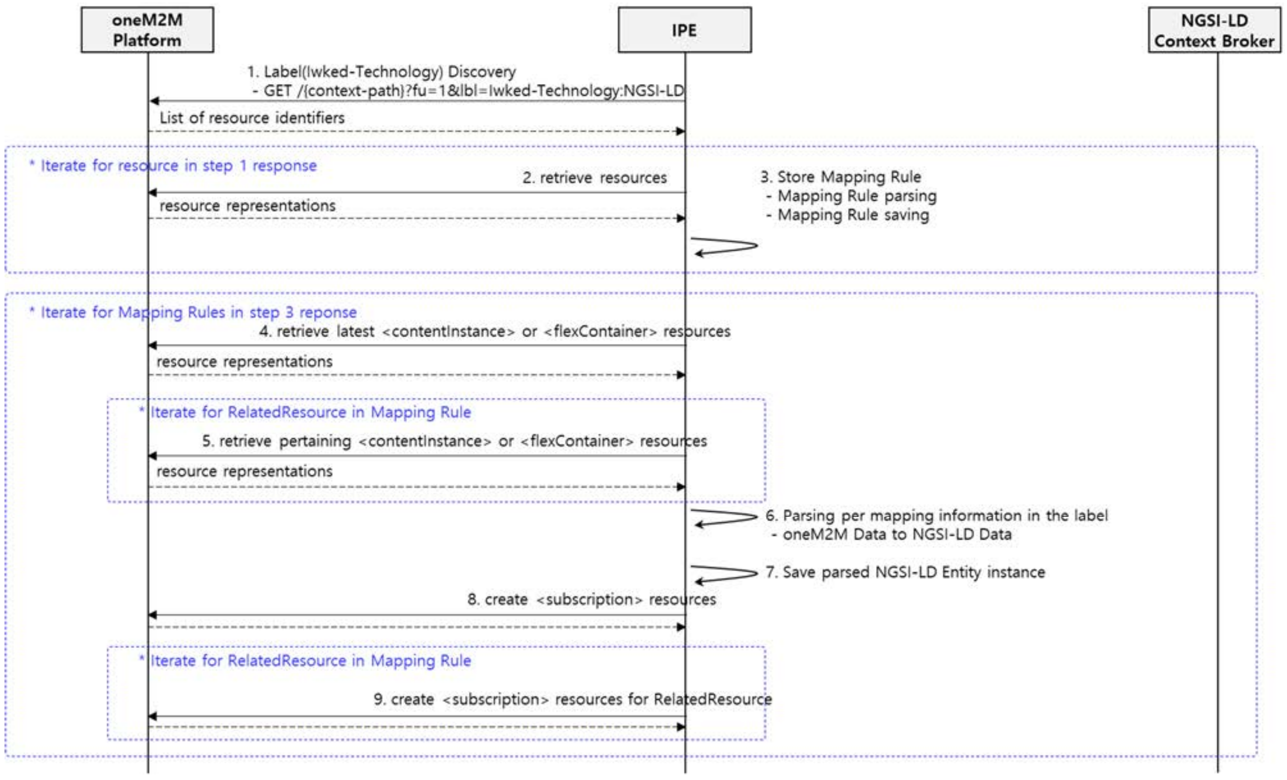

oneM2M Platform
IPE
NGSI-LD Context Broker
1. Label(Iwked-Technology) Discovery
- GET /{context-path}?fu=1&lbl=Iwked-Technology:NGSI-LD
List of resource identifiers
* Iterate for resource in step 1 response
2. retrieve resources
resource representations
3. Store Mapping Rule
- Mapping Rule parsing
- Mapping Rule saving
* Iterate for Mapping Rules in step 3 reponse
4. retrieve latest <contentInstance> or <flexContainer> resources
resource representations
* Iterate for RelatedResource in Mapping Rule
5. retrieve pertaining <contentInstance> or <flexContainer> resources
resource representations
6. Parsing per mapping information in the label
- oneM2M Data to NGSI-LD Data
7. Save parsed NGSI-LD Entity instance
8. create <subscription> resources
* Iterate for RelatedResource in Mapping Rule
9. create <subscription> resources for RelatedResource

Figure 5.4.2-1: Procedures for IPE initialization

Step 1. IPE discovers <*container*> and <*flexContainer*> resource identifiers which contains the interworking information in the labels attribute.

Step 2. Retrieve the resources returned in step 1. and check the labels attribute.

Step 3. Parse the mapping rules in the labels attribute for oneM2M - NGSI-LD interworking and save it.

Steps 4. to 5. Per mapping rules in step 3., retrieve <*latest*> resource, which is the lastly created <*contentInstance*> resource, in case of <*container*> resource. If it is <*flexContainer*> resource, then retrieve the <*flexContainer*> resource. If the mapping rule contains relatedResource, then corresponding resources are also retrieved.

Steps 6. to 7. Convert the resource instances from step 4. into NGSI-LD Entity instances and store them.

Steps 8. to 9. Create <*subscription*> resources for the oneM2M resources in step 5.

5.4.3 Resource synchronization

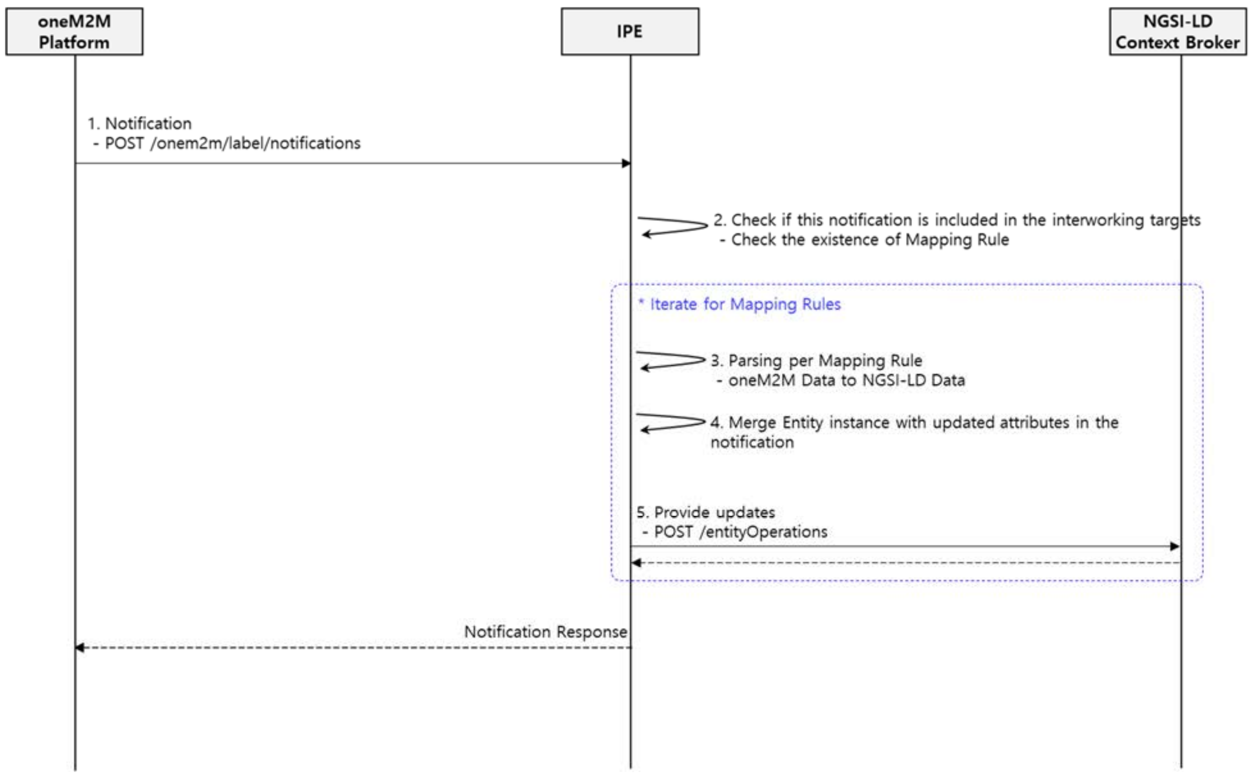

oneM2M Platform
IPE
NGSI-LD Context Broker
1. Notification
- POST /onem2m/label/notifications
2. Check if this notification is included in the interworking targets
- Check the existence of Mapping Rule
* Iterate for Mapping Rules
3. Parsing per Mapping Rule
- oneM2M Data to NGSI-LD Data
4. Merge Entity instance with updated attributes in the notification
5. Provide updates
- POST /entityOperations
Notification Response

Figure 5.4.3-1: Procedures for Interworking Data

Step 1. IPE receives a notification requests per subscription.

Step 2. Check if there is a corresponding mapping rule for the subscriptionReference in the notification.

Step 3. Per corresponding mapping rule, convert oneM2M resource representation into NGSI-LD Entity instance.

Step 4. Merge NGSI-LD Entity instance with the updated one(s) from step 3.

Step 5. Provide updates to the NGSI-LD Entity instance via Batch Entity Creation of Update (Upsert) interface.

5.5 Historical data synchronization procedures

Clause 5.4 illustrates the procedure to get the latest oneM2M data instances, in contentInstance resource type, and synchronize them into NGSI-LD systems as NGSI-LD Entities. Since a <contentInstance> resource is immutable, every data instance is stored as a new <contentInstance> resource. In this sense, for a data interworking proxy, to fetch historical data instances from a oneM2M platform, the proxy needs to retrieve multiple <contentInstance> resources.

[Step 4 modification in clause 5.4.2 - replacing single latest instance retrieval into historical instances fetch]

To retrieve multiple resources with specific conditions (e.g. time period), the ***Result Content*** and the ***Filter Criteria*** request parameter can be used. Among the allowed ***Result Content*** parameter values, to get <contentInstance> resources, "child resources" can be used. To get instances for an interested period of time, the *createdAfter* (cra) and the *createdBefore* (crb) conditions of the ***Filter Criteria*** parameter can be set. The conditions match with the *creationTime* (ct), which is one of the universal attributes in oneM2M, of <contentInstance> resources. Also, there can be other resources in different types than *contentInstance*, the *resourceType* (ty) condition also needs to be set as *contentInstance* (enum value "4") in the ***Filter Criteria***.

NOTE: Historical data retrieval request with the number of instances is being standardized in oneM2M Release 5. When it is published, a corresponding procedure can also be illustrated.

[Step 5 modification in clause 5.4.2 - replacing single latest instance retrieval into historical instances fetch]

As the same as step 4 modification, to retrieve historical resources with specific conditions (e.g. time period), the Result Content and the Filter Criteria request parameter can be used. Among the allowed ***Result Content*** parameter values, to get <contentInstance> resources, "child resources" can be used. To get instances for an interested period of time, the *createdAfter* (cra) and the *createdBefore* (crb) conditions of the ***Filter Criteria*** parameter can be set. The conditions match with the *creationTime* (ct), which is one of the universal attributes in oneM2M, of <contentInstance> resources. Also, there can be other resources in different types than *contentInstance*, the *resourceType* (ty) condition also needs to be set as *contentInstance* (enum value "4") in the ***Filter Criteria***.

[Step 6 extension in clause 5.4.2 - Parsing per mapping information in the label]

If a Iwked-Related-Resources value exists in the mapping rule, the IPE merges historical data instances of the resources, that are mapped to the NGSI-LD Entity. To merge historical data from more than one resources, there needs a consideration for null value handling policy. For example, there are two <container> resources representing two IoT sensors, instances from the two could have different timestamps (i.e. lastModifiedTime). Sensor A reported humidity at 13:00:00PM and sensor B reported temperature at 13:00:07PM. The merged data for 13:00:07PM can include the sensor A measurement at 13:00:00PM, which is the last report by the 13:00:07PM.

When the merge is done, for each merged resource instances, the IPE converts historical data instances of oneM2M resources into NGSI-LD Entity historical evolutions using corresponding mapping rule and provides updates to the NGSI-LD Entity instance.

[Step 7 modification in clause 5.4.2 - save parsed NGSI-LD Entity instance]

IPE saves the latest converted resource instance among the merged resource instances from step 6 and store them.

6 Solution 2 - Mapping ontology based dynamic interworking

6.1 Introduction

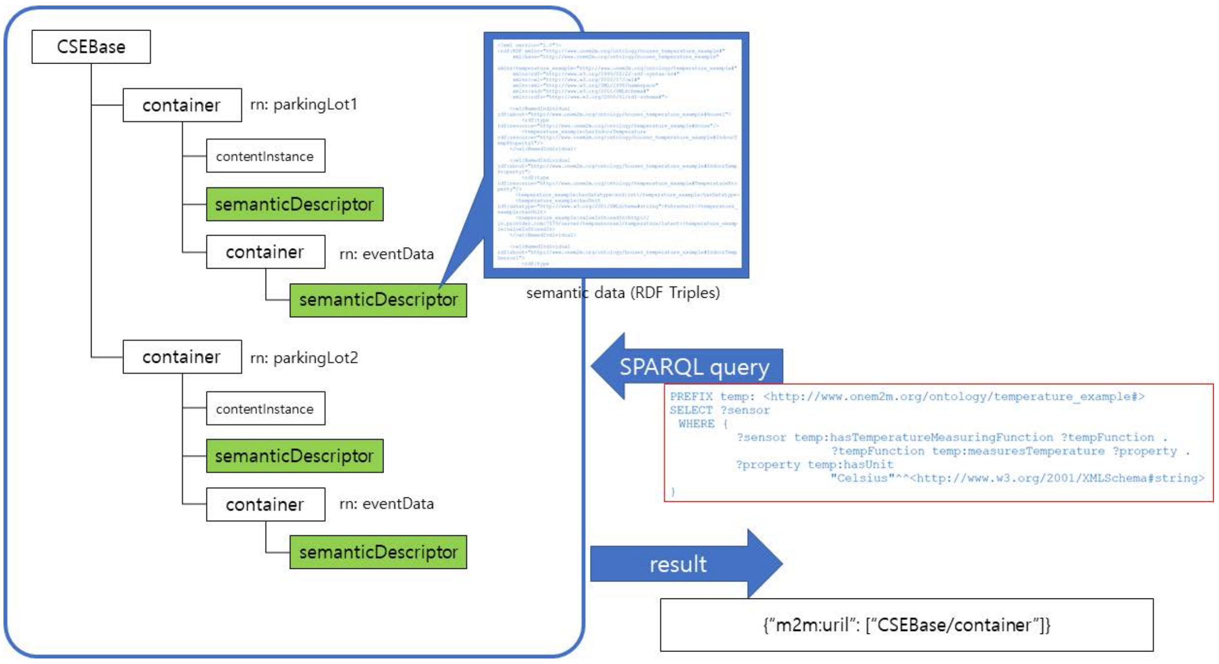
oneM2M and NGSI-LD both supports semantic data and this is the main motivation to define the semantics based interworking scheme. In case of oneM2M, non-semantic data can be annotated for an ontology and RDF triples can be stored and (query, reasoning, mash-up, etc.) used by oneM2M applications. In NGSI-LD, the core meta-model is defined upon semantic groundings such as RDFS and cross-domain ontologies. Therefore, it is feasible to map semantic data representations in both systems and the procedures for this semantic interworking is derived in this clause.

Since typical use case for oneM2M and NGSI-LD interworking would be to collect and manage city data from IoT infrastructures, the present document elaborates oneM2M to NGSI-LD interworking method. However, the other directional interworking can be done leveraging the mapping ontology in clause 6.3.2.

6.2 oneM2M features

6.2.1 Overall semantic capabilities

The basic of oneM2M-supported semantic features are storing RDF triple data that is defined by W3C® (World Wide Web Consortium) and get the result of SPARQL (SPARQL Protocol and RDF Query Language) queries. Storing a set of RDF triples does not mean creating an arbitrary semantic resource in oneM2M resource tree, but putting semantic annotation to the parent resource. For instance, when creating a <*semanticDescriptor*> resource under a <*container*> resource, the semantic resource contains the semantic annotation of the parent <*container*> resource.



```
PREFIX temp: <http://www.onem2m.org/ontology/temperature_example#>
SELECT ?sensor
 WHERE {
        ?sensor temp:hasTemperatureMeasuringFunction ?tempFunction .
                    ?tempFunction temp:measuresTemperature ?property .
        ?property temp:hasUnit
                    "Celsius"^^<http://www.w3.org/2001/XMLSchema#string>
}
```

```
{"m2m:uril": ["CSEBase/container"]}
```

Figure 6.2.1-1: A resource tree with semanticDescriptor resources and its usage

Figure 6.2.1-1 depicts the oneM2M resource tree containing semantic descriptor resources. By the definition in ETSI TS 118 101 [i.1], it can be created under several resource type instances (e.g. *container*).

Unlikely the normal oneM2M discovery requests, semantic discovery includes SPARQL query. When a <*semanticDescriptor*> resource has a matching result with the query, then the oneM2M platform returns the parent resource identifier of the semantic resource.

6.2.2 *semanticDescriptor* resource type

Table 6.2.2-1 shows some resource specific attributes of the <*semanticDescriptor*> resource type from o ETSI TS 118 101 [i.1]. The semantic Descriptor defines descriptor (drp) attribute to store RDF triples. Since there are several standard serialization formats for RDF triples, the descriptorRepresentation (dcrp) attribute indicates which format is used for the descriptor attribute. The ontologyRef (or) attribute refers to the ontology by its URI that is used in the descriptor.

By the parent-child resource type definition in oneM2M, a <*semanticDescriptor*> resource describes the parent resource in RDF. In case of Figure 6.2.1-1, each semantic resource annotates the parent *container* resource.

Table 6.2.2-1: Resource specific attributes of <*semanticDescriptor*> resource type

Attribute Name	Short Name	Description
descriptorRepresentation	dcrp	Indicates the type used for the serialization of the descriptor attribute, e.g. RDF/XML, OWL/XML.
descriptor	drp	Stores a semantic description pertaining to a resource and potentially sub-resources. Such a description need to be according to subject predicate-object triples as defined in the RDF graph-based data model. Examples of such descriptors in RDF can be found in oneM2M TR-0007 [i.6].
ontologyRef	or	A reference (URI) of the ontology used to represent the information that is stored in the *descriptor attribute*. If this attribute is not present, the *ontologyRef* from the parent resource is used if present.

6.2.3 Semantic discovery and query

With the SPARQL query execution, oneM2M specifies semantic discovery and semantic query. As depicted in Figure 6.2.1-1, oneM2M semantic discovery results are resource identifiers of the parent <*semanticDescriptor*> resources that match with SPARQLs.

On the other hand, when semantic queries are requested, the SPARQL query results from the execution gets returned to the Originators. Depending on the application scenario, semantic discovery can be used to further retrieve non-semantic service data (e.g. *contentInstance*) or get the SPARQL results as processed data.

SPARQL query statements in oneM2M request is included in the *semanticsFilter* condition of the ***Filter Criteria*** request parameter. To distinguish semantic discovery and query, the ***Semantic Query Indicator*** request parameter is used. When its value is true, it is considered as the semantic query, while false indicates semantic discovery.

6.2.4 Semantic discovery

Per oneM2M parent-child relationship specification, a oneM2M semantic discovery returns parent resource identifiers of the matching <*semanticDescriptor*> resources to the SPARQL query. For example, when a semantic discovery wants to find thermometers with a specific smart home ontology, the oneM2M platform gets <*semanticDescriptor*> resources that matches with the SPARQL having that specific ontology and returns the parent temperature data containing <*container*> resource identifier to the Originator.

This discovery interface is useful when an application discovers data in oneM2M platform with specific semantics. When that specific data resource (e.g. *container* with a semantic description) is discovered, the application can use the child <*contentInstance*> resources. In this case, content of <*contentInstance*> resource should be latest temperature readings from the thermometer. To get the latest sensor readings, the application also can use the subscription/notification feature from the oneM2M platform.

6.2.5 Semantic query

As highlighted in clause 6.2.3, the semantic query returns SPARQL query execution results. This can be helpful for applications that want a sort of processed data like count. With the semantic discovery, the application needs to send subsequent requests to get the information, but it could be composed as the SPARQL query so the answer can be directly derived for the semantic descriptions.

6.3 Resource mapping

6.3.1 Introduction

It is assumed that a (sub-)resource tree in oneM2M platform consists of NGSI-LD Entity instance. This also means that in a (sub-)resource tree there is no more one NGSI-LD Entity instance map-able representation. When a oneM2M resource represents an NGSI-LD Entity, the child resources refers NGSI-LD Attributes (i.e. Property of Relationship).

There are a list of requirements for oneM2M - NGSI-LD interworking:

- The interworking scheme uses the NGSI-LD ontology (e.g. ngsi-ld:Entity) in the ETSI GS CIM 009 [i.2].
- The interworking scheme defines resource mapping rules between oneM2M (e.g. <*container*> resource) and NGSI-LD (e.g. Entity).
- The interworking scheme specifies addressing method to indicate an attribute of oneM2M and NGSI-LD systems.
- The interworking scheme defines the method to annotate a <*contentInstance*> resource attribute to create the <*semanticDescriptor*> resource as the child of a <*container*> resource (i.e. sibling to the <*contentInstance*> resource). This avoids unnecessarily duplicated <*semanticDescriptor*> resource creation at every <<*contentInstance*> resource creation.
- The interworking scheme specified data type definitions to support data type conversion.

6.3.2 Mapping ontology

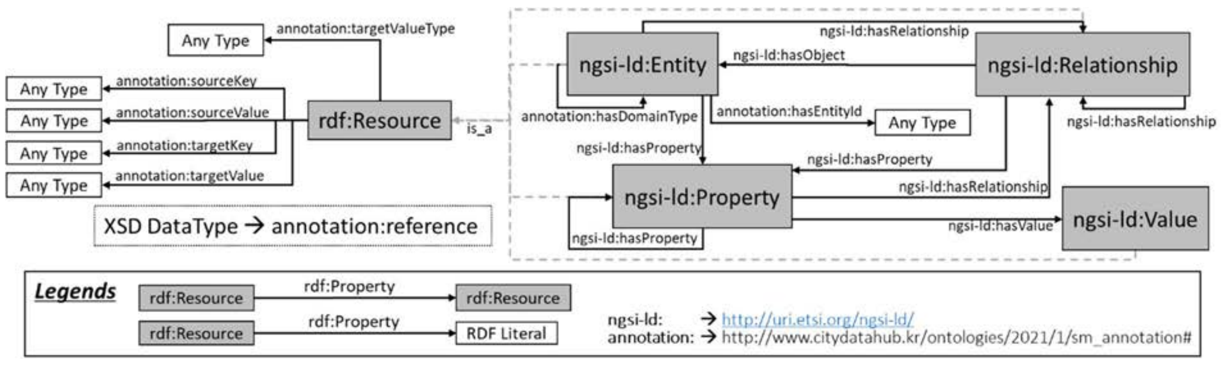


Figure 6.3.2-1: oneM2M - NGSI-LD Mapping Ontology

As depicted in Figure 6.2.1-1, the mapping ontology between oneM2M and NGSI-LD basically defines source key/value from oneM2M resources and target key/value of NGSI-LD Entity instances from NGSI-LD context brokers. The concept definitions having the URIs with prefix "ngsi-ld:" are the core concepts of NGSI-LD meta-model, whereas the ones having the URIs with prefix "annotation:" represent the concept from the mapping ontology. The properties "annotation:sourceKey" and "annotation:sourceValue" are used to map the oneM2M resource attributes as JSON key-value pairs, whereas the properties "annotation:targetKey" and "annotation:targetValue" are used to map the respective NGSI-LD Entity instance attributes as JSON-LD key-value pairs. When a value from oneM2M resource is used as an address or reference, it needs to be handled separately, so the data type "annotation:reference " is added.

Table 6.3.2-1 describes the mapping ontologies for semantic annotations.

Table 6.3.2-1: Mapping Ontologies

Ontology	Description
annotation:ngsi-ldContext	Indicates the @context URIs of the NGSI-LD Entity.
annotation:sourceValueType	Data type of a oneM2M resource attribute value in W3C defined data type URL (e.g. http://www.w3.org/2001/XMLSchema#float).
annotation:sourceValue	Attribute value of a oneM2M resource.
annotation:sourceKey	Attribute address of a oneM2M resource.
annotation:sourceTargetType	Data type of an NGSI-LD Entity's Attribute (e.g. a Property) value in W3C defined data type URL (e.g. http://www.w3.org/2001/XMLSchema#float).
annotation:targetValue	Attribute value of an NGSI-LD Entity.
annotation:targetKey	Attribute address of an NGSI-LD Entity.
annotation:hasDomainType	Specify the type of the Entity instance as a direct/indirect subclass of ngsi-ld:Entity. (e.g. Parking being a subclass of Entity, specified in a user-defined @context).
annotation:hasEntityId	Specify the @type of the Entity instance if its value is not in a URI format.

6.3.3 Attribute addressing

When refers to an attribute address, there should be a fixed addressing scheme. In Table 6.3.2-1, sourceKey and targetKey use this addressing scheme.

The attribute address notation uses JSON and Unix file system conventions. For instance, "../cnt.rn" refers the rn (resourceName) attribute of the parent <*container*> resource. In the same sense, "./la.con.loc.PointSystem.long" refers the sibling la (latest) <*contentInstance*> resource. In the <*contentInstance*> resource, location → PointSystem → long property of the con (content) attribute is referred. In this case the value type of the address should be "annotation:reference", in order for it to be properly dereference during the translation.

In some cases, the attribute value may involve both address and a static value. In order for a system to distinguish between the two, the address part should be enclosed in "{}" (curly braces) inside which, same addressing scheme need to be followed as described above. In this case, the translation system will treat the attribute value as one of the specified XSD data type, until it locates the address enclosed in "{}". Therefore "{}" are reserved for this purpose. In case of attribute value type being "annotation:reference", the complete value should be considered as address.

6.3.4 Semantic annotation example

The semantic annotation examples are based on a parking spot data stored as a oneM2M resource, which is required to be mapped using the Mapping ontology, into the <*semanticDescriptor*> resource. The example resource content is as follows:

```
{
    "m2m:cin": {
        "rn": "cin_01",
        "ty": 4,
        "pi": "SkmnDBfeHX",
        "ri": "SymEIuUHrQ",
        "ct": "20180806T051036",
        "et": "20210806T051036",
        "lt": "20180806T051036",
        "con": {
            "name": "spot_001",
            "Datelog": {
            "creation": "2018-11-15T20:10:00,000+09:00"
        },
            "category": [
                "forDisabled"
                ],
            "loc": {
            "type": "PointSystem",
              "PointSystem": {
                "lat": -2.558246945,
                "long": 50.378546923
            }
            },
        "Dimensions": {
            "width": 2.5,
            "length": 5.1,
            "unit": "m"
        }
        "availability": {
            "value": "free",
            "observation": "2018-11-15T20:09:55,000+09:00",
        }
        },
}
```

In this case, *contentInstance* resource is considered for the parking spot data. The main attributes to be mapped reside under "con" attribute.

Other attributes may also be required for mapping purposes, such as "rn", which in this example, is required to define the Entity Identifier. This attribute is defined in the parent <*container*> resource. The parent <*container*> resource contains conventional attributes based on oneM2M specifications, as shown below:

```
{
    "m2m:cnt": {
        "pi": "3-20201104210538537018",
        "ri": "3-20201104210928149670",
        "ty": 3,
        "ct": "20201104T210928",
        "st": 21876,
        "rn": "spot_001",
        "lt": "20201104T210928",
        "et": "20221104T210928",
        "lbl": [
            "sc"
        ],
        "cr": "SW001",
        "mni": 3153600000,
        "mbs": 3153600000,
        "mia": 31536000,
        "cni": 21875,
        "cbs": 94052
    }
}
```

Per the mapping ontology definition with the addressing scheme, semantic annotations for oneM2M <*container*> resource can be:

```
<rdf:RDF xmlns:annotation="http://www.citydatahub.kr/ontologies/2021/1/sm_annotation#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:ngsi-ld="http://uri.etsi.org/ngsi-ld/">
    <annotation:Mapping rdf:about="http://www.citydatahub.kr#example_graph_uri">
        <annotation:ngsi-ldContext
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">http://uri.citydatahub.kr/ngsi-
ld/parking.jsonld</annotation:ngsi-ldContext>
    </annotation:Mapping>

    <ngsi-ld:Entity
rdf:about="http://www.citydatahub.kr/ontologies/2021/1/sm_annotation#iotParking">
        <annotation:hasEntityId
rdf:datatype="http://www.citydatahub.kr/ontologies/2021/1/sm_annotation#reference">urn:datahub:Par
kingSpot:{../cnt.rn}</annotation:hasEntityId>
        <annotation:hasDomainType rdf:resource="http://www.citydatahub.kr/parking#ParkingSpot"/>
    </ngsi-ld:Entity>
</rdf:RDF>
```

Its resource involves two main mapping definitions: one is the @context definition, and other is the Entity definition. The Entity mappings involves the mapping for its Id and the domain type. Furthermore, the URI "http://www.citydatahub.kr#example_graph_uri" defines the identifier of the mapping resource, which can be used to locate it in the oneM2M system as well as in the triple store (if required).

With the different addressing notation, specifically for <*contentInstance*> resources, example semantic description for NGSI-LD mapping can be:

```
<rdf:RDF xmlns:annotation="http://www.citydatahub.kr/ontologies/2021/1/sm_annotation#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:ngsi-ld="http://uri.etsi.org/ngsi-ld/">
    <annotation:Mapping rdf:about="http://www.citydatahub.kr#example_graph_uri">
        <annotation:ngsi-ldContext
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">http://uri.citydatahub.kr/ngsi-
ld/parking.jsonld</annotation:ngsi-ldContext>
    </annotation:Mapping>

    <ngsi-ld:Entity
rdf:about="http://www.citydatahub.kr/ontologies/2021/1/sm_annotation#iotParking">
        <ngsi-ld:hasProperty>
            <ngsi-ld:Property rdf:about="http://www.citydatahub.kr/parking#status">
                <annotation:sourceValueType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">http://www.w3.org/2001/XMLSchema#string</an
notation:sourceValueType>
```

```
                <annotation:sourceKey
rdf:datatype="http://www.citydatahub.kr/ontologies/2021/1/sm_annotation#reference">./la.con.availa
bility.value</annotation:sourceKey>
                <ngsi-ld:hasProperty>
                    <ngsi-ld:observedAt
rdf:about="http://www.citydatahub.kr/ontologies/2021/1/sm_annotation#observation_1">
                        <annotation:sourceValueType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">http://www.w3.org/2001/XMLSchema#dateTimeSt
amp</annotation:sourceValueType>
                        <annotation:sourceKey
rdf:datatype="http://www.citydatahub.kr/ontologies/2021/1/sm_annotation#reference">./la.con.availa
bility.observation</annotation:sourceKey>
                    </ngsi-ld:observedAt>
                </ngsi-ld:hasProperty>
            </ngsi-ld:Property>
        </ngsi-ld:hasProperty>
        <ngsi-ld:hasProperty>
            <ngsi-ld:Property rdf:about="http://www.citydatahub.kr/parking#width">
                <annotation:sourceValueType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">http://www.w3.org/2001/XMLSchema#float</ann
otation:sourceValueType>
                <annotation:sourceKey
rdf:datatype="http://www.citydatahub.kr/ontologies/2021/1/sm_annotation#reference">./la.con.Dimens
ions.width</annotation:sourceKey>
            </ngsi-ld:Property>
        </ngsi-ld:hasProperty>
        <ngsi-ld:hasProperty>
            <ngsi-ld:Property rdf:about="http://www.citydatahub.kr/parking#length">
                <annotation:sourceValueType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">http://www.w3.org/2001/XMLSchema#float</ann
otation:sourceValueType>
                <annotation:sourceKey
rdf:datatype="http://www.citydatahub.kr/ontologies/2021/1/sm_annotation#reference">./la.con.Dimens
ions.length</annotation:sourceKey>
                <ngsi-ld:hasProperty>
                    <ngsi-ld:Property rdf:about="http://www.citydatahub.kr/parking#unit">
                        <annotation:targetValue>meter</annotation:targetValue>
                        <annotation:sourceValueType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">http://www.w3.org/2001/XMLSchema#string</an
notation:sourceValueType>
                        <annotation:sourceValue>m</annotation:sourceValue>
                        <annotation:sourceKey
rdf:datatype="http://www.citydatahub.kr/ontologies/2021/1/sm_annotation#reference">./la.con.Dimens
ions.unit</annotation:sourceKey>
                    </ngsi-ld:Property>
                </ngsi-ld:hasProperty>
            </ngsi-ld:Property>
        </ngsi-ld:hasProperty>
        <ngsi-ld:hasProperty>
            <ngsi-ld:GeoProperty rdf:about="http://www.citydatahub.kr/parking#location">
                <annotation:targetValueType>Point</annotation:targetValueType>
                <ngsi-ld:hasValue>
                    <ngsi-ld:Value rdf:about="http://www.citydatahub.kr/parking#value_2">
                        <annotation:targetKey
rdf:datatype="http://www.citydatahub.kr/ontologies/2021/1/sm_annotation#reference">location.value.
coordinates[1]</annotation:targetKey>
                        <annotation:sourceValueType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">http://www.w3.org/2001/XMLSchema#double</an
notation:sourceValueType>
                        <annotation:sourceKey
rdf:datatype="http://www.citydatahub.kr/ontologies/2021/1/sm_annotation#reference">./la.con.loc.Po
intSystem.long</annotation:sourceKey>
                    </ngsi-ld:Value>
                </ngsi-ld:hasValue>
                <ngsi-ld:hasValue>
                    <ngsi-ld:Value rdf:about="http://www.citydatahub.kr/parking#value_1">
                        <annotation:targetKey
rdf:datatype="http://www.citydatahub.kr/ontologies/2021/1/sm_annotation#reference">location.value.
coordinates[0]</annotation:targetKey>
                        <annotation:sourceValueType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">http://www.w3.org/2001/XMLSchema#double</an
notation:sourceValueType>
                        <annotation:sourceKey
rdf:datatype="http://www.citydatahub.kr/ontologies/2021/1/sm_annotation#reference">./la.con.loc.Po
intSystem.lat</annotation:sourceKey>
                    </ngsi-ld:Value>
                </ngsi-ld:hasValue>
            </ngsi-ld:GeoProperty>
        </ngsi-ld:hasProperty>
        <ngsi-ld:hasProperty>
            <ngsi-ld:Property rdf:about="http://www.citydatahub.kr/parking#category">
                <annotation:sourceKey
rdf:datatype="http://www.citydatahub.kr/ontologies/2021/1/sm_annotation#reference">./la.con.catego
ry</annotation:sourceKey>
                <annotation:sourceValueType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">http://www.w3.org/2001/XMLSchema#string</an
notation:sourceValueType>
            </ngsi-ld:Property>
        </ngsi-ld:hasProperty>
        <ngsi-ld:hasProperty>
            <ngsi-ld:Property rdf:about="http://www.citydatahub.kr/parking#name">
                <annotation:sourceValueType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">http://www.w3.org/2001/XMLSchema#string</an
notation:sourceValueType>
                <annotation:sourceKey
rdf:datatype="http://www.citydatahub.kr/ontologies/2021/1/sm_annotation#reference">./la.con.name</
annotation:sourceKey>
            </ngsi-ld:Property>
        </ngsi-ld:hasProperty>
    </ngsi-ld:Entity>
</rdf:RDF>
```

For ease of understanding, the mappings can be visualized in Figure 6.3.4-1.

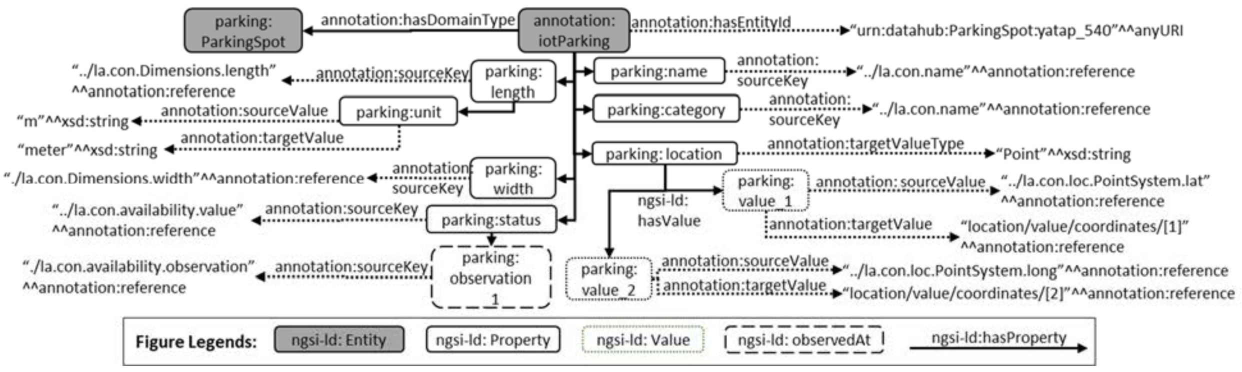


Figure 6.3.4-1: Mapping Example Visual Representation

Here different mappings are worth noticing. The RDF resources such as "http://www.citydatahub.kr/parking#name", "http://www.citydatahub.kr/parking#width", etc. do not involve "annotation:targetValue" property. This is because the property "annotation:sourceKey" can be used to locate the value for that key in the <*contentInstance*> resource. The RDF resource "http://www.citydatahub.kr/parking#location" is an example of complex value mapping using two instances of ngsi-ld:Value namely: "http://www.citydatahub.kr/parking#value_1" and "http://www.citydatahub.kr/parking#value_2".

These are defined as the attribute value structure in the oneM2M resource is not aligned with the target structure in the NGSI-LD for its respective Property. Therefore the value cannot be implicitly mapped using "annotation:sourceKey" only. For each of these instances, "annotation:targetValue" has also been used which specifies the address of the value to the positions in the array for the respective NGSI-LD Property. The RDF resource "http://www.citydatahub.kr/parking#unit" shows another usage of "annotation:targetValue". As the unit value defined in source (oneM2M resource) and target (NGSI-LD Entity instance) are not the same, the mapping defines an "annotation:sourceValue" as a condition for the respective "annotation:targetValue". In case of multiple conditions, instances of "ngsi-ld:Value" can be defined, for each condition. These instances will be linked with the RDF resource "http://www.citydatahub.kr/parking#unit" using the property "ngsi-ld:hasValue".

The translated NGSI-LD Entity is shown as below:

```
{
   "@context": [
      "http://uri.etsi.org/ngsi-ld/core-context.jsonld",
      "http://cityhub.kr/ngsi-ld/parking.jsonld"
   ],
   "id": "urn:datahub:ParkingSpot: spot_001",
   "type": "ParkingSpot",
   "createdAt": "2018-11-15T20:10:00,000+09:00",
   "modifiedAt": "2018-11-15T20:10:00,000+09:00",
   "name": {
      "type": "Property",
      "value": "spot_001"
   },
   "location": {
      "type": "GeoProperty",
      "value": {
         "type": "Point",
         "coordinates": [
            127.1293735,
            37.4114423
         ]
      }
   },
   "address": {
      "type": "Property",
      "value": {
         "addressCountry": "KR",
         "addressRegion": "Gyeonggi-do",
         "addressLocality": "Seongnam-si",
         "streetAddress": "8th Seungin-ro",
         "addressTown": "Yatap-dong"
      }
   },
   "category": {
      "type": "Property",
      "value": [
         "forDisabled"
      ]
   },
   "width": {
      "type": "Property",
      "value": 2.5
   },
   "length": {
      "type": "Property",
      "value": 5.1
   },
   "status": {
      "type": "Property",
      "value": "free",
      "observedAt": "2018-11-15T20:09:55,000+09:00"
   },
   "refParkingLot": {
      "type": "Property",
      "value": "urn:ngsi-ld:OffStreetParking:5038"
   }
}
```

6.3.5 Translation Guidelines

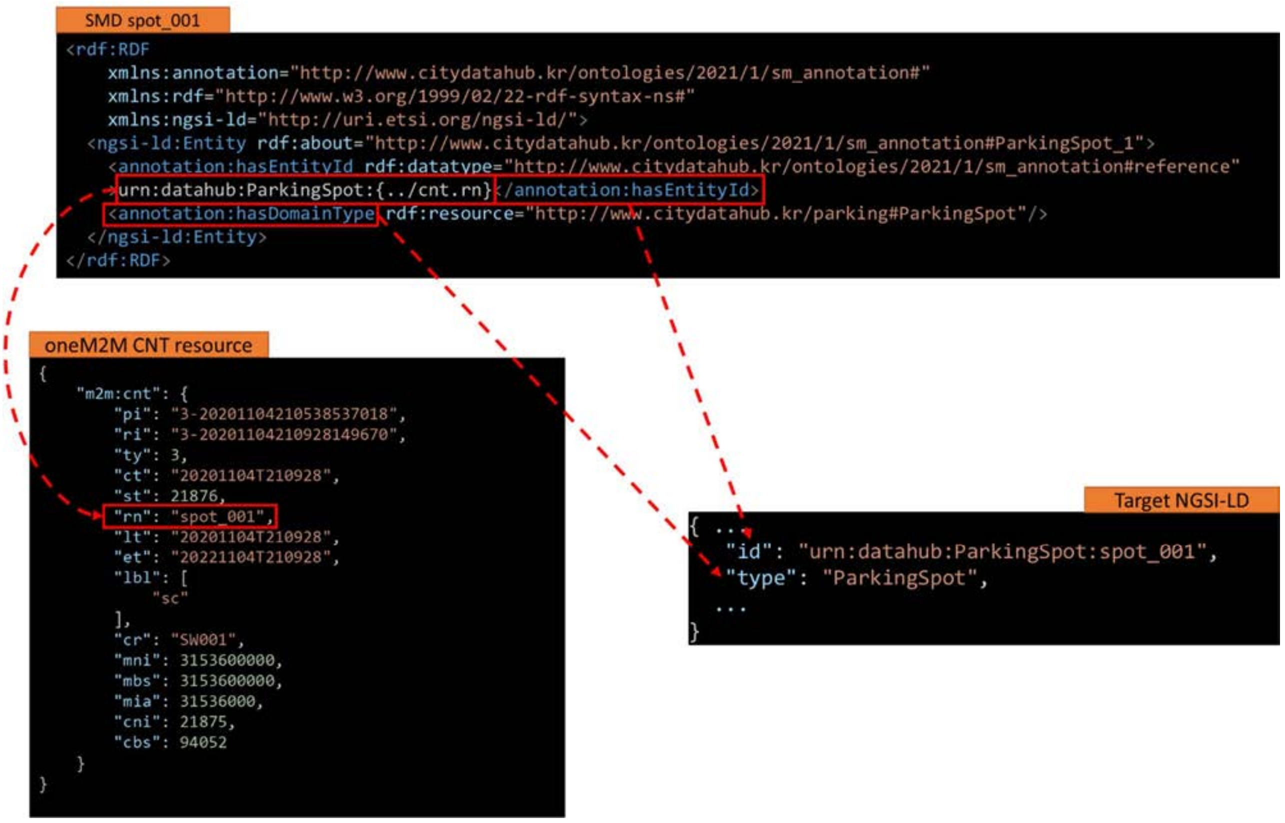
For better explanation of the translation guidelines, the parking spot example in clause 6.3.4 has been used. The guidelines are defined as follows:

1) If the rdf:datatype is annotation:reference, it means that the value is referring to an address, from where the actual value can be retrieved.

   1.1) If the value is enclosed with "{ }", then that part of value should be considered as reference. The other part outside "{ }" will be treated as a value.

2) The tag <ngsi-ld:Entity> defines all the mapping information related to an Entity.

3) annotation:hasEntityId defines the type of NGSI-LD Entity.



```
<rdf:RDF
    xmlns:annotation="http://www.citydatahub.kr/ontologies/2021/1/sm_annotation#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:ngsi-ld="http://uri.etsi.org/ngsi-ld/">
  <ngsi-ld:Entity rdf:about="http://www.citydatahub.kr/ontologies/2021/1/sm_annotation#ParkingSpot_1">
    <annotation:hasEntityId rdf:datatype="http://www.citydatahub.kr/ontologies/2021/1/sm_annotation#reference"
    urn:datahub:ParkingSpot:{../cnt.rn}</annotation:hasEntityId>
    <annotation:hasDomainType rdf:resource="http://www.citydatahub.kr/parking#ParkingSpot"/>
  </ngsi-ld:Entity>
</rdf:RDF>
```

```
{
    "m2m:cnt": {
        "pi": "3-20201104210538537018",
        "ri": "3-20201104210928149670",
        "ty": 3,
        "ct": "20201104T210928",
        "st": 21876,
        "rn": "spot_001",
        "lt": "20201104T210928",
        "et": "20221104T210928",
        "lbl": [
            "sc"
        ],
        "cr": "SW001",
        "mni": 3153600000,
        "mbs": 3153600000,
        "mia": 31536000,
        "cni": 21875,
        "cbs": 94052
    }
}
```

```
{ ...
  "id": "urn:datahub:ParkingSpot:spot_001",
  "type": "ParkingSpot",
  ...
}
```

Figure 6.3.5-1: Example illustration for Guidelines 1-3

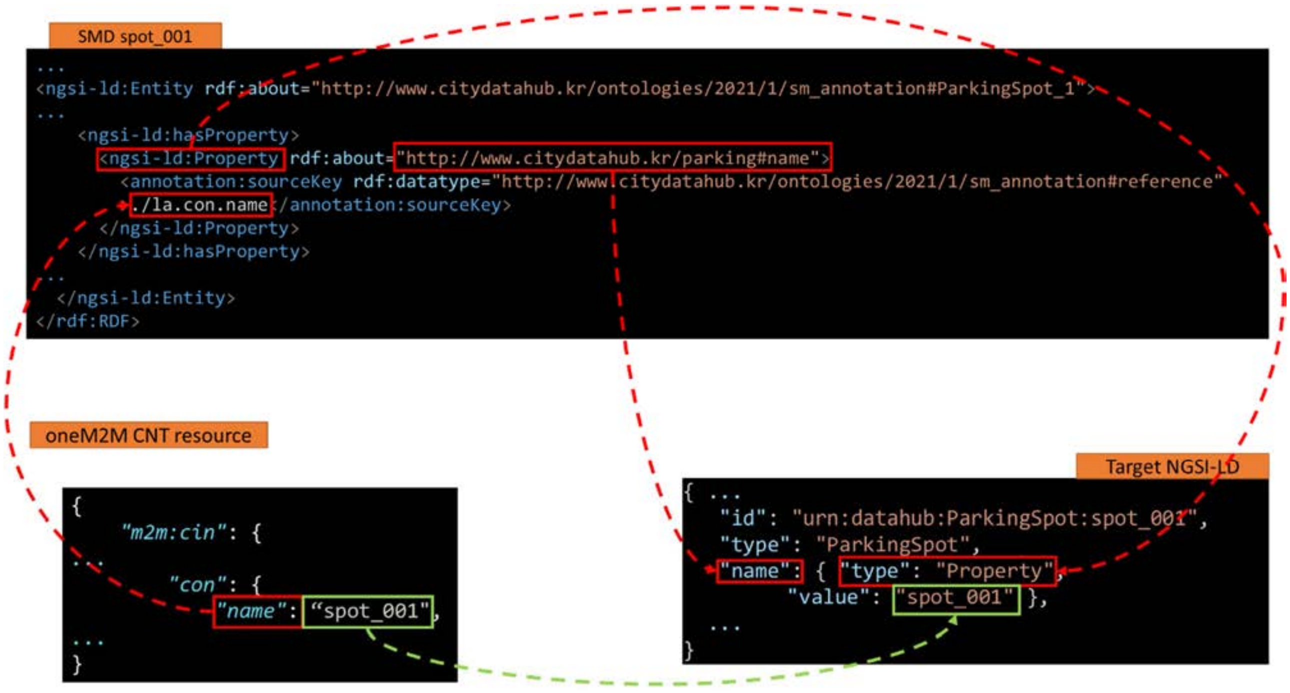
4) The tag <ngsi-ld:Property> defines the mapping related to the NGSI-LD Property.

   4.1) This will be used to define "type": "Property" in NGSI-LD data.

   4.2) "rdf:about" defines the id of the NGSI-LD Property. Since in NGSI-LD, the labels is only used, therefore the prefix (URI part before #) will be stripped.

5) annotation:sourceKey defines the information related to the Source Key in the Key-Value pairs in oneM2M resource.

6) When there is no other information given under the tag <ngsi-ld:Property>, then the value of the Source Key will be used as is for the target NGSI-LD data.



```
...
<ngsi-ld:Entity rdf:about="http://www.citydatahub.kr/ontologies/2021/1/sm_annotation#ParkingSpot_1">
...
    <ngsi-ld:hasProperty>
      <ngsi-ld:Property rdf:about="http://www.citydatahub.kr/parking#name">
        <annotation:sourceKey rdf:datatype="http://www.citydatahub.kr/ontologies/2021/1/sm_annotation#reference"
         ./la.con.name</annotation:sourceKey>
      </ngsi-ld:Property>
    </ngsi-ld:hasProperty>
 ...
  </ngsi-ld:Entity>
</rdf:RDF>
```

```
{
    "m2m:cin": {
...
        "con": {
          "name": “spot_001",
...
}
```

```
{ ...
    "id": "urn:datahub:ParkingSpot:spot_001",
    "type": "ParkingSpot",
  "name": { "type": "Property",
          "value": "spot_001" },
  ...
}
```

Figure 6.3.5-2: Example illustration for Guidelines 4-6

7) The tag <ngsi-ld:GeoProperty> defines the mapping related to the NGSI-LD GeoProperty. Therefore, its format representation should be known implicitly.

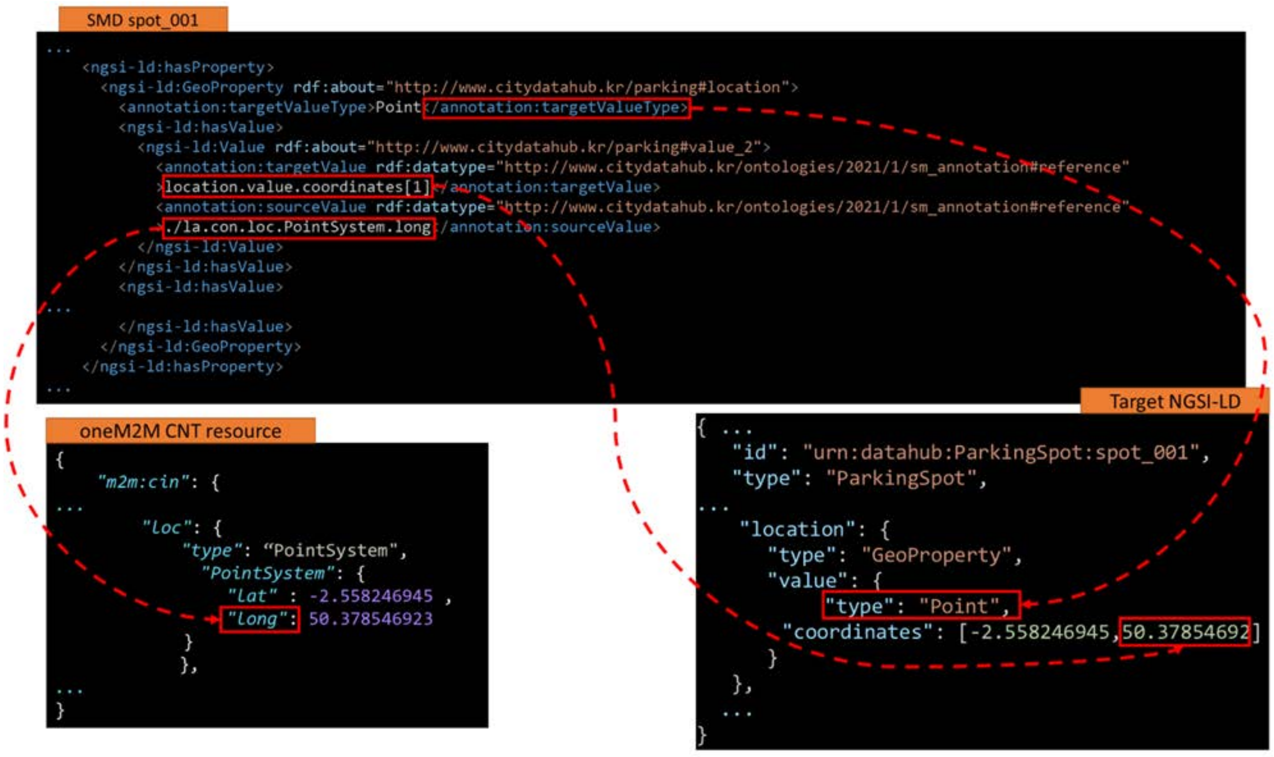
  7.1) This will be used to define "type": "Property" in NGSI-LD.

  7.2) The tag <annotation:targetValueType> is used to define specific NGSI-LD type for the target value in NGSI-LD.

8) Here, since the Value in Source has different structure than the one in Target, <ngsi-ld:Value> tag is required to specify their mapping features.

  8.1) The tag <annotation:sourceValue> defines mapping for the values in oneM2M resource, however for the reference, the respective key will be specified.

  8.2) The tag <annotation:targetValue> having rdf:datatype "annotation:reference", indicates the address where the translated value (in this case: 50.378546923) should be placed.



```
...
   <ngsi-ld:hasProperty>
     <ngsi-ld:GeoProperty rdf:about="http://www.citydatahub.kr/parking#location">
       <annotation:targetValueType>Point</annotation:targetValueType>
       <ngsi-ld:hasValue>
         <ngsi-ld:Value rdf:about="http://www.citydatahub.kr/parking#value_2">
           <annotation:targetValue rdf:datatype="http://www.citydatahub.kr/ontologies/2021/1/sm_annotation#reference"
           >location.value.coordinates[1]</annotation:targetValue>
           <annotation:sourceValue rdf:datatype="http://www.citydatahub.kr/ontologies/2021/1/sm_annotation#reference"
           >./la.con.loc.PointSystem.long</annotation:sourceValue>
         </ngsi-ld:Value>
       </ngsi-ld:hasValue>
       <ngsi-ld:hasValue>
...
       </ngsi-ld:hasValue>
     </ngsi-ld:GeoProperty>
   </ngsi-ld:hasProperty>
...
```

```
{
    "m2m:cin": {
...
        "Loc": {
            "type": "PointSystem",
             "PointSystem": {
                "Lat" : -2.558246945 ,
                "Long": 50.378546923
            }
          },
...
}
```

```
{ ...
   "id": "urn:datahub:ParkingSpot:spot_001",
   "type": "ParkingSpot",
...
    "location": {
      "type": "GeoProperty",
      "value": {
        "type": "Point",
        "coordinates": [-2.558246945,50.37854692]
      }
    },
    ...
}
```

Figure 6.3.5-3: Example illustration for Guidelines 7-8

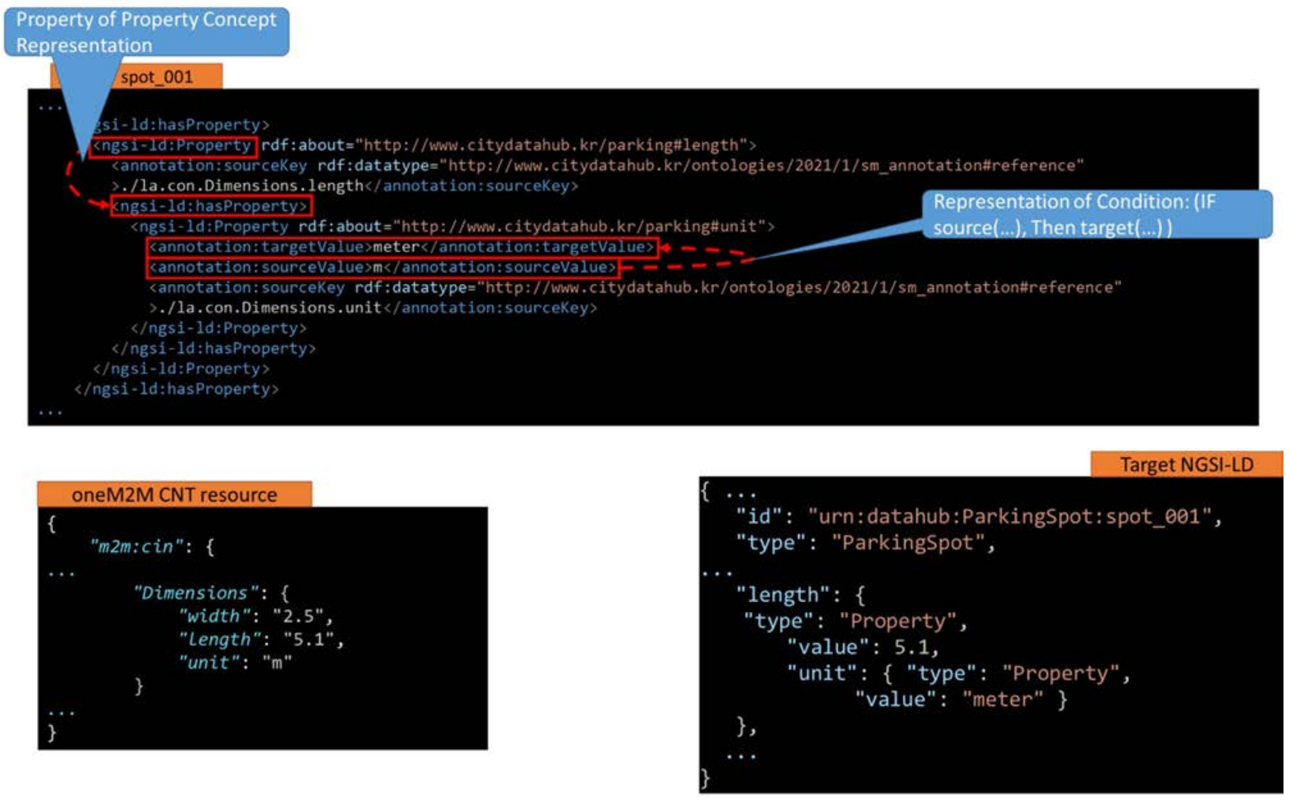
9) To represent the mapping for "Property of Property" concept, a subsequent tag <ngsi-ld:hasProperty> will be defined.

    9.1) This will be used to define "type": "Property" in NGSI-LD.

10) Here, both the tags <annotation:sourceValue> and <annotation:targetValue> are defined, even though the value ("m") is available in the oneM2M resource and could have been referenced.

    10.1) The reason of defining both tags is to put condition: check if the source value is equal to the one specified (in this case "m"), then the target value should be placed accordingly (in this case "meter").

    10.2) In case if the source has any other value (for e.g. "cm"), target value would not be generated.


Property of Property Concept Representation
spot_001
Representation of Condition: (IF source(...), Then target(...))
oneM2M CNT resource
Target NGSI-LD

```
...
  gsi-ld:hasProperty>
  <ngsi-ld:Property rdf:about="http://www.citydatahub.kr/parking#length">
    <annotation:sourceKey rdf:datatype="http://www.citydatahub.kr/ontologies/2021/1/sm_annotation#reference"
    >./la.con.Dimensions.length</annotation:sourceKey>
    <ngsi-ld:hasProperty>
      <ngsi-ld:Property rdf:about="http://www.citydatahub.kr/parking#unit">
        <annotation:targetValue>meter</annotation:targetValue>
        <annotation:sourceValue>m</annotation:sourceValue>
        <annotation:sourceKey rdf:datatype="http://www.citydatahub.kr/ontologies/2021/1/sm_annotation#reference"
        >./la.con.Dimensions.unit</annotation:sourceKey>
      </ngsi-ld:Property>
    </ngsi-ld:hasProperty>
  </ngsi-ld:Property>
</ngsi-ld:hasProperty>
...
```

```
{
    "m2m:cin": {
...
        "Dimensions": {
            "width": "2.5",
            "Length": "5.1",
            "unit": "m"
        }
...
}
```

```
{ ...
   "id": "urn:datahub:ParkingSpot:spot_001",
   "type": "ParkingSpot",
...
   "length": {
    "type": "Property",
        "value": 5.1,
        "unit": { "type": "Property",
              "value": "meter" }
   },
   ...
}
```

Figure 6.3.5-4: Example illustration for Guidelines 9-10

6.4 New data synchronization procedures

6.4.1 NGSI-LD entity creation

This clause defines the semantic interworking procedures using existing oneM2M standard interfaces including semantic discovery.

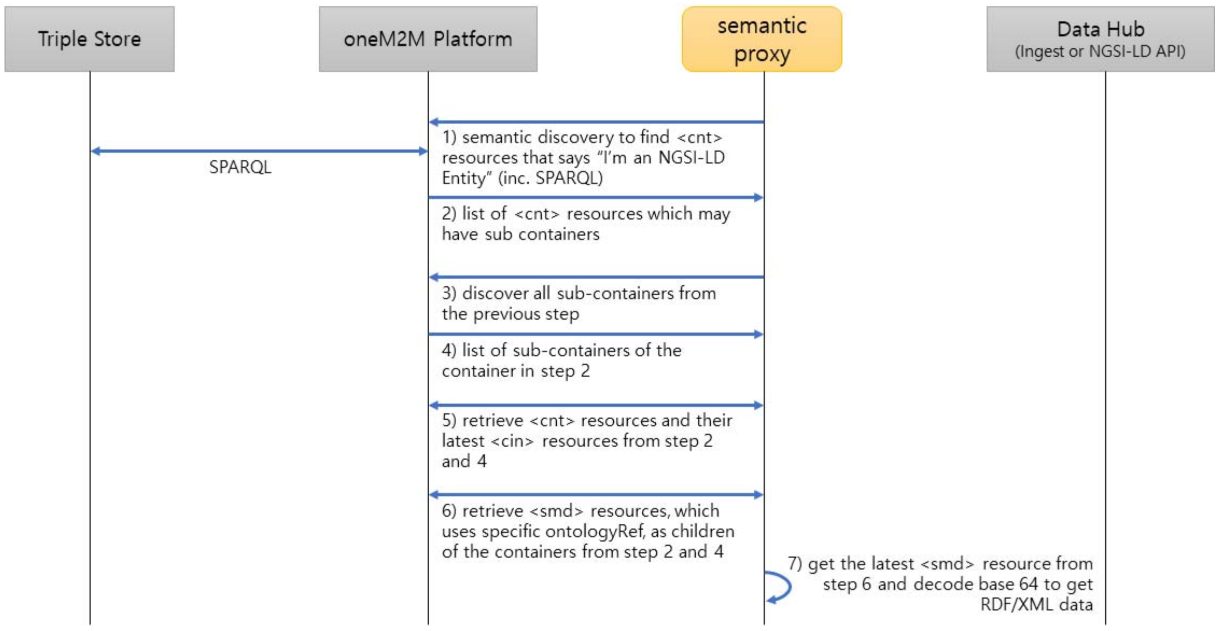

Triple Store
oneM2M Platform
semantic proxy
Data Hub
(Ingest or NGSI-LD API)
SPARQL
1) semantic discovery to find <cnt> resources that says "I'm an NGSI-LD Entity" (inc. SPARQL)
2) list of <cnt> resources which may have sub containers
3) discover all sub-containers from the previous step
4) list of sub-containers of the container in step 2
5) retrieve <cnt> resources and their latest <cin> resources from step 2 and 4
6) retrieve <smd> resources, which uses specific ontologyRef, as children of the containers from step 2 and 4
7) get the latest <smd> resource from step 6 and decode base 64 to get RDF/XML data

Figure 6.4.1-1: Semantic Interworking Procedures (1/2)

Step 1) After creating annotations as <*semanticDescriptor*> resources in oneM2M platform, a semantic proxy application for interworking performs the oneM2M semantic discovery to find NGSI-LD Entity corresponding resources. The semantic discovery includes the specific SPARQL query:

```
PREFIX annotation: <http://www.citydatahub.kr/ontologies/2021/1/sm_annotation#>
PREFIX ngsi-ld: <http://uri.etsi.org/ngsi-ld/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT DISTINCT ?entity
WHERE {
  ?entity  rdf:type ngsi-ld:Entity.
  ?entity annotation:hasDomainType  ?entityType
}
```

Step 2) By the assumption in clause 6.3.1, when a resource representing NGSI-LD Entity is discovered, the descendant resources would represent NGSI-LD Attributes (i.e. Property or Relationship). The semantic discovery sample result shows that sub-resources of the "base/sem_iwk/spot_0001" would comprise an NGSI-LD Entity instance.

```
{
    "m2m:uril": [
        "base/sem_iwk/spot_001"
    ]
}
```

Steps 3) to 4): The proxy discovers all child <*container*> resources of the one representing NGSI-LD Entity.

Step 5) Retrieve all <*container*> resources from steps 2) and 4), also retrieves all <*latest*> resources of the <*container*> resources.

Step 6) From the <*container*> resources, discover all <*semanticDescriptor*> resources that has the interested ontology URI by the ontologyRef (or) attribute.

Step 7) Retrieve the each latest <*semanticDescriptor*> resource of the <*container*> resources. Since there is no feature like <*latest*> for <*semanticDescriptor*> resources, ***Result Content*** (rcn) request parameter can be used to retrieve multiple child resources so creationTime (ct) can be assessed to find the latest one by the proxy application.

6.4.2 Resource synchronization

Once initial NGSI-LD entity instances(resources) creation gets done, newly updated resources on oneM2M platform side is reflected in NGSI-LD context broker. oneM2M subscription/notification is leveraged by the semantic proxy in this case.

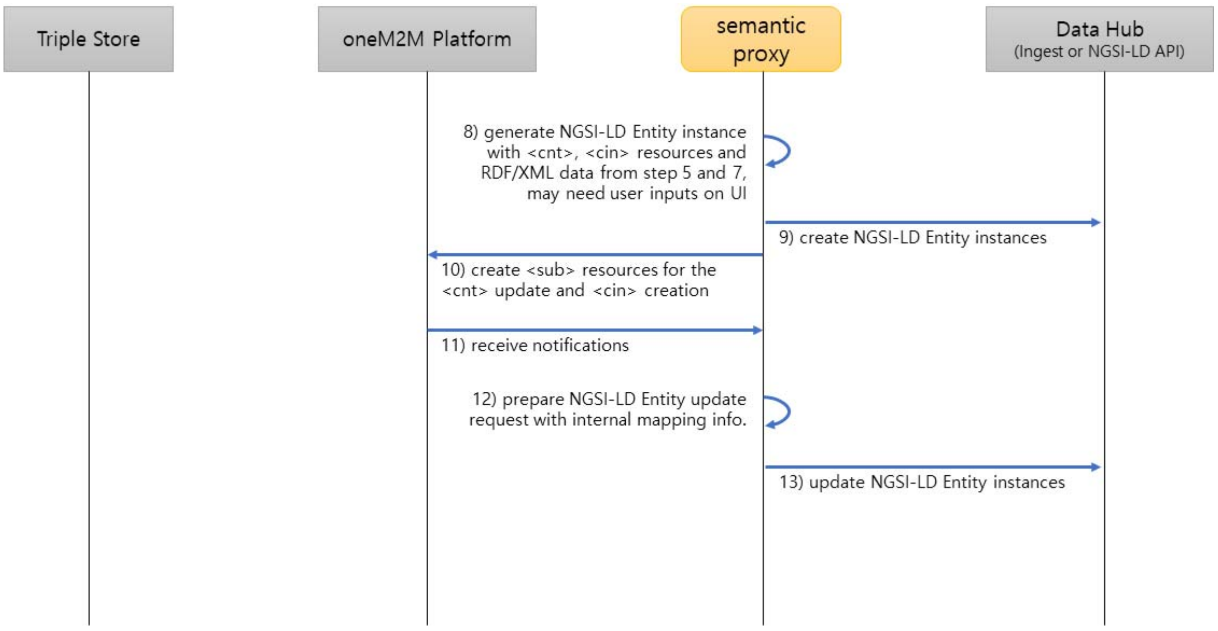

Triple Store
oneM2M Platform
semantic proxy
Data Hub
(Ingest or NGSI-LD API)
8) generate NGSI-LD Entity instance with <cnt>, <cin> resources and RDF/XML data from step 5 and 7, may need user inputs on UI
9) create NGSI-LD Entity instances
10) create <sub> resources for the <cnt> update and <cin> creation
11) receive notifications
12) prepare NGSI-LD Entity update request with internal mapping info.
13) update NGSI-LD Entity instances

Figure 6.4.2-1: Semantic Interworking Procedures (2/2)

Step 8)	After retrieving all <*container*>, <*contentInstance*> and <*semanticDescriptor*> resources, which means all IoT service data and mapping rules, the proxy converts oneM2M resources into NGSI-LD Entity instances.
Step 9)	The semantic interworking proxy creates NGSI-LD Entity instances.
Step 10)	The semantic interworking proxy creates <*subscription*> resources to get any updates on <*container*> resources and <*contentInstance*> resources.
Step 11)	The semantic interworking proxy receives notifications.
Step 12)	The proxy converts the notification event to the corresponding NGSI-LD instance updates.
Step 13)	The proxy sends updates requests to NGSI-LD context broker for the existing Entity instances.

6.5 Historical data synchronization procedures

Clause 6.4 illustrates the procedure to get the latest semantically annotated oneM2M data instances, in contentInstance resource type, and synchronize them into NGSI-LD systems as NGSI-LD Entities. Since a <contentInstance> resource is immutable, every data instance is stored as a new <contentInstance> resource. In this sense, for a data interworking proxy, to fetch historical data instances from a oneM2M platform, the proxy needs to retrieve multiple <contentInstance> resources.

[Step 5 modification in clause 6.4.1 - replacing single latest instance retrieval into historical instances fetch]

To retrieve multiple resources with specific conditions (e.g. time period), the ***Result Content*** and the ***Filter Criteria*** request parameter can be used. Among the allowed ***Result Content*** parameter values, to get <contentInstance> resources, "child resources" can be used. To get instances for an interested period of time, the *createdAfter* (cra) and the *createdBefore* (crb) conditions of the ***Filter Criteria*** parameter can be set. The conditions match with the *creationTime* (ct), which is one of the universal attributes in oneM2M, of <contentInstance> resources. Also, there can be other resources in different types than *contentInstance*, the *resourceType* (ty) condition also needs to be set as *contentInstance* (enum value "4") in the ***Filter Criteria***.

NOTE: Historical data retrieval request with the number of instances is being standardized in oneM2M Release 5. When it is published, a corresponding procedure can also be illustrated.

[Step 6 extension in clause 6.4.1 – retrieving semanticDescriptors of individual data instance]

The assumption made in clause 6.4 is that <contentInstance> children resources of a <container> resource get annotated by a single <semanticDescriptor> resource, which is also the child of the <container> resource. However, by the oneM2M specification [i.7], each <contentInstance> resource can have its own <semanticDescriptor> resource. Depending on use cases, new instances could have different annotations. To include this varying annotation during the historical data synchronization, semantic descriptors of content instances also need to be retrieved.

To check whether there is a <semanticDescriptor> resource as a child of a <contentInstance> resource, the interworking proxy can perform a oneM2M resource discovery targeting a container and resourceType as semanticDescriptor (eum value "24") and resource tree depth or level (lvl) as "2".

For example, when there is a parking spot container named "spot_001" and it has content instances with resource name prefix "cin_", an example discovery result seems like the below. "smd_abexd" is the resourceName (rn) of the <semanticDescriptor> resource that annotates the <contentInstance> resource named "cin_0033".

```
{
    "m2m:uril": [
        "base/parkingLot1/spot_001/cin_0033/smd_abexd"
    ]
}
```

7 Solution 3 - Annotation ontology based static interworking

7.1 Introduction

This solution for mapping NGSI-LD to oneM2M is summarized as follows: every NGSI-LD entity is mapped to a different <*container*> resources within the same oneM2M <*ae*> resource. For this purpose, NGSI-LD Property values or Relationship object are mapped directly to oneM2M resources of type <*contentInstance*>. The type (i.e. "Property" or "Relationship") is added as a labels in the semantic descriptor for the <*container*> resource. Each NGSI-LD entity is mapped to a top-level <*container*> resource.

This solution then exploits <*container*> resource nesting, hence using child <*container*> resources to represent Properties or Relationships, so that each NGSI-LD Entity is considered as a top-level <*container*> and be created, together with all other entities, inside the networked oneM2M <*ae*> resource where they belong.

What <*ae*> resource is used in a CSE of oneM2M system for interworking with NGSI-LD is configured in IPE internally from oneM2M system to NGSI-LD system.

What entity is used for interworking with oneM2M is configured in IPE internally from NGSI-LD system to oneM2M system.

This solution has been developed and used in FED4IoT [i.3] project.

7.2 oneM2M features

This solution uses labels attribute, one of the common attributes in all resource types. Normally in oneM2M labels attribute is used for discovery purposes like hash tag, but this solution uses for storing several meta information.

Labels attribute is a list of individual labels, and each of them is either a standalone label-key, used as a simple "tag", that can be used for example for discovery purposes when looking for particular resources that one can "tag" using that label-key or a composite element made of a label-key and a label-value, separated by a special character defined in ETSI TS 118 101 [i.1].

Also, this solution uses <*semanticDescriptor*> resource type to specify what ontology is used.

7.3 Resource mapping

7.3.0 Introduction

Table 7.3.0-1 specifies what labels key is used for this interwoking solution.

Table 7.3.0-1: Labels used in the interworking solution

labels key name	data type	limitation	multiplicity	description
type	String		0..1	NGSI-LD entity type in labels of top-level <*container*> resource Property/Relationship type in labels of child <*container*> resource

7.3.1 Resource Mapping from NGSI-LD to oneM2M

It groups all NGSI-LD Entities under one common oneM2M <*ae*> resource. This means that top-level <*container*> of the <*ae*> resource now represent NGSI-LD Entities, not Properties/Relationships. It then exploits <*container*> resource nesting, hence using child <*container*> resources to represent Properties or Relationships, so that each NGSI-LD Entity is considered as a top-level <*container*> resource and be created, together with all other entities, inside the oneM2M <*ae*> resource where they belong.

Suppose there is the following NGSI-LD Entity, that comprises three Properties, where @context is omitted:

```
{
    "id": "urn:ngsi-ld:Sensor:AparcamientoBicis:180",
    "type": "Sensor",
    "description": {
        "type": "Property",
        "value": "Santo%20Domingo"
    },
    "location": {
        "type": "GeoProperty",
        "value": "{\"type\":\"Point\", \"coordinates\":[37.987769,-1.129766]}"
    },
    "freeSpots": {
        "type": "Property",
        "value": "16"
    }
}
```

oneM2M is extremely flexible in the structuring of data sources. Nonetheless, different resource types come with different levels of overhead when creating them. For instance, at creation time, Application Entities need to explicitly register themselves to the higher-level resource that coordinates them, i.e. the Common Services Entity (CSE). This is because Application Entities in oneM2M capture the concept of data producers (and consumers), rather than the concept of the produced data items. Furthermore, each AE can be the originator of a request to be granted a set of access operations (such as CREATE, RETRIEVE, UPDATE, DELETE, DISCOVER and NOTIFY), giving birth to a complex system of access control policies for those Application Entities that are going to operate on certain <*container*> resources. Table 7.3.1-1 is the guidelines for mapping between NGSI-LD and oneM2M.

Table 7.3.1-1: Guidelines for mapping between NGSI-LD and oneM2M

NGSI-LD	oneM2M
entity	top-level <*container*> resource
entity id	resourceID of top-level <*container*> resource
entity type	labels of top-level <*container*> resource
property/relationship	child <*container*> resource
property/relationship name	resourceName of child <*container*> resource
property/relationship type	labels of child <*container*> resource
property value/relationship object	<*contentInstance*> of the child <*container*> resource
sub-property/sub-relationship	grand child <*container*> resource (<*container*> resource nested within child <*container*> resource)

Concerning values of the NGSI-LD properties, they are always going to be mapped to oneM2M resources of type <*contentInstance*>. <*contentInstance*> resources make no assumptions regarding their content, and they behave as blackboxes, so that not much is done on their content, except retrieving it. Mostly the content is base64-encoded. This is the reason why the idea is to put the whole property into the <*contentInstance*> resource, including types, so that it can be unboxed in a straightforward way, without loosing any information, for example whenever there is a need to reverse map from oneM2M back to NGSI-LD. Furthermore, the type is mapped onto the labels of <*container*> resource, in order to enable discovery by means of queries/filters on the labels' values. Similarly, oneM2M <contentInstances> do not express meta-information about the instance's units or geographic position, essentially treating the data as opaque. If the system (or data producer) wants to specify complex metadata in oneM2M resources, then the <*semanticDescriptor*> resource type is available. It enables attaching RDF triples to the resources, and the oneM2M specification describes how to build queries and semantic filters to discover resources based on <*semanticDescriptor*> resources. The ontologyRef attribute of <*semanticDescriptor*> is used for specifying what ontology is used as the basis for the semantic content of <semanticDescriptor>.

In the example that follows, it is shown how to exploit <*semanticDescriptor*> of the child <*container*> resource to capture the fact that the property of the original NGSI-LD entity concerns its geographic location. As the example NGSI-LD entity presented above, it contains information coming from the Murcia bike parking place, and apply the mapping guidelines to it:

- An Application Entity is created and registered to the CSE of the Silo broker. The AE resource name may be equal to the vThingID, e.g. "BikeParking". The AE is the contact point for retrieving information about all bike parking places of Murcia. This strategy avoids creating a huge number of AE resources.
- "id":"AparcamientoBicis:180" (which is a specific parking place) → becomes a top-level <*container*> within the AE (i.e. every NGSI-LD Entity is mapped to a different top-level <*container*> resource).
- "type":"Sensor" → becomes a "labels" of the top-level <*container*> resource.
- "freeSpots" property → spawns a child <*container*> resource. It becomes name of the child <*container*> resource.
- "type":"Property" of "freeSpots" → becomes labels of the "freeSpots" child <*container*> resource.
- "value":"16" → spawns a <*contentInstance*> of the "freeSpots" child <*container*> resource.
- "location" → another child <*container*> resource.
- it is inferred that "location" refers to a geographic position, hence a semantic descriptor of the "location" child <*container*> resource is created. RDF triples are inserted stating the fact that the data of the <*container*> resource concerns geographic positions or geometries.
- "description" → another child <*container*> resource.

Also, the other values become content value of <*contentInstance*> resources associated with their respective child <*container*> resources. It has to be noted that the whole JSON block of the "location" Property, shown below, becomes the content of <*contentInstance*>, so that no information is lost. This holds true for all properties/relationships in the original NGSI-LD Entity.

```
{
    "type": "GeoProperty",
    "value": "{\"type\":\"Point\", \"coordinates\":[37.987769,-1.129766]}"
}
```

Basically, this approach can be summarized as: every Property/Relationship of the NGSI-LD Entity becomes a child resource of <*container*>; the values become content of <*contentInstance*> resources, and the Entity itself becomes the top-level <*container*> resource.

It is possible to do query and discovery, by exploiting the fact that the names are copied to the labels. Whenever the mapping algorithm is able to infer complex data types, semantic descriptors can additionally be created.

Figure 7.3.1-1 (copied from [i.3], "libres" means "freeSpots" and "descripcion" means "description") represents that mapping as resource tree, having one AE for all bikeparkings and assuming that the CSE has name "Murcia", i.e. the tenant identifier. <*container*> resources in the same level could allow to query all the bikeparkings at once. This perspective is interesting from the Murcia city point of view. However, it is not possible to *a priori* exclude that bikeparking managers may prefer to have an alternative with one AE per bikeparking.

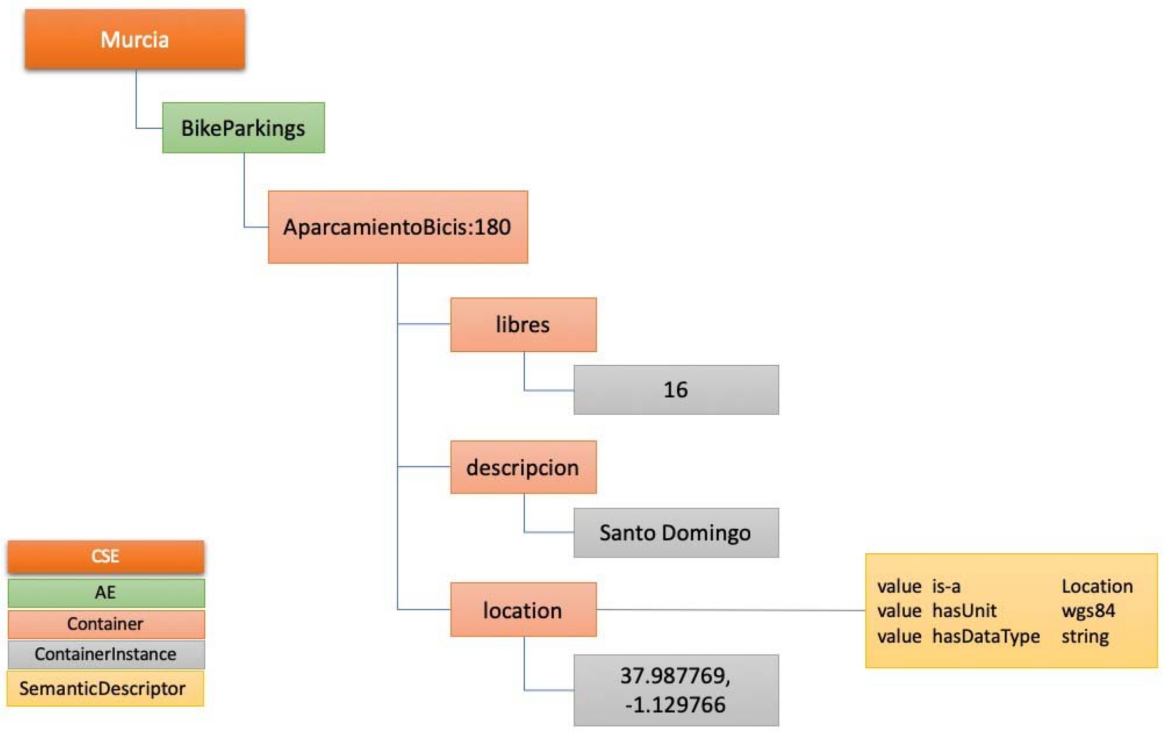


Figure 7.3.1-1: oneM2M instantiation of the NGSI-LD bike parking entity (copied from [i.3])

7.3.2 Resource Mapping from oneM2M to NGSI-LD

Here a "smart camera" use-case is used (real-life example from camera located in the city of Grasse) in order to briefly present the strategy for mapping from NGSI-LD to oneM2M and the reverse mapping, i.e. from oneM2M to NGSI-LD. The scenario is that of a sensitive site monitored by a smart camera, which may be able to detect events of wild deposits and provide the location, the car plate number and the type of waste illegally deposited.

The NGSI-LD data model is the starting point for mapping NGSI-LD to oneM2M. Figure 7.3.2-1 models this example in NGSI-LD.

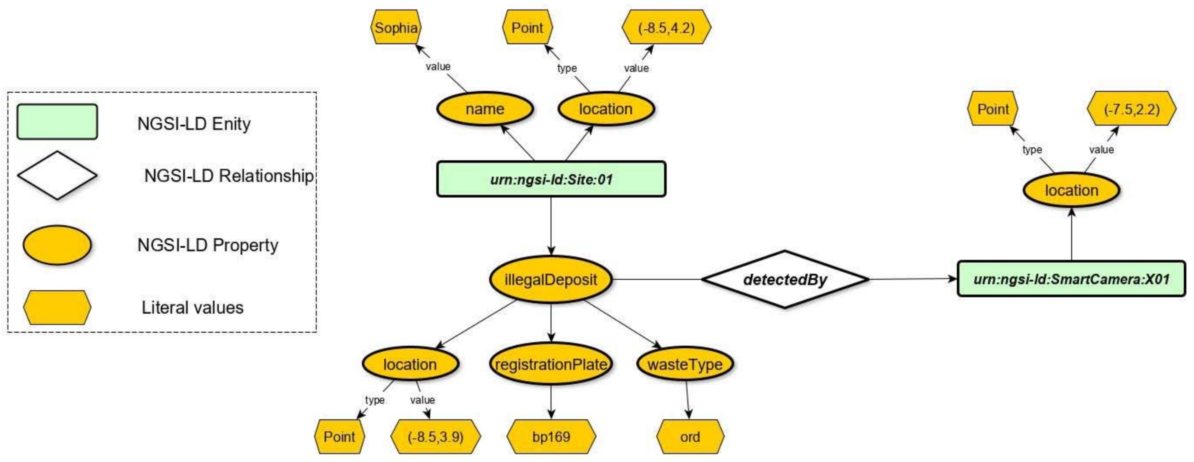


Figure 7.3.2-1: Grasse Use Case Example modelled in NGSI-LD

Below, the NGSI-LD entities of Site and Smart Camera. The @context is omitted.

```
{
    "id": "urn:ngsi-ld:Site:01",
    "type": "Site",
    "location": {
        "type": "GeoProperty",
        "value": {
            "type": "Point",
            "coordinates": [-8.5, 4.2]
        }
    },
    "name": {
        "type": "Property",
        "value": "Sophia"
    },
    "illegalDeposit": {
        "type": "Property",
        "location": {
            "type": "GeoProperty",
            "value": {
                "type": "Point",
                "coordinates": [-8.5, 3.9]
            }
        },
        "registrationPlate": {
            "type": "Property",
            "value": "bp169"
        },
        "wasteType": {
            "type": "Property",
            "value": "ORD"
        },
        "detectedBy": {
            "type": "Relationship",
            "object": "urn:ngsi-ld:SmartCamera:X01"
        }
    }
}
```

```
{
    "id": "urn:ngsi-ld:SmartCamera:X01",
    "type": "SmartCamera",
    "location": {
        "type": "GeoProperty",
        "value": {
            "type": "Point",
            "coordinates": [-7.5, 2.2]
        }
    }
}
```

The starting point for mapping oneM2M to NGSI-LD is the oneM2M resource tree, which is detailed in Figure 7.3.2-2.

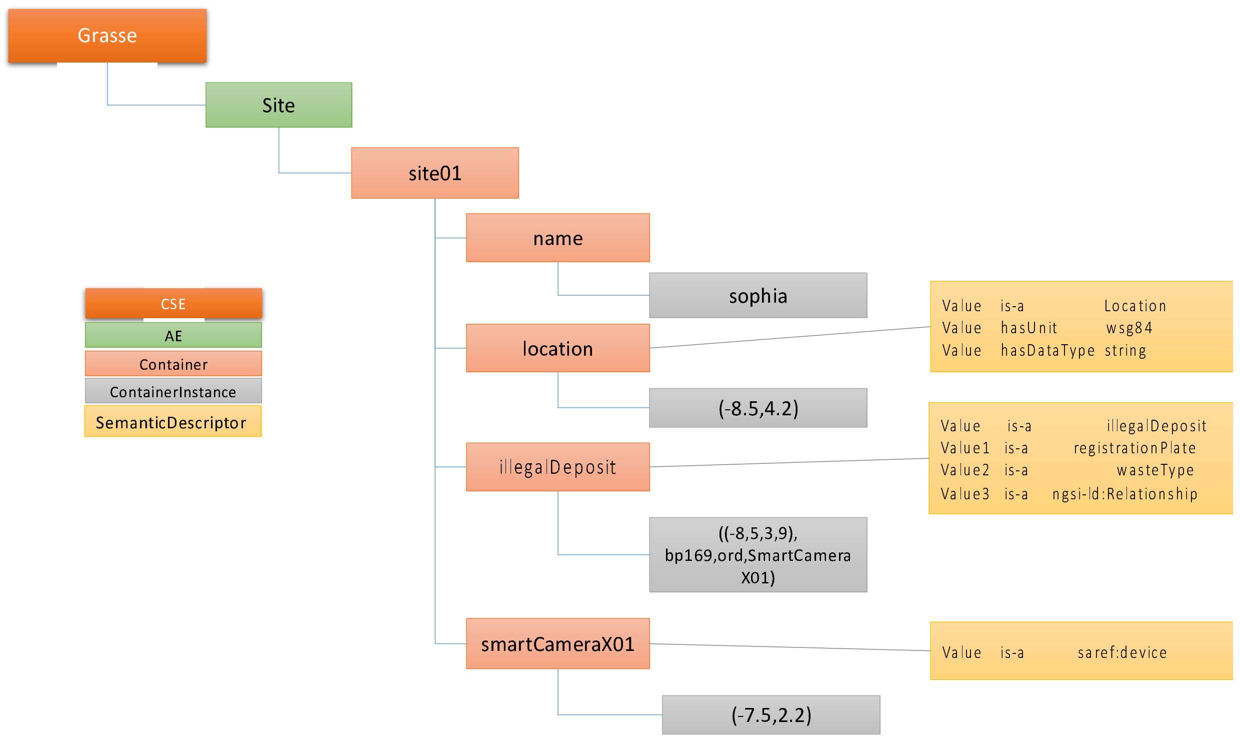


Figure 7.3.2-2: oneM2M resource tree of the Grasse Use Case Example

The <semanticDescriptor>, part of the oneM2M standard specification, enables attaching semantic description via RDF triples to the resource. These added triples are exploited for mapping oneM2M to NGSI-LD. For this purpose, this clause will focus on the impact of the semantic description on the mapping of oneM2M to NGSI-LD according to 2 main categories:

1) Mapping oneM2M to NGSI-LD allowing to choose the reference ontology; and

2) Mapping oneM2M to NGSI-LD with a specific standardized reference ontology (e.g. NGSI-LD and SAREF).

Mapping oneM2M to NGSI-LD allowing for user to choose the reference ontology: In this case it is supposed that all <*ae*> resources, <*container*> resources and child <*container*> resources have an URI pointing to their semantic description. These URIs are directly added within the @context file, as a preliminary necessary step towards achieving a well-formed mapping. Table 7.3.2-1 summarizes the further steps of the mapping strategy from oneM2M to NGSI-LD.

Table 7.3.2-1: First steps towards mapping oneM2M to NGSI-LD

oneM2M	NGSI-LD
top-level <*container*> resource	Entity
<*ae*> resource	Entity type
resourceID of top-level <*container*> resource	Entity Id = "urn:ngsi-ld:" + AE + ":" + resource ID
resourceName of child <*container*> resource	Property name
<*contentInstance*> of the child <*container*> resource	Property value

Applying this strategy in our use case example, three main problems appear:

1) Properties (location, registrationPlate, wasteType) of the Property "illegalDeposit" are all mapped to a unique value as follows:

```
"illegalDeposit": {
    "type": "Property",
    "value": "((-8,5,3,9), bp169,ord,SmartCameraX01)"
}
```

Properties of Property and Relationships of Property are not supported using this strategy.

2) Relationships following this mapping strategy are not detectable, all child <*container*> resources of the resource tree are seen as NGSI-LD properties and <*contentInstance*> resource of the child <*container*> resources as property values.

3) The child <*container*> resource "smartCameraX01" has to be mapped to a new NGSI-LD entity not to a property.

Mapping oneM2M to NGSI-LD with a specific reference ontology (NGSI-LD and SAREF): In order to produce a correct mapping strategy, there is a need to refer to some known ontologies and critically to SAREF and NGSI-LD ontologies. Referring these ontologies in the semantic descriptor and adding new generic rules will allow a successful oneM2M to NGSI-LD mapping strategy.

In the following, the previously detected issues are solved by adapting the semantic description and referring NGSI-LD and SAREF ontologies.

- Rules for detecting Properties of Property

If a child <*container*> resource is described as an RDF class or NGSI-LD Entity that have properties as domain, then the assumption that this child <*container*> resource will be mapped as an NGSI-LD property, its semantic properties as NGSI-LD properties of the property and content values of the <*contentInstance*> resources as values of the properties, can directly be used.

The same strategy may be applied for detecting NGSI-LD Relationships of Property, especially when the following rules for detecting NGSI-LD Relationships is applied.

- Rules for detecting Relationships

Each desired Relationship in the oneM2M resource tree has to be defined and specified as a NGSI-LD Relationship in the semantic descriptor by referencing the URI of the NGSI-LD relationship " https://uri.etsi.org/ngsi-ld/Relationship".

The Relationship concept is defined and introduced by the NGSI-LD standard, and detecting it in the oneM2M resource tree or even in the semantic description is hard task if the NGSI-LD URI is not referenced. When an NGSI-LD Relationship is detected, its value will be added to the "object" field and a new Entity will be created. Properties of this newly created Entity are not known and have to be brought in. That is why a child <*container*> resource with the properties of the Smart camera was added.

- Rules for detecting new Entities

Each saref:device is mapped to a new NGSI-LD Entity where its type is the labels of the child <*container*> resource, the Id is the resourceID of the child <*container*> resource. <*contentInstance*> resources of the child <*container*> resource are mapped as properties of the new Entity.

The SAREF ontology was mentioned in the previous clause as a solution for detecting new entities in the oneM2M resource tree and thus mapping all SAREF properties of the semantic description to NGSI-LD properties. As mentioned before, Relationships have to be declared, in order for them to be detectable and thus mapped to NGSI-LD Relationship correctly.

7.4 New data synchronization procedures from NGSI-LD to oneM2M

7.4.1 Pre-configuration

There is nothing to configure before the procedure.

7.4.2 IPE initialization

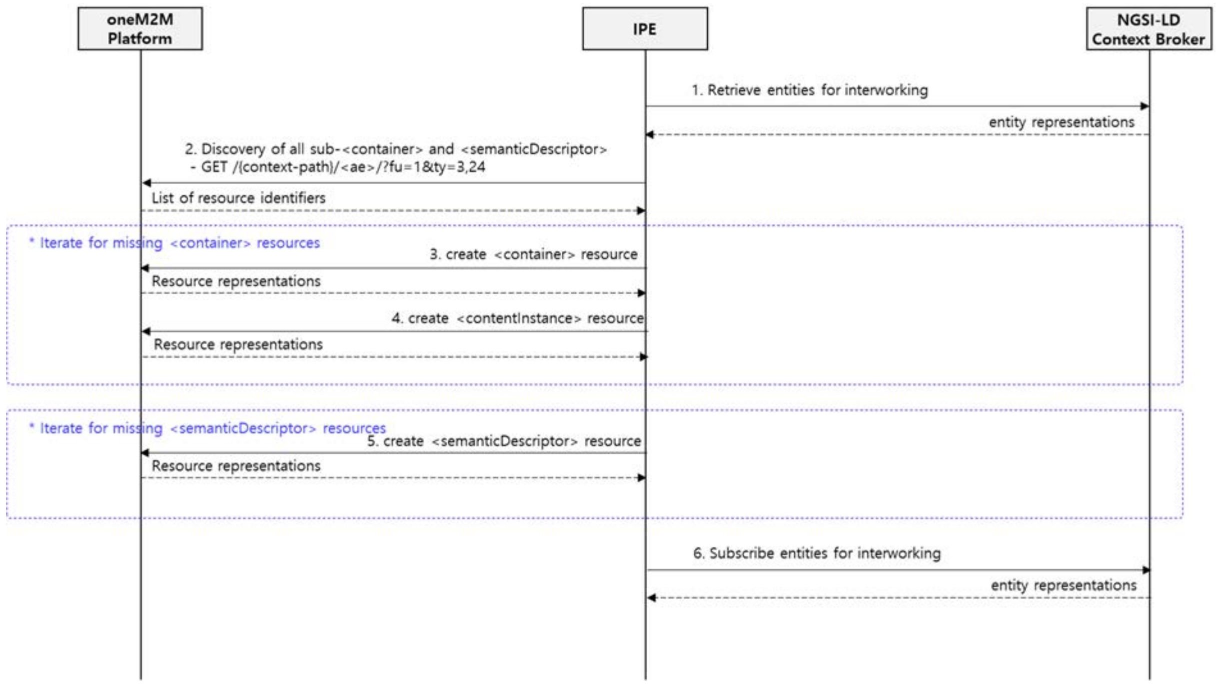

oneM2M Platform
IPE
NGSI-LD Context Broker
1. Retrieve entities for interworking
entity representations
2. Discovery of all sub-<container> and <semanticDescriptor>
- GET /{context-path}/<ae>/?fu=1&ty=3,24
List of resource identifiers
* Iterate for missing <container> resources
3. create <container> resource
Resource representations
4. create <contentInstance> resource
Resource representations
* Iterate for missing <semanticDescriptor> resources
5. create <semanticDescriptor> resource
Resource representations
6. Subscribe entities for interworking
entity representations

Figure 7.4.2-1: Procedures for IPE initialization

Step 1. Retrieve entities for interworking from NGSI-LD context broker. IPE already has knowledge for entities for interworking.

Step 2. Discover all child <*container*> resources and <*semanticDescriptor*> resources for target <*ae*> resource.

Step 3. If there is no resource for entity or attribute of entity, IPE creates <*container*> resource for the entity or the attribute of the entity as specified in clause 7.3.1.

Step 4. If IPE creates <*container*> resource in step 3., IPE also creates <*contentInstance*> resource under the <*container*> resource as specified in clause 7.3.1.

Step 5. If there is missing <*semanticDescriptor*> resource, IPE creates <*semanticDescriptor*> resource according to clause 7.3.1.

Step 6. IPE subscribes the entities for interworking.

7.4.3 Resource synchronization

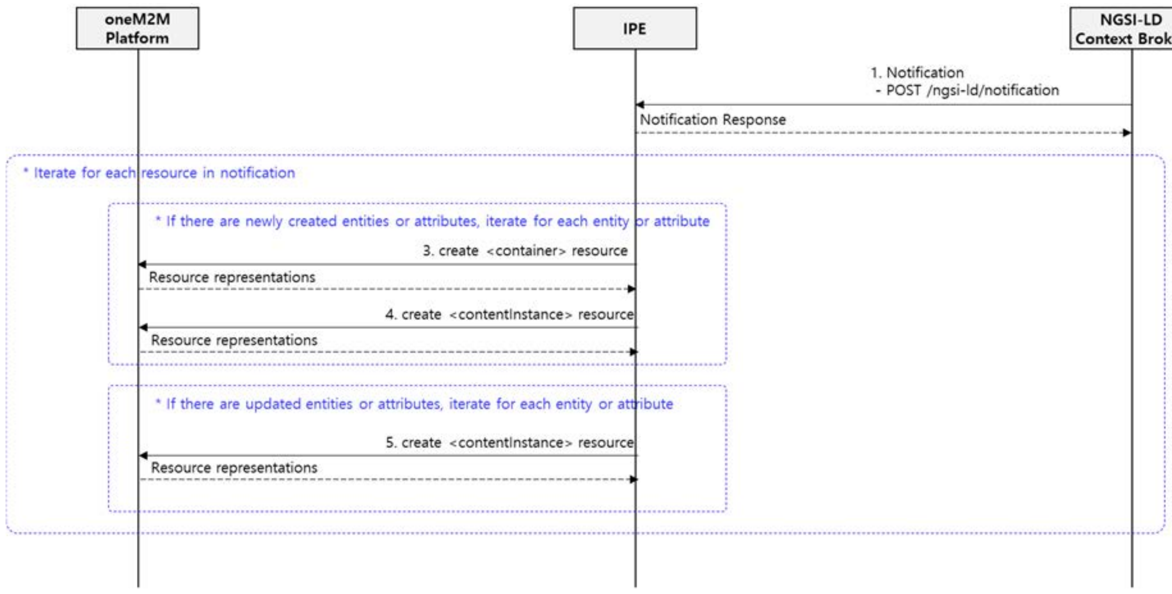

oneM2M Platform
IPE
NGSI-LD Context Broker
1. Notification
- POST /ngsi-ld/notification
Notification Response
* Iterate for each resource in notification
* If there are newly created entities or attributes, iterate for each entity or attribute
3. create <container> resource
Resource representations
4. create <contentInstance> resource
Resource representations
* If there are updated entities or attributes, iterate for each entity or attribute
5. create <contentInstance> resource
Resource representations

Figure 7.4.3-1: Procedures for Interworking Data

Step 1. IPE receives a notification requests per subscription.

Step 2. Check if notification is sent due to the <*subscription*> resource created during IPE initialization. If not, not processed further.

Step 3. If entity or attribute in the resource is newly created, create <*container*> resource as specified in clause 7.3.1.

Step 4. If <*container*> resource is created in step 3., create <*contentInstance*> resource as specified in clause 7.3.1.

Step 5. If entity or attribute in the resource is updated, create <*contentInstance*> resource according to clause 7.3.1.

7.5 New data synchronization procedures from oneM2M to NGSI-LD

7.5.1 Pre-configuration

A NGSI-LD system user configures <*semanticDescriptor*> resource for oneM2M resources that need interworking to NGSI-LD.

7.5.2 IPE initialization

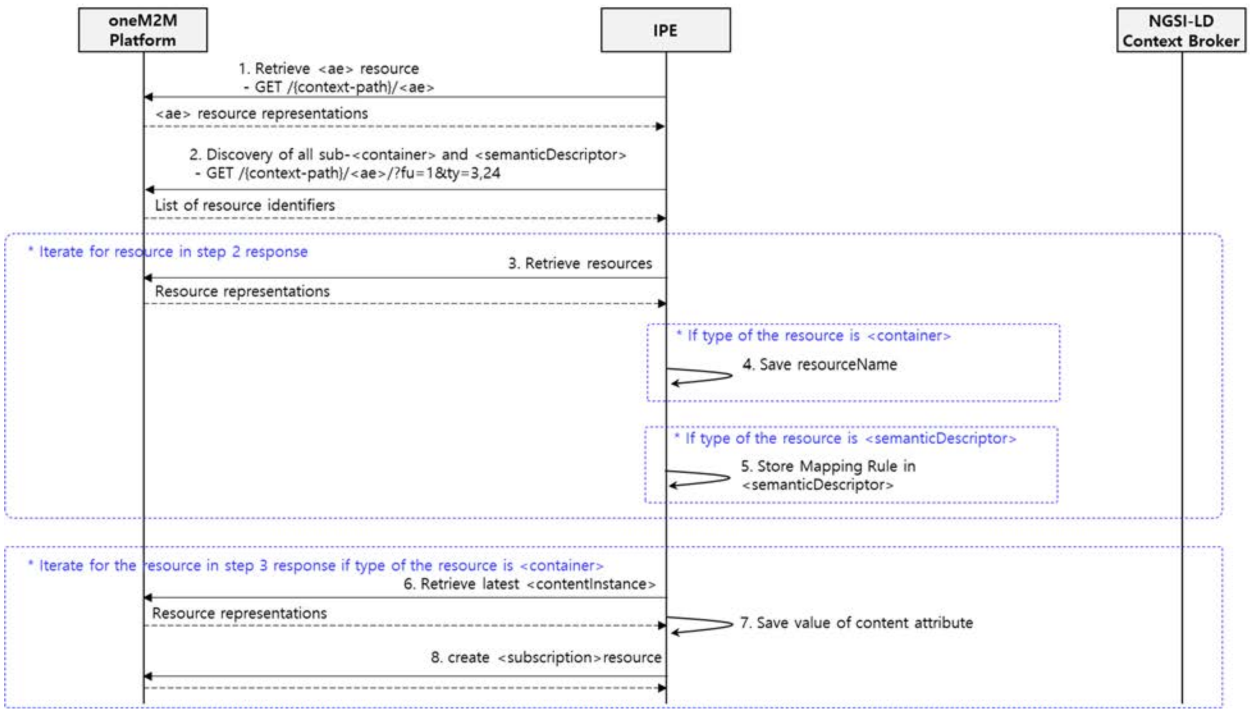

oneM2M Platform
IPE
NGSI-LD Context Broker
1. Retrieve <ae> resource - GET /{context-path}/<ae>
<ae> resource representations
2. Discovery of all sub-<container> and <semanticDescriptor> - GET /{context-path}/<ae>/?fu=1&ty=3,24
List of resource identifiers
* Iterate for resource in step 2 response
3. Retrieve resources
Resource representations
* If type of the resource is <container>
4. Save resourceName
* If type of the resource is <semanticDescriptor>
5. Store Mapping Rule in <semanticDescriptor>
* Iterate for the resource in step 3 response if type of the resource is <container>
6. Retrieve latest <contentInstance>
Resource representations
7. Save value of content attribute
8. create <subscription>resource

Figure 7.5.2-1: Procedures for IPE initialization

Step 1.	Retrieve <*ae*> resource. This solution assumes that IPE already has knowledge what <*ae*> resources are need interworking.
Step 2.	Retrieve list of resource identifier for child <*container*> resources and child <*semanticDescriptor*> resources.
Step 3.	Retrieve resource in step 2. response.
Step 4.	If type of the resource is <*container*> in step 3. response, IPE saves resourceName of the resource.
Step 5.	If type of the resource is <*semanticDescriptor*> in step 3. response, IPE saves mapping rule defined in <*semanticDescriptor*> resource.
Step 6.	Retrieve latest <*contentInstance*> if the resource is <*container*> resource type in step 3. response.
Step 7.	Save value of content attribute.
Step 8.	Create <*subscription*> resources if the resource is <*container*> resource type in step 3. response.

7.5.3 Resource synchronization

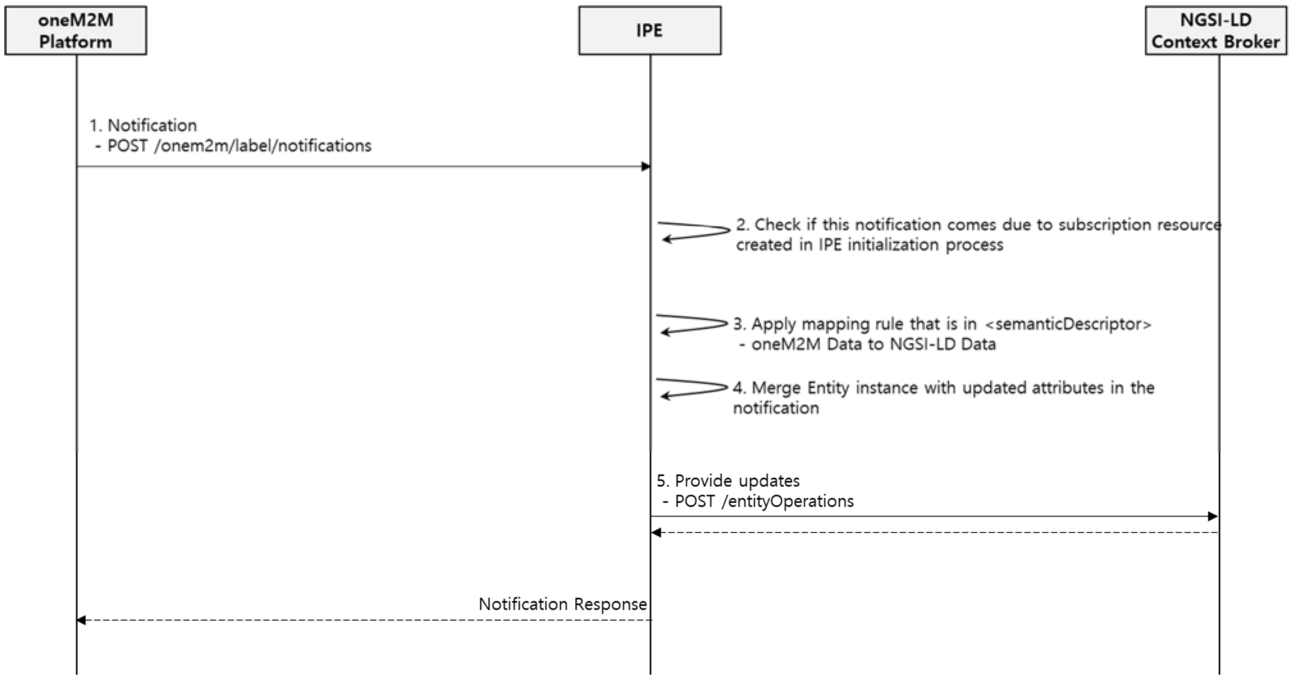

oneM2M Platform
IPE
NGSI-LD Context Broker
1. Notification
- POST /onem2m/label/notifications
2. Check if this notification comes due to subscription resource created in IPE initialization process
3. Apply mapping rule that is in <semanticDescriptor>
- oneM2M Data to NGSI-LD Data
4. Merge Entity instance with updated attributes in the notification
5. Provide updates
- POST /entityOperations
Notification Response

Figure 7.5.3-1: Procedures for Interworking Data

Step 1.	IPE receives a notification requests per subscription.
Step 2.	Check if notification is sent due to the <*subscription*> resource created during IPE initialization. If not, not processed further.
Step 3.	If there is <*semanticDescriptor*> for the resource in the notification, apply mapping rule as specified in clause 7.3.2.
Step 4.	Merge NGSI-LD Entity instance with the updated one(s) from step 3. as specified in clause 7.3.2.
Step 5.	Provide updates to the NGSI-LD Entity instance via Batch Entity Creation of Update (Upsert) interface.

8 Make use of OAuth token to synchronize data from oneM2M to NGSI-LD

8.1 Introduction

oneM2M defines multiple access control mechanisms and the flow of applying those different ones in clause 10.2.3.1 of [i.7]. Each authorization procedure is specified in in the oneM2M security specification [i.8].

This clause describes OAuth token-based access control to retrieve oneM2M resources for data synchronization from a oneM2M platform to an NGSI-LD broker. Therefore, the authorization mechanism in this clause can be used with the solutions in the previous clauses.

8.2 Issuing an OAuth token

oneM2M adopts JSON Web Token (JWT) which can be represented in either JSON Web Signature (JWS), or JSON Web Encryption (JWE). The oneM2M security specification defines the procedures to issue and use the token with the Dynamic Authorization System (DAS). Unlikely the original <accessControlPolicy> resource based access control handling procedures, which is a more static authorization, DAS lets access privileges given by the resource owners be managed outside oneM2M platforms.

In direct Dynamic Authorization procedure (clause 7.3.2.2 of [i.8]), a token(s) can be issued by a DAS server when it is requested by the Hosting CSE for a dynamic authorization. After the access decision by the Hosting CSE, within the response the issued token is sent to the Originator so it can be used for subsequent requests.

In indirect Dynamic Authorization procedure (clause 7.3.2.3 of [i.8]), an Originator can request to issue tokens to DAS server. The DAS server can send token IDs or optionally encrypted tokens to the Originator in the response. For oneM2M request to a Hosting CSE, if a token ID is used, the Hosting CSE retrieves the token from the DAS Server for access decision.

The interface between AE as an Originator and the DAS server is not the scope of oneM2M standard. This means an AE can use an OAuth token which is compatible with the JWT token structure in [i.8]. When the token, not the token ID, is carried in Tokens request parameter in a oneM2M primitive, it can be used like the token issued from the DAS server.

8.3 Token based access control on oneM2M platform

The structure of an access token in oneM2M, which is compliant to the JWT, is defined in Table 8.3-1. In case of a token issued by a DAS Server, the issuer identifies the DAS server. The holder specifies the Originator that uses the token in a oneM2M request, while audience indicates the list of CSEs who receives a request with a JWT token and perform an authorization. The permission information can have the access privileges or role IDs with targeted resource IDs. The privilege data structure is the one used in the accessControlPolicy resource type.

Table 8.3-2: Token structure

Token Claimset Object Element Path	oneM2M JWT claim name	Description
version	"tkvr"	version of the token
tokenID	"jti"	unique ID of the token
issuer	"iss"	ID of the token issuer
holder	"azp"	Originator AE-ID or CSE-ID
notBefore	"nbf"	token is valid since this time
notAfter	"exp"	token is not valid since this time
tokenName	"tknm"	human readable token name
audience	"aud"	CSE-IDs that uses this token for authorization
permissions	"tkps"	permissions pertaining to this token which can consist of privileges, target resource IDs and role-IDs (clause 9.6.39 of [i.7])
extension	"tkex"	other information

When a Hosting CSE receives a request with a token with the specified structure, it checks the token validity (e.g. Originator is equal to the holder) and make the access decision with the permissions information.

9 Conclusions and future considerations

The present document introduces three candidate interworking solutions for data synchronization between oneM2M system and NGSI-LD system. It is not intended to mandate any of these solutions, these are illustrated as they have been developed and adapted in several projects as proof-of-concept.

Since the scope of the present document is limited to data synchronization, further study is needed for other interworking scenarios such as resource discovery from one to the other, data provision or consumption from one to the other.

Table 9-1: Comparison of the data synchronization solutions

Features	Label based dynamic interworking	Mapping ontology based dynamic interworking	Annotation ontology based static interworking
Interworking direction	oneM2M to NGSI-LD	oneM2M to NGSI-LD	oneM2M to NGSI-LD, NGSI-LD to oneM2M
New Data Synchronization method	Subscription/Notification	Subscription/Notification	Subscription/Notification
Historical Data Synchronization Support	Yes	Yes	No
oneM2M version compatibility	Since Rel-1	Since Rel-2	Since Rel-2
Applicability of oneM2M resource types	<*container*>, <*contentInstance*>, <*flexContainer*>, <*flexContainerInstance*>, <*timeSeries*>, <*timeSeriesInstance*>	<*container*>, <*contentInstance*>, <*flexContainer*>, <*flexContainerInstance*>, <*timeSeries*>, <*timeSeriesInstance*>	<*container*>, <*contentInstance*>
Granularity of oneM2M attribute mapping	An attribute or an element of an attribute	An attribute or an element of an attribute	An attribute or an element of an attribute
Flexibility of resource mapping rules	Dynamic mapping based on mapping rule in labels	Dynamic mapping based on mapping rule in semantic descriptor	Static mapping
Mapping rules location	labels attribute	<*semanticDescriptor*> resource	labels attribute, <*semanticDescriptor*> resource
Semantic Ontology	n/a	A new ontology for oneM2M and NGSI-LD resource mapping to store mapping rules in <*semanticDescriptor*> resources.	NGSI-LD and SAREF to annotate the oneM2M resource.
Dependency of oneM2M resource tree structure	Independent. Any resources, even in different sub-trees, can compose an NGSI-LD Entity.	Dependent. A set of resources in the same tree compose an NGSI-LD Entity.	Dependent. A set of resources in the same tree compose an NGSI-LD Entity.

Annex A:
Change history

Date	Version	Information about changes
September 2023	1.2.1	CIM(23)025012r1 add oneM2M interworking using OAuth token CIM(23)025011 add procedure for oneM2M interworking historical data with semanticDescriptor
December 2023	1.2.1	CIM(23)000163: add procedure for oneM2M interworking historical data with labels CIM(23)000172: add oneM2M child resource type attribute of mapping rule Technical Officer review for publication pre-processing after TB approval

History

Document history		
V1.1.1	December 2022	Publication
V1.2.1	January 2024	Publication