ETSI GS CIM 006 V1.2.1 (2023-06)

Context Information Management (CIM); NGSI-LD Information Model

Disclaimer

The present document has been produced and approved by the cross-cutting Context Information Management (CIM) ETSI Industry Specification Group (ISG) and represents the views of those members who participated in this ISG.
It does not necessarily represent the views of the entire ETSI membership.

Reference
RGS/CIM-006v121

Keywords
information model, interoperability, smart city

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

Contents

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The declarations pertaining to these essential IPRs, if any, are publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (https://ipr.etsi.org/).

Pursuant to the ETSI Directives including the ETSI IPR Policy, no investigation regarding the essentiality of IPRs, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

DECT™, **PLUGTESTS™**, **UMTS™** and the ETSI logo are trademarks of ETSI registered for the benefit of its Members. **3GPP™** and **LTE™** are trademarks of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners. **oneM2M™** logo is a trademark of ETSI registered for the benefit of its Members and of the oneM2M Partners. **GSM**® and the GSM logo are trademarks registered and owned by the GSM Association.

Foreword

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

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**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.

1 Scope

The purpose of the present document is to give property graphs a formal semantic grounding based on RDF/RDFS/OWL, with blank nodes reification, geared to JSON-LD serialization. On top of it, a set of core cross-domain ontology classes have been defined, based on this meta-model. This whole information model is meant to be used by many applications as a basis for data representations. It is compatible with the NGSI-LD API defined in ETSI GS CIM 009 [2].

2 References

2.1 Normative references

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

Referenced documents which are not found to be publicly available in the expected location might be found at https://docbox.etsi.org/Reference.

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 necessary for the application of the present document.

[1] W3C® Recommendation 16 July 2020: "JSON-LD 1.1: A JSON-based Serialization for Linked Data".

[2] ETSI GS CIM 009 (V1.7.1): "Context Information Management (CIM); NGSI-LD API".

[3] W3C® Candidate Recommendation Draft 15 November 2022: "Time Ontology in OWL".

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] Guinard, D., & Trifa, V. (2016): "Building the web of things", Shelter Island: Manning.

[i.2] Tim Berners-Lee (2006-07-27): "Linked Data", Design Issues W3C.

[i.3] J. Frey, K. Müller, S. Hellmann, E. Rahm and M.-E. Vidal (2017): Semantic Web - Interoperability, Usability, Applicability an IOS Press Journal: "Evaluation of Metadata Representations in RDF stores".

[i.4] Cassandras, C. G., & Lafortune, S. (2009) Springer Science & Business Media: "Introduction to discrete event systems".

[i.5] W3C® Editor's Draft 11 August 2005: "Simple part-whole relations in OWL Ontologies".

[i.6] W3C® Working Group Note 9 March 2006: "A Semantic Web Primer for Object-Oriented Software Developers".

[i.7] W3C®: "HttpRange14Webography".

[i.8] W3C® Interest Group Note 3 December 2008: "Cool URIs for the Semantic Web", Leo Sauermann and Richard Cyganiak.

3 Definition of terms, symbols and abbreviations

3.1 Terms

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

cross-domain ontology: part of the information model that defines generic classes (formal concepts and constructs, with associated constraints) that serve as common denominators between domain specific models, addressing the temporal and structural description of physical systems

domain-specific ontologies: information models that define base classes and their constraints, within specific technical domains (e.g. buildings, transportation, agriculture) and define their structure and vocabulary

meta-model: part of the information model that formally defines the NGSI-LD foundational classes (Entities, Relationships, Properties and reification constructs) on the basis of RDF/RDFS/OWL

NGSI-LD entity: informational representative of something that is supposed to exist in the real world, physically or conceptually. Any instance of such an entity shall be uniquely identified by a URI, and characterized by reference to one or more NGSI-LD Entity Type(s)

NGSI-LD property: description instance which associates a main characteristic, which is an NGSI-LD Value, to either an NGSI-LD Entity, an NGSI-LD Relationship or another NGSI-LD Property

NOTE: It includes the special "hasValue" property to define its target value.

NGSI-LD relationship: description of a directed link between a subject which is either an NGSI-LD Entity, an NGSI-LD Property, or another NGSI-LD Relationship on one hand, and an object, which is an NGSI-LD Entity, on the other hand

NOTE: It includes the special "hasObject" property to define its target object.

NGSI-LD value: JSON value (i.e. a string, a number, true or false, an object, an array), **or** a JSON-LD typed value (i.e. a string as the lexical form of the value together with a type, defined by an XSD base type or more generally an IRI), **or** a JSON-LD structured value (i.e. a set, a list, a language-tagged string)

EXAMPLE: An NGSI-LD Entity of type (Type Name) "Vehicle" (when parked) can be the subject of an NGSI-LD Relationship which has as its object a NGSI-LD Entity of type "Parking".

3.2 Symbols

Void.

3.3 Abbreviations

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

API	Application Programming Interface
CIM	Context Information Management
CNIT	Consorzio Nazionale Interuniversitario per le Telecomunicazioni
DBMS	DataBase Management System
GSMA™	GSM Association
HTTP	HyperText Transfer Protocol
IoT	Internet of Things
IRI	Internationalized Resource Identifier
ISG	Industry Study Group
JSON	JavaScript Object Notation

JSON-LD	JavaScript Object Notation for Linked Data
LD	Linked Data
LOD	Linked Open Data
NGSI	Next Generation Service Interfaces
NIR	Non-Informational Resource
OGC	Open Geospatial Consortium
OWL	Web Ontology Language
OWL-DL	Web Ontology Language - Description Logic
PG	Property Graph
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
SAREF	Smart Applications REFerence ontology
SAS	Société par Actions Simplifiée
SPARQL	SPARQL Protocol and RDF Query Language
URI	Uniform Resource Identifier
WoT	Web of Things
XML	eXtended Markup language
XSD	XML Schema Definition

4 Rationale for a multi-layered and multi-scale graph-based context information model

4.1 Why use a graph-based model?

Systems and environments about which context information is stored and managed encompass many physical and non-physical entities. Context comprises all characteristics of these entities, as well as their states and other dynamic properties, together with relationships that stand for real-world connections, i.e. physical, virtual, or even abstract connections between them. This context information may be consolidated on the basis of data obtained from many different primary sources and infrastructures. Typical examples of such systems would be smart homes, buildings, or cities. Such systems, due to the wide range of requirements and granularities, are complex from the semantic, structural and behavioural viewpoints.

The expressivity and versatility of graph-based models allows to bring the whole corpus of graph theory to bear and to capture key information about such complex environments, in a directly usable way, as the graph matches all kinds of real-world connections between different physical and non-physical entities.

Graph models bring a fresh view on the definition of context information. In the first wave of context-awareness research dating back to the early 2000s, context used to be mostly, and implicitly, user-centric, typically capturing e.g. the activity or location of mobile users to adapt services offered to them. A widely publicized definition of context, dating from this stage of research, was "any information that can be used to characterize the situation of an entity". In a broader view of context where the very notion of context is de-centred and relative, this definition may in fact remain valid if entities are represented as the nodes of a graph. Rather than through a vague notion of situation, context is defined in the present document as the set of properties characterizing these nodes, together with the set of relationships that enmesh them together, and the properties of these relationships. In this perspective, the primary data of one application may be the context of another, and vice versa. Context is, though decentered and broadly defined, the graph itself. NGSI-LD thus maintains and exposes context information as a graph of matching links between the informational units corresponding to real-world entities of these environments.

Traditional (mono-centred) context fitted rather well a classical object-oriented or key-value description, with a set of more or less detailed context features attached to a single entity. The multi-centered notion of context address in the present document requires breaking this rigid hierarchical model by using a more expressive, flexible and adaptable information model. Graphs are the only model adapted to capture the complex structure of inter-entity relationships that make up context information in the sense in which it is defined it here. This information need not be semantically defined from the outset: it may be natively structural information, capturing e.g. containment or adjacency relationships. The semantics of this context may be added in a later stage of graph enrichment. This model fits the natively distributed nature of context data sources.

The Web of Things (WoT) [i.1] does also involve a graph of sorts, but it dispenses with maintaining it explicitly inside a database. The WoT graph is, like the graph of the original web, an implicit 100 % distributed graph of hyperlinks, not between web pages but between resources corresponding to connected devices that expose an interface on the web (e.g. using HTTP, WebSockets, etc.).

This view also aligns well with the grand evolution of the Web towards Linked Data, an evolution proposed by W3C from the Semantic Web project [i.2] that is currently supported by RDF-derived graph models. Linked Data provides a method of publishing structured data so that it can be interlinked and support semantic queries.

4.2 Separating semantic referencing from structural descriptions

The NGSI-LD information model separates semantic referencing, used in the classical sense of the Semantic Web, from the structural description proper. The structural description may itself be decomposed into a basis structural graph whose nodes are physically-matched entities, and an overlay layer used to capture the way in which these entities are clustered into subgraphs.

Semantic referencing used by NGSI-LD is based on standard RDF/RDS/OWL typing and public ontologies, as shared by all other semantic information models. All nodes and edges of the structural graph are thus matched to several relevant classes/categories of these ontologies that jointly characterize the features shared by all instances of these classes.

A structural graph is a model of the structural description of an environment, capturing the relationships between the different subsystems that make up this environment. This description is, to some extent, independent of the overlaying semantic referencing, and it could be considered to "stand on its own", even without this referencing. A structural graph does in fact have a different kind of semantics of its own, such as e.g. when a graph captures and matches the structure of a physical network like a power grid or a water distribution network. These semantics apply to the graphs as a whole and are not reducible to the kind of "per-resource" semantics, which RDF is meant to describe.

4.3 Graph Examples used in the present document

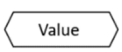
Two examples of structural representations of city environments will be used as lead examples throughout the present document and are presented in Figure 1 Property graph example (1) and Figure 2 Property graph example (2).

The following graphical conventions are used throughout the present document:

- 

Regular (physically-matched) entities are represented as black rectangles.
- 

Relationships between these entities are represented as diamonds (rhombuses) overlaid on the corresponding arc of the graph, a convention borrowed from "entity-relationship" diagrams.
- 

Properties are represented by ovals that are on an arc between their entity or relationship and the property value, but often the arc is shortened to zero length for compactness.
- 

Values are represented as hexagons that may about the oval of the property of which they are the target, omitting an arc between the two.

Figure 1 describes a parking scenario, adjacent to two different streets. Information about the streets, parking places, and the sensors that monitor are attached to entities as shown in the figure. This example is intended to illustrate the full expressivity of a property graph as used to capture not only pure semantics, as an RDF graph would, but also structural and behavioural (in this case, the real-time state) information.

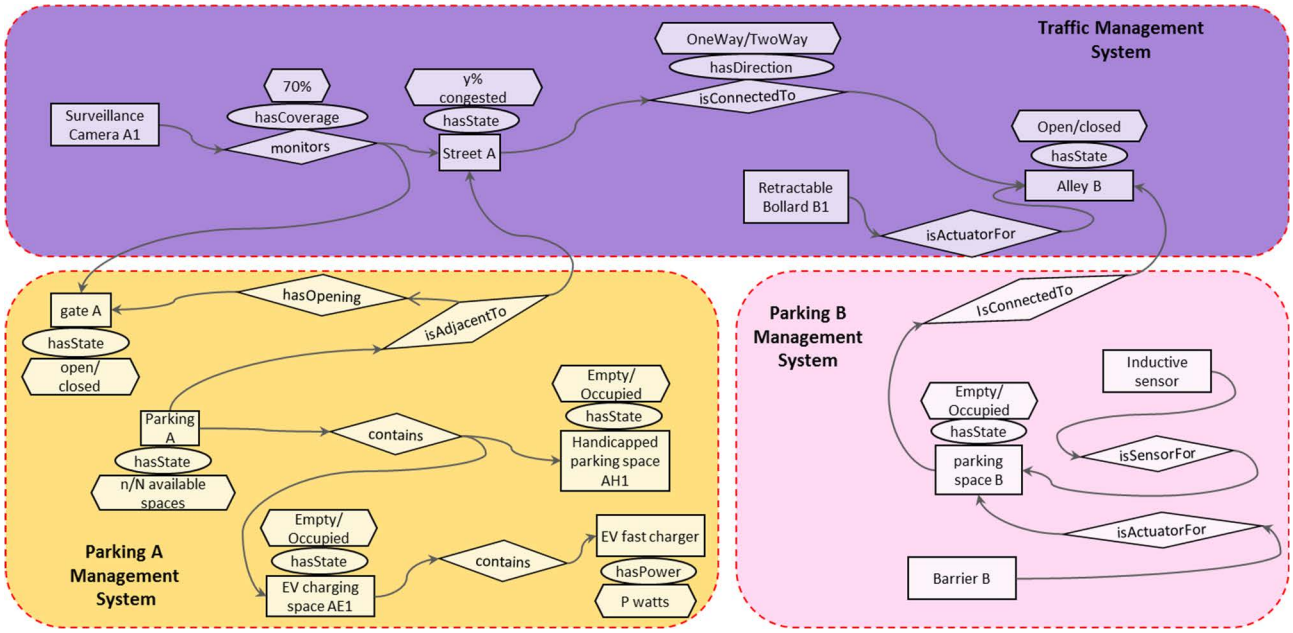


Figure 1: Property graph example (1)

Figure 2 example (2) is a more complex example used to illustrate intersecting domains and intertwined technical systems. The example consists of a building and its parts (using "hasPart" relationships) forming the structure of the building, in addition to other technical systems that are included in the building. The building is comprised of a garage and apartments (only one instance is represented below). A parking place within the garage belongs to the apartment, thus forming one system together. The building is equipped with a security system containing security devices. Additionally, there is a separate public parking that also appears in the example.

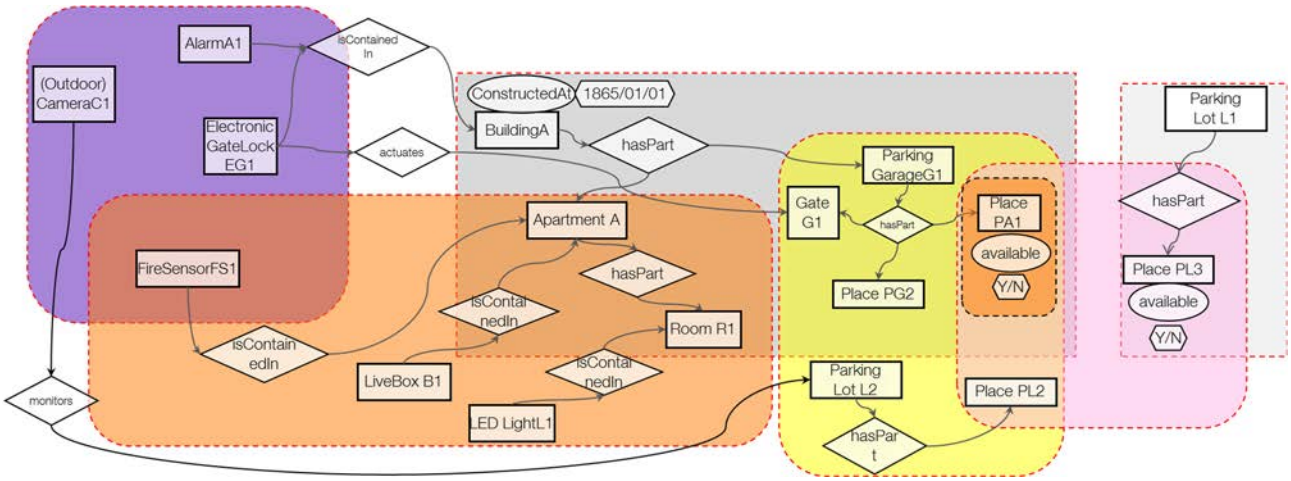


Figure 2: Property graph example (2)

5 NGSI-LD meta-model

5.0 Introduction

The NGSI-LD meta-model provides a formal basis for representing "property graphs" using RDF/RDFS/OWL. It makes it possible to perform back and forth conversion between datasets based on the property graph model on the one hand and linked data datasets which rely on the RDF framework, on the other hand. This may be seen as raising the semantic expressivity of RDF triples to the level of property graphs. Property graphs may, contrary to RDF, use predicates as subjects of other predicates (properties of properties and properties of relationships).

5.1 Fundamentals of property graphs and graph databases

Property graphs are the implicit semi-formal data models underlying most present-day graph databases. They have gained widespread following, more in industry than in academia. They make it possible to attach properties (defined as key-value pairs) to relationships, a feature which RDF does not directly support, but they lack the standardization and formal underpinnings of RDF and do not interoperate directly with linked data and other RDF datasets. Also they do not lend themselves to reasoning with RDF-based reasoning tools or querying with standard query languages such as SPARQL.

Property graphs are usually defined (informally) as follows:

- A property graph is made up of nodes (vertices), relationships, and properties.
- Nodes may have properties in the form of arbitrary key-value pairs. Keys are strings and values are arbitrary data types.
- A relationship is an arc (uni-directional, i.e. directed edge) of the graph proper, which always has an identifier, a start node and an end node. Like nodes, relationships can have properties attached to them.

There are several key differences between property graphs (PG in the following) and RDF graphs:

- RDF properties are expressed as regular triples, i.e. arcs of the graph with start node and end node, and their target can be either a literal, an IRI or a blank node [1], whereas the target of a PG property always corresponds to an RDF literal.
- PG relationships (i.e. primary graph links between PG vertices) are first-class citizens of the PG model and have an internal structure similar to that of a vertex, inherited from object-oriented modelling object, with an optional set of properties defined by key-value pairs.
- The distinction between relationships and properties in the PG model is similar to the distinction between object properties and datatype properties in OWL, but stronger.
- PG properties are, for simplicity and avoiding clutter in diagrams, usually not represented as additional arcs of an underlying graph, but are represented as attached to vertices or relationships.
- Identifiers in a Property Graph need only be unique within the scope of a given graph (typically as internal identifiers assigned by the Graph DBMS), and need not be universally unique like URIs/IRIs.
- Property graphs can be queried with graph-specific query languages that may use graph patterns (complete subgraphs) as query terms, i.e. are not limited to only nodes identified with specific key/values.
- PG properties and relationships are individually identified when instantiated, whereas RDF properties are not instantiated nor identified as individual resources, being only defined by their property type.
- Properties cannot be directly attached to the arc (predicate) of an RDF triple, but RDF reification makes it possible, in several different ways, to circumvent this limitation by turning a triple into a resource.

5.2 Reification with blank nodes

In the RDF formalism, the *reification* of a statement turns it into a resource, so that it can be the subject of another statement. Making statements about statements is useful e.g. for providing information about the provenance (lineage) of data. It is indispensable for transforming a property graph into an RDF dataset. Many different reification solutions have been proposed. Reification by way of blank nodes is the simplest for the current purposes and is the solution chosen by ETSI ISG CIM. Consider the following simple example.

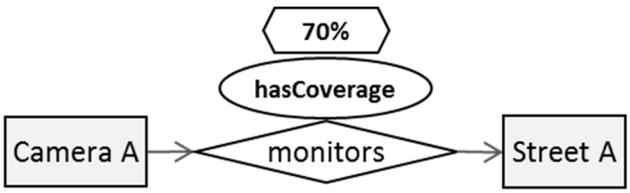


Figure 3: Property graph example to be represented in RDF using reification

To express that the camera monitors only 70 % of the street area, which obviously is not a property of the street, nor of the camera, but of their relationship, it is needed to reify this statement about the relationship:

[CameraA → monitors → StreetA]

in order to make it the subject of another statement:

[[statement_1] → hasCoverage → 70 %],

This can be done by adding a blank node to obtain an RDF-reified equivalent of the example property graph with three triples as follows and as visualized in Figure 4:

[CameraA → monitors → ***_blankNode_n****]*

*[****_blankNode_n*** *→ hasObject → StreetA]*

*[****_blank_node_n*** *→ hasCoverage → "70 %"]*

Figure 4: RDF reified example

This solution is especially convenient when the graph is serialized using JSON-LD ([1] see following clause) because blank nodes do not explicitly appear in the textual serialized description, and actually show up only when it is represented as a RDF graph. It is thus possible for a developer to generate the JSON-LD payload required by the NGSI-LD API in a form that is very similar to what he would have generated in plain JSON. The simplicity of JSON-LD representation of property graphs reified with blank nodes is a key argument behind the choice of this solution.

With alternative reification methods, users and developers shall include supplementary terms and shall deal with complex redundant terms that may distract and confuse them. Several such reification methods have been proposed in the literature (see e.g. [i.3]). For comparison, here is a brief description of three of the more widely used reification methods:

- *Classical RDF reification* defines a new RDF resource that is linked back to the original statement. This uses RDF built-in reification capabilities, as RDF natively provides a vocabulary intended for describing RDF statements, namely the type `rdf:Statement`, and the properties `rdf:subject, rdf:predicate, and rdf:object`. A total of 4 additional statements (corresponding to the so-called "reification quad") are required to fully define a statement as a resource, and this is just in order to be able to make this resource the subject or object of other statements.

- *Singleton properties:* this other simple solution to reification amounts to identifying each predicate instance individually as a resource with its own per instance IRI, and using this new resource as the subject of another statement. This actually changes the nature of the original RDF graph because what was originally an arc of the graph becomes a vertex of the transformed graph.

- *Named graphs/quads:* all RDF triples are redefined as "quadruples" ("quads" for short, unrelated to the "reification quads" mentioned above). These quads are generalizations of triples, with 4-fold *arity* (whereas the above "reification quads" are sets of 4 triples!). These quads have as the extra element an associated IRI that identifies the statement as an instance of the corresponding predicate (which is itself, as per the RDF model, defined by a generic IRI). This fourth element, the IRI, makes it possible for the statement to be the subject (or object) of another RDF triple. Named graphs are much more powerful than this basic quad mechanism, in that they allow any subgraph (set of interconnected triples) to be jointly identified. JSON-LD [1] supports these general named graphs but it is a very cumbersome and heavyweight means to reify simple triples. The reification method used in the present document is much more lightweight than the "*Named graphs*" approach.

These reification methods are compared in Figure 5 (Standard RDF reification - with quads) and in Figure 6 (Singleton property reification), with the proposed blank-node-based reification method, from two points of views: (a) JSON-LD corresponding representation and (b) SPARQL query complexity for extracting data.

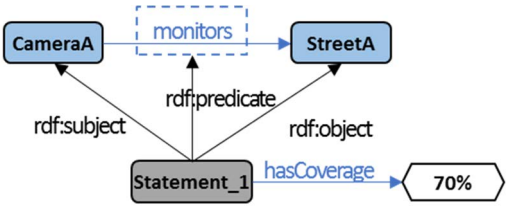


Figure 5: Standard RDF reification

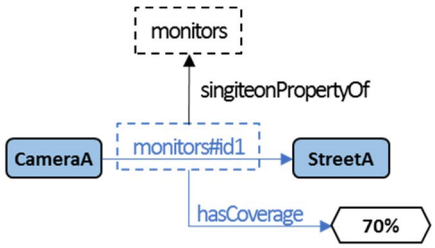


Figure 6: Singleton property reification

JSON-LD Format:

```
[
{"@id": "CameraA",
"monitors": {"@id": "StreetA"}}

{"@id": "Statement_1",
"subject":
 {"@id": "CameraA"},
"predicate":
 {"@id": "monitors"},
"object":
 {"@id": "StreetA"},
"hasCoverage": "70%" }
]
```

SPARQL Query:

```
SELECT ?R WHERE {
?st rdf:subject  :CameraA.
?st rdf:predicate
                :monitors.
?st rdf:object StreetA.
?st :hasCoverage ?R
}
```

JSON-LD Format:

```
[
{"@id": "CameraA",
"monitors#id1":
 {"@id": "StreetA}}

{"@id": "monitors#id1",
"singletonPropertyOf":
 {"@id": "monitors"},
"hasCoverage":"90%" }
]
```

SPARQL Query:

```
SELECT ?R WHERE {
:CameraA ?p :StreetA.
?p :singletonPropertyOf
                :monitors.
?p :hasCoverage ?R
}
```

Using reification with blank nodes, the SPARQL query is as follows:

```
SELECT ?R WHERE {
:CameraA :monitors ?bn.
?bn :hasObject :StreetA.
?bn :hasCoverage ?R.
}
```

For targeting directly the query to the object of "monitors" instead of the value of the coverage of the monitoring, the `owl:propertyChainAxiom` is used as follows:

```
: monitors owl:propertyChainAxiom (:monitors:hasObject).
```

This can be defined for all reifiable properties similar to "monitors" in the preceding statement. The SPARQL query for the object of the property becomes simple and equivalent to queries without reification.

```
SELECT ?S where {
:CameraA : monitors ?S.}
```

5.3 Formal definition

5.3.0 Introduction

For the purposes of the present document, and as shown in clause 3.1, the following terms and definitions apply (for convenience and brevity the "NGSI-LD" prefix may be omitted in the rest of the present document):

NGSI-LD Entity: An NGSI-LD Entity is the informational representative of something that is supposed to exist in the real world, physically or conceptually. Any instance of such an entity shall be **uniquely identified by a URI**, and characterized by reference to one or more **NGSI-LD Entity Type(s)**.

NGSI-LD Property: A description instance which associates a main characteristic, which shall be an **NGSI-LD Value**, to either an NGSI-LD Entity, an NGSI-LD Relationship or another NGSI-LD Property. It shall include the special "hasValue" property to define its target value.

NGSI-LD Value: An NGSI-LD Value is either a JSON value (i.e. a string, a number, true or false, an object, an array), or a JSON-LD typed value (i.e. a string as the lexical form of the value together with a type, defined by an XSD base type or more generally an IRI), or a JSON-LD structured value (i.e. a set, a list, a language-tagged string).

NGSI-LD Relationship: An NGSI-LD Relationship describes a directed link between a subject which shall be either an NGSI-LD Entity, an NGSI-LD Property, or another NGSI-LD Relationship on one hand, and an object, which shall be an NGSI-LD Entity, on the other hand. It shall include the special "hasObject" property to define its target object.

EXAMPLE: An NGSI-LD Entity of type (Type Name) "Vehicle" (when parked) can be the subject of an NGSI-LD Relationship which object is a NGSI-LD Entity of type "Parking".

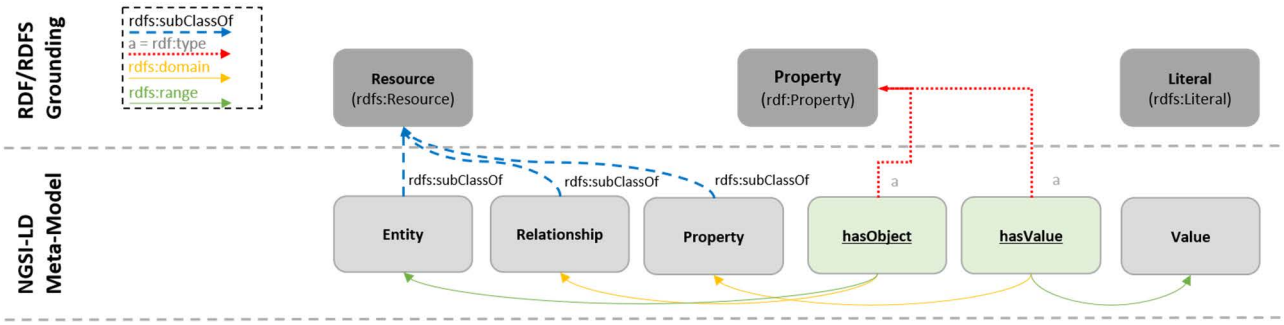


Figure 7: NGSI-LD core meta-model diagram

5.3.1 Entity Types

Entity types are classes in OWL and shall be defined in a hierarchy as subclasses of the class "Entity".

This allows inheriting common characteristics from super classes, and gives a common vocabulary for all domain-specific meta-models that will ever be defined over NGSI-LD.

5.3.2 Properties and Relationships Types

Similar to Entity types, the types for Properties and Relationships are defined in a hierarchal manner as subclasses of the classes "Property" and "Relationship" respectively. They are used to categorize an NGSI-LD Relationship as belonging to a class of similar relationships.

Property (or Relationship) types shall be identified by a URI. These types shall also be used for typing blank nodes used for reification, according to the type of the property or relationship which they reify.

`Entity, Property, Relationship` are direct subclasses of the `rdfs:Resource` or `owl:Thing` class (default for classes for RDFS and OWL respectively).

```
:Entity rdf:type owl:Class .
…
```

`Value` in the NGSI-LD meta-model is not limited to the `rdfs:Literal`. The class `Value` is extensible as needed to allow more value formats in domain specific extensions of the meta-model. `Value` may be an `rdfs:Literal`, but may also be some specific type as defined by a programming language, e.g. integer. A value shall be neither an `Entity`, a `Property`, nor a `Relationship`.

```
:Entity owl:disjointWith :Value .
:Property owl:disjointWith :Value .
:Relationship owl:disjointWith :Value .
```

Two primitive RDF properties are defined: `hasValue` and `hasObject`. These are used to "pre-reify" all relationships and properties by enabling their potential reification with blank nodes as explained in the previous clause, even if these properties or relationships are not actually the subject of another property or relationship.

In OWL, `owl:DatatypeProperty` and `owl:ObjectProperty` correspond to two kinds of `rdf:Property`, yet in OWL there is a distinction between the default range of these two classes, where the range of the first is a literal while the range of the other is a class.

```
:hasValue rdf:type owl:DatatypeProperty .
          rdfs:domain :Property .
:hasObject rdf:type owl:ObjectProperty .
          rdfs:domain :Relationship ;
          rdfs:range :Entity .
```

An example of usage of `hasValue` for applying a property (ObservedAt) to another "pre-reified" property (colour) is given below:

```
:Car1 :colour _:bn
_:bn hasValue "Blue"
_:bn observedAt "2018-01-01T00:00:00Z"
```

5.4 Serialization with JSON-LD

JSON-LD is a JSON-based syntax standardized by W3C [1] for serialization of Linked Data, and more generally, RDFdatasets.

The property graph example given Figure 1 can be represented in RDF, using the reification method with blank nodes as shown in Figure 8.

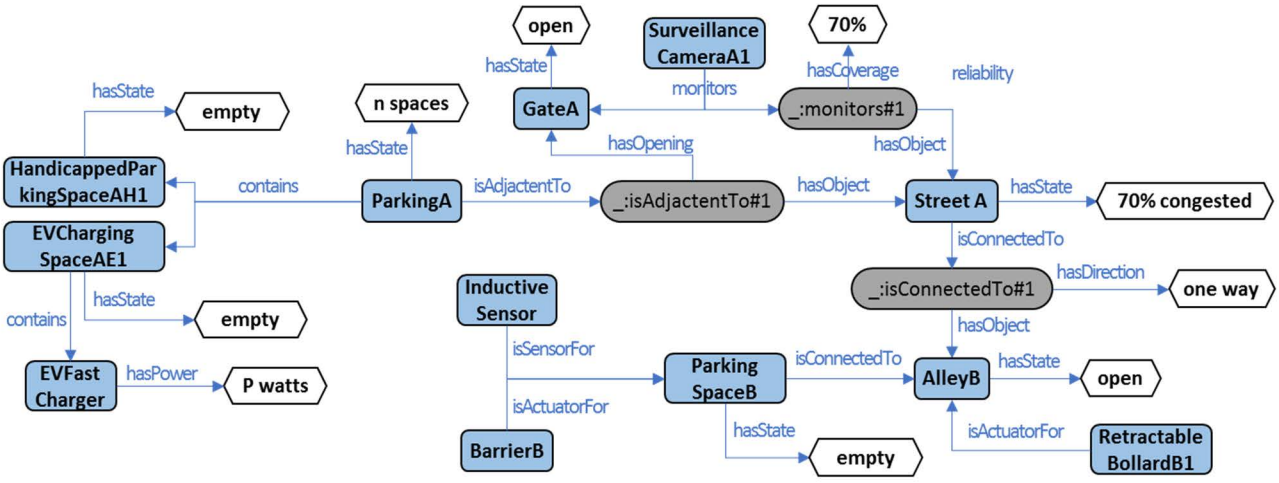


Figure 8: Property graph example transformed to RDF with blank-node reification

It is an implementation choice whether all property and relationship instances shall be reified, or only specific instances where reification is needed (i.e. in case of properties/relationships applied on other properties/relationships). In the RDF example of Figure 8, for simplicity, only reified instances are shown where needed, that is where extra information is attached to the properties and relationships (e.g. "hasDirection" property is attached to "isConnectedTo" relationship in the example above). It should be mentioned that the NGSI-LD API requires all properties and relationships to be "pre-reified" by default (i.e. even if no extra properties are attached to them) using the "special "hasValue and "hasObject" properties and relationships, as explained in the previous clause.

Our meta-model solution that is based on blank node reification is especially convenient when the graph is serialized with JSON-LD because blank nodes do not explicitly appear in the textual serialized description, and actually show up only when it is represented as an RDF graph. It is thus possible for a developer to generate the JSON-LD payload of an API in a form that is very similar to what he would have generated in plain JSON.

The previous RDF example can be written in the JSON-LD format as follows:

```
[
{"@id": "ParkingA",
"isadjacentTo": {"hasObject": {"@id":"StreetA"},
                              "hasOpening": {"@id":"GateA"}},
"hasState":"25 spaces"
"contains": {"@id":"HandicappedParkingSpaceAH1"},
"contains": {"@id":"EVChargingSpaceAE1"}},

{"@id": "StreetA",
"hasState": "70% congested",
"isConnectedTo":    {"hasObject": {"@id":"AlleyB"},
                                   "hasDirection":"one way"}},

{"@id": "HandicappedParkingSpaceAH1",
"hasState":"empty"},

{"@id": "EVChargingSpaceAE1",
"hasState":"empty",
"contains": {"@id":"EVFastCharger"}},

{"@id": "EVFastCharger",
"hasPower":"1000 watts"},

{"@id": "GateA",
"hasState":"open"},

{"@id": "SurviellanceCameraA1",
"monitors": [{"@id":"GateA"},       {"hasObject":{"@id":"StreetA"},
                                                                  "hasCoverage":"70%"}]},

{"@id": "AlleyB",
"hasState":"open"},

{"@id": "ParkingSpaceB",
"isConnectedTo":    {"@id":"AlleyB"},
"hasState":"empty"},

{"@id": "RetractableBollardB1",
"isActuatorFor":    {"@id":"AlleyB"}},

{"@id": "InductiveSensor",
"isSensorFor":  {"@id":"ParkingSpaceB"}},

{"@id": "BarrierB",
"isActuatorFor":    {"@id":"ParkingSpaceB"}}
]
```

Such JSON-LD representation is much simpler than that for the same set of information if the standard RDF reification with quads was used, which is cumbersome, and shows nodes that are not desired or needed to be shown for developers and users.

6 ETSI ISG CIM cross-domain ontology

6.1 Rationale

The proposed cross-domain ontology provides definitions of entity types, relationship types, property types and types that are considered to be common denominator between all domains where NGSI-LD will be applied, bridging vertical ontologies used in these domains and the core meta-model defined before.

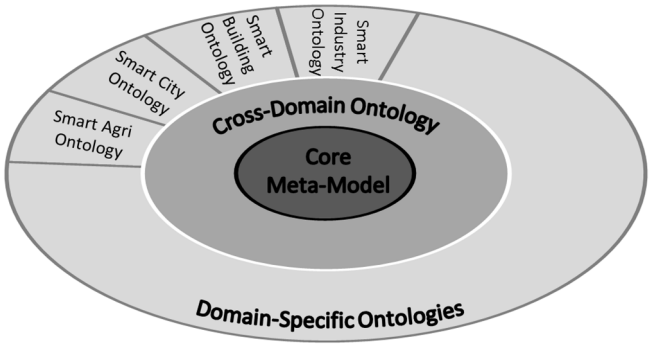


Figure 9: NGSI-LD cross-domain ontology positioning

The cross-domain ontology is the second part of the NGSI-LD information model, which is no less important than the core meta-model defined before. The scope of this cross-domain ontology is strictly limited so as to be useful for describing common concepts, situations or constraints, to avoid redefining them separately (and probably in a different way) in each domain-specific ontology. This promotes consistency when applications need to combine data from different sources, themselves from different domains.

The cross-domain ontology is defined such that all needed domain-specific ontologies should be expressible on top of the NGSI-LD cross-domain ontology, but the cross-domain ontology should not be so specific that it is tied to any domain-specific definitions that would render its vocabulary irrelevant for other use cases.

6.2 NGSI-LD API compatibility

The terms used by the API previously specified by the ETSI GS CIM 009 [2] are compliant with the current information model specification The API actually uses the information model for maintaining a stable structure with the convenient restrictions among different implementations/utilizations of the system so as to achieve consistent results in almost all cases.

The minimum requirement for the cross-domain information model is that it include all the API vocabulary for the data representation, yet it is not necessary that it be limited to the API vocabulary and needs.

An extended information model that is able to better express information in a stable way, and which is compliant with other standards, provides an opportunity for additional adoptions of the NGSI-LD standard.

Even if it has been previously defined by the CIM API specification (besides other protocols and implementation recommendations) the present information model specification is not subordinate to the NGSI-LD API. It can be used independently of this API by CIM-compatible platforms and systems, This is why it is necessary to describe the information model separately and more in depth in the present document.

6.3 Formal specification

6.3.0 Comparison with other approaches

Figure 10 shows an RDF/RDFS/OWL based diagram representing the hierarchical relation between types defined in the cross-domain ontology, as well as their relation to the core meta-model, as they have been defined in a minimal way for the strict needs of the NGSI-LD API.

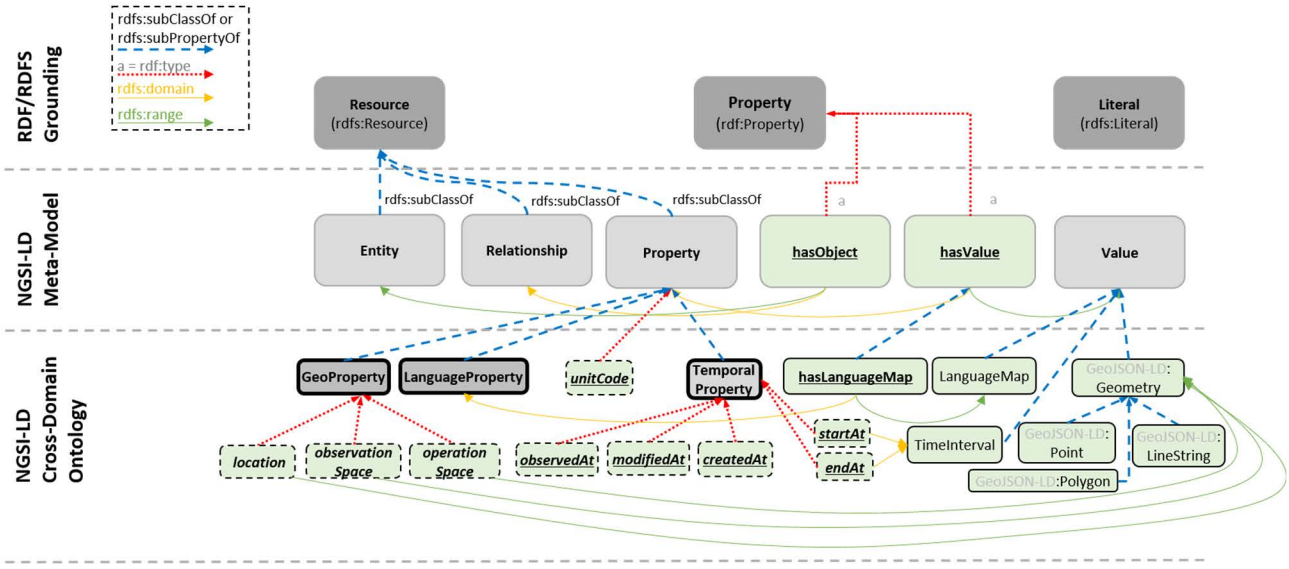


Figure 10: Minimal cross-domain ontology used in NGSI-LD API

This cross-domain information model is completely compliant with the previous API ontology (ETSI GS CIM 009 [2]). It uses the same vocabulary yet it additionally provides a bigger set of definitions that may be helpful for many users for maintaining consistency. Figure 11 shows the full cross-domain ontology of the information model.

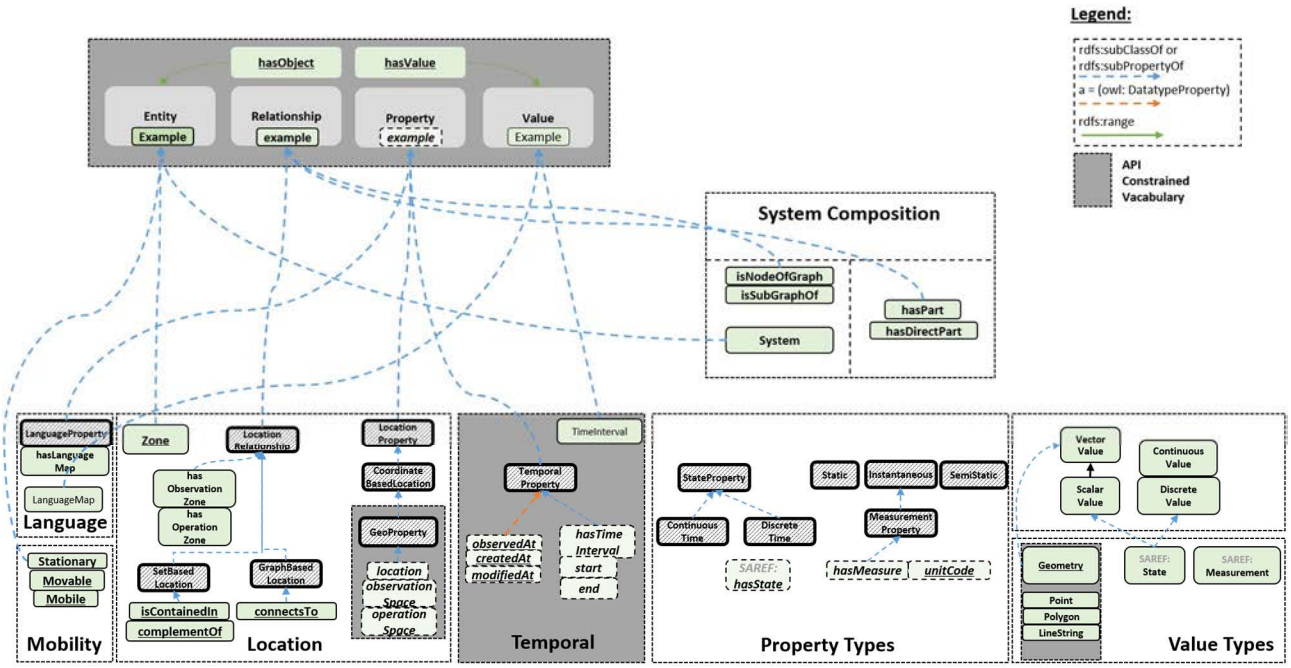


Figure 11: NGSI-LD cross-domain ontology diagram (Full)

The classes in the cross-domain meta-model are colour-coded according to the following legend.

Class (Entity/Value): [With capital initial]. Used to refer to a class that is a subclass of Entity or Value.

Class (Properties/ Relationships): [With capital initial]. Used to refer to a class that is a subclass of Property or Relationship, but which is not itself a property or a relationship. These classes serve as super-classes for a set of properties or relationships in the same domain or aspect.

property Name and relationship Name: [With small initial]. Used to refer to a proper (direct) class of properties or relationships..

liteProperty Name: [With small initial and underlined text]. Used to refer to the name of a property that is considered to be "lite" in its informational representation since it shall not be reified, rather a value is directly attached to it.

Ontology:Class: [With small or capital initial]. Used to refer to a class or a vocabulary that is inherited from another publicly available standard or ontology.

The class characteristics of the information model are general enough to be used in a common way throughout different domain specific ontologies. In addition, such distinction is itself useful for enriching the semantics of data.

Although such classifications are recommended where applicable, they are considered optional if they go beyond the API vocabulary. That is, deployments are encouraged to use them, but it is not mandatory. If a deployment uses them, however, then the classes shall be used as defined in the present document.

In addition to the Class descriptions presented above, for further defining the specifications of the NGSI-LD information model, it is explained in the following how typing works with reified RDF representation, and also how restrictions are applied to them in order to enable correct reasoning on the data representation.

A property type in the NGSI-LD model is identified by a URI which corresponds to an RDF property, and at the same time to an RDF class (which is a subclass of the class Property of the meta-model). The RDF class interpretation of the URI will be used to type blank nodes that follow the property usage in the reified form.

The same holds for relationships. A single URI corresponds to an RDF property and an RDF class (which is a subclass of the class Relationship of the meta-model).

Figure 12 shows an example of how typing holds for blank nodes following the reification of the relationship "isContainedIn".

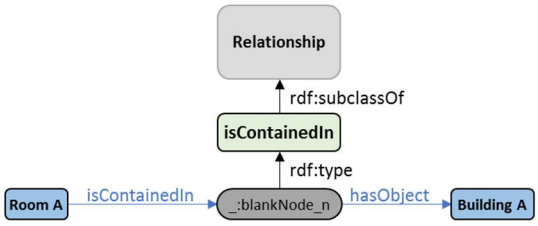


Figure 12: Blank node typing example for reification

The example of Figure 12 includes the information that "Room A" is spatially contained in "Building A".

The first step is to assert a property chain as follows:

```
:isContainedIn owl:propertyChainAxiom
                                        (:isContainedIn :hasObject) .
```

In the current example this means it can be inferred that "Building A" is also directly connected to the "isContainedIn" RDF property, and not only to the "`hasObject`" property.

The second step is to define the RDF class that is a range for the "`isContainedIn`" RDF property. This should be a union of two RDF classes, the RDF class "`isContainedIn`" (to account of the blank node instances) and the RDF class "`Entity`". This is defined in OWL as follows:

```
:isContainedIn rdfs:range
                        [ rdf:type owl:Class ;
                            owl:unionOf ( :isContainedIn :Entity ) ] .
```

A general set of restriction rules that serve for establishing compatibility between the cross-domain ontology and the core meta-model are defined in the following. For the definition of the information model, some basic rules should always be followed while defining the vocabulary of the cross-domain ontology in the RDF/OWL standards (Turtle format is used in the following descriptions).

Another set of rules are defined for property types (e.g. `observationSpace`). These rules apply to both OWL-DL and OWL-full when not specified, but they only apply to OWL-full when specified beside the rule.

For each property type `p`:

- the following rules shall hold:
  - `p rdf:type owl:ObjectProperty`
  - `p rdfs:subPropertyOf Property`
    (Either direct or by inference, i.e. p may also be defined as an rdfs:subPropertyOf some owl:ObjectProperty which is a subproperty of Property.)
  - `p rdfs:subClassOf Property`
    (Either direct or by inference, i.e. p may also be defined as an rdfs:subClassOf some class which is a subclass of Property.)
  - [OWL-full] `p owl:propertyChainAxiom (p hasValue)`
    (Since reification uses a supplementary property hasValue with Properties, a shortcut for extracting data by only using the property p is allowed. This assertion means that any property p usage followed by a hasValue usage is semantically equivalent to using p alone. This allows implicitly inferring the values of properties without using the reification through blank nodes, but directly through the name of the property when no other information is needed.)
- the following rules may hold: (where restrictions are needed)
  - `p rdfs:domain C` (*Where* `C` *is a subclass of* `Entity, Property`, *or* `Relationship`)
    (*This limits the usage of p on a certain class of Entities, Properties, or Relationships.*)
  - `p rdfs:range V` (*Where* `V` *is a subclass of* `Value`)
    (*This limits the values that can be associated with p.*)

Some properties in the NGSI-LD API have specifically been defined such that they shall never be reified and do not use the "hasValue" construct. Those properties include: *observedAt*, *createdAt*, *modifiedAt*, *start*, *end* and *unitCode*. Those properties are direct instances (using *rdf:type*) of the class `Property`**.** Similar to property types, the rules for relationship types are defined.

For each relationship type `r`:

- the following rules shall hold:

  `r rdf:type owl:ObjectProperty`

  - `r rdfs:subPropetyOf Relationship`
    (Either direct or by inference, i.e. r may also be defined as an rdfs:subPropertyOf some owl:ObjectProperty which is a subproperty of Relationship.)
  - `r rdfs:subClassOf Relationship`
    (Either direct or by inference, i.e. r may also be defined as an rdfs:subClassOf some class which is a subclass of Relationship.)
  - `r rdf:type r`
    (r is both a class which is the class of relationship blank node instances, and an instance of itself because it is used as a property that links subjects to the relationship blank nodes.)
  - `r owl:propertyChainAxiom (r hasObject)`
    (Since reification uses a supplementary property "hasObject" with Relationships, a shortcut is allowed for extracting data by only using the relationship r. This assertion means that any relationship r usage followed by a hasObject usage is semantically equivalent to using r alone. This allows implicitly inferring the values of relationships without using the reification through blank nodes, but directly through the name of the relationship when no other information is needed.)

- the following rules may hold: (where restrictions are needed):

  - `r rdfs:domain C1` (*Where* `C1` *is a subclass of* `Entity, Property`, *or* `Relationship`)
    (*This limits the usage of r on a certain class of Entities, Properties, or Relationships*)

  - `r rdfs:range C2` (*Where* `C2` *is a subclass of* `Entity`)
    (*This limits the Entities that can be associated with r*)

In the following clauses, different subsets of the cross-domain meta-model are described separately.

6.3.1 Mobility (of Entities)

Given an entity, it is optional, but recommended, to choose a mobility typing which expresses generic constraints on this entity, so that these constraints do not have to be re-specified by each domain-specific ontology or application. For example, if an entity is typed as a stationary entity as defined here then it shall not have a "speed" property. Any entity defined by a domain-specific ontology, such as a building, need not exclude the "speed" property, it is excluded by default if its type subtypes "stationary".

The cross-domain ontology distinguishes between three mutually exclusive types of mobility:

1) **Stationary:**
   Location property associated with such an entity is also a static property (by `owl:restriction`).

2) **Movable:**
   Location property associated with such an entity is also a semi-static property (by `owl:restriction`).
   Location historic data may be available for such an entity.

3) **Mobile:**
   Location property associated with such an entity is also an instantaneous property (by owl:restriction).
   Location historic data may be available for such an entity.

6.3.2 Properties

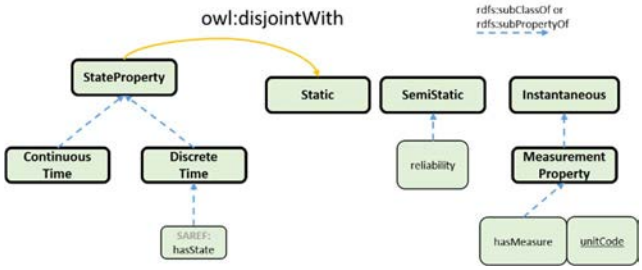


Figure 13: Cross-Domain Properties

The cross-domain ontology distinguishes between three types of mutually exclusive properties with respect to the time-validity of their respective values.

1) Static

2) SemiStatic

3) Instantaneous

This distinction may be useful within some applications for defining error/prompt messages where convenient (e.g. trying to change a static property).

Like mobility, this property distinction also allows implementations to control historic data storing behaviour.

`StateProperty`**:** in the formal acceptation of dynamical system theory, a `StateProperty` captures a component of the state of a system represented by the corresponding entity, in the sense that it memorizes/summarizes past interactions between this system and its environment ("inputs" to the system), in a way that is *sufficient to adequately predict future states of this system, given present and future inputs*. This is related to the notion of "statefulness" used in computer science, yet the state thus maintained is defined as an abstraction of the state to the physical system being represented, not a purely informational construct. This property does also capture the necessary differentiation between the state properties of a system and other "stateless" data streams like sensor readings.

The `StateProperty` class is disjoint with the `Static` property class. A state property cannot be static since it represents a property that, by its very nature, changes with time. There are two types of state properties [i.4]:

1) `ContinuousTime` (e.g. Car position): Such properties shall correspond to ContinuousValue system states.

2) `DiscreteTime` (e.g. Occupancy of a parking place): DiscreteTime systems in general may have state properties whose value may be either ContinuousValue or DiscreteValue. Event-driven discrete time systems, which are the most usual case addressed in computer-controlled cyber-physical systems, have discrete value states.

6.3.3 Location (Property or Relationship)

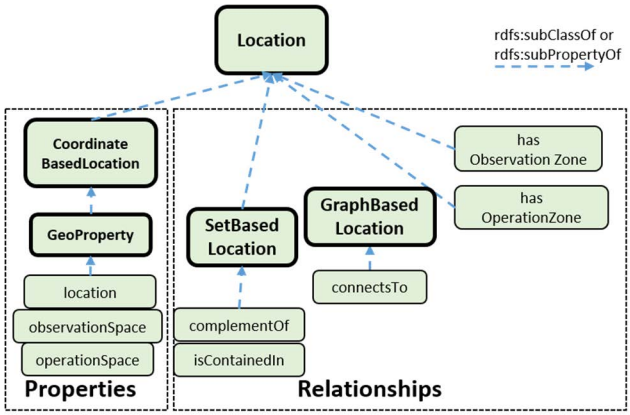


Figure 14: Cross-Domain Location

Our cross-domain meta-model differentiates between three notions of location:

1) **Geographic Location (GeoProperty):**
More generally, this is a special case of `CoordinateBasedLocation`. There are many standards/ontologies that allow expressing geographic location data. Since the NGSI-LD API uses JSON-LD as a main serialization format, the **geoJSON** standard is used for geographic location expressions. Only the specified GeoProperties represented in geoJSON can be used in geographic queries of the NGSI-LD API. Other standards/ontologies may be more expressive in terms of vocabulary (e.g. OGC standards). While these are not promoted as main methods for representing location, they can be mapped as subclasses of the **Geographic Location** class in order to utilize their rich vocabulary.

NOTE: "observationSpace" and "operationSpace" are properties whose range is a **Value** of type **Geometry**, whereas "hasObservationZone" and "hasOperationZone" are relationships whose range is an **Entity** of type **Zone**.

2) **SetBasedLocation (Relationships):**
In many cases, such as indoor location, it is more relevant to describe location in terms of relationships to an Entity like a room, floor, or other building part the relationship "**IsContainedIn**" is used to assert this meaning.

EXAMPLE 1: TableA (isContainedIn)→ RoomA (isContainedIn)→ BuildingA.

3) GraphBasedLocation (Relationships):
This is the notion of location that is used for indoor navigation, or for navigation in a city through its street system represented as a grapH.

EXAMPLE 2: RoomA (connectsTo)→ RoomB, street A connectsTo →Crossing AB ConnectsTo→ Street B.

6.3.4 Values

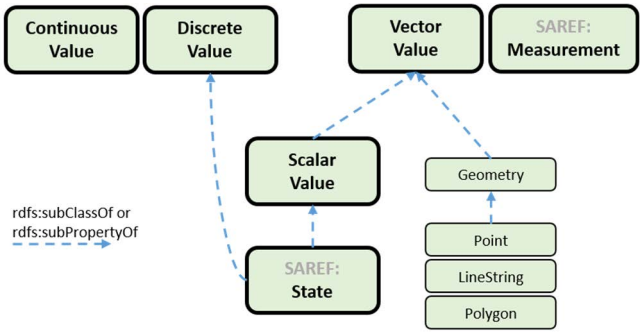


Figure 15: Cross-Domain Values

In compliance with properties distinctions, values that are **Continuous** are distinguished from those that are **Discrete**.

Vector values are tuples (multiple components) - (they may also be called complex values).

EXAMPLE: For a geometric value two components are needed:

- type: [point, polygon, etc.];
- coordinates.

Scalar value is a simple value, or in other words a vector value of single dimension.

6.3.5 Temporal Properties and Values

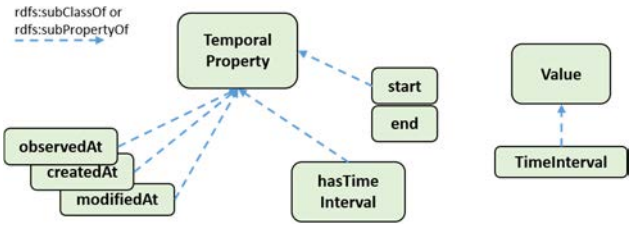


Figure 16: Cross-Domain Temporal Properties

The core vocabulary for time related meta-data is given as follows:

1) observedAt
2) createdAt
3) modifiedAt

The time related properties listed above do not follow the reification procedure (i.e. values are attached to them directly, without the blank node mediator). Such properties are considered to be completely specified through one value statement in the NGSI-LD information model, and cannot be further annotated with other properties or relationships.

The other time related properties and values shown in Figure 16 are taken from the W3C Time Ontology [3].

6.3.6 Systems Composition

6.3.6.0 Introduction

The information model of NGSI-LD needs to take into account modelling of context information for real-world systems that have contexts which are not only nested (a sub-system is part of a larger system, possibly at several levels) but also intertwined (systems or sub-systems sharing contexts with several other systems) and overlapping (systems sharing a sub-system), depending on the scenario or use case. Therefore definitions of systems shall be possible in terms of groupings of entities according to many diverse criteria: structural (a component of a machine, a branch of a physical network), functional (a software function in a security architecture, spatial (all the objects in a room), legal or administrative (objects belonging to someone, all towns within a government area), etc.

A purely conceptual criterion (entities instances belonging to the same category, using classical semantic referencing that can be accounted for with RDF/RDFS/OWL typing and subtyping), may be used here, but only in combination with the previous criteria that place appropriate additional physical constraints.

A complementary motivation for defining systems is the separation of concerns which allows retrieving/storing data within appropriate scopes and levels, and thus providing a tool for dividing information based on security, privacy, and/or management criteria.

In the NGSI-LD information model the following vocabulary is used for the definition of systems and their composition as explained in the following subclauses. For comparison, see W3C specifications for part/whole relations [i.5].

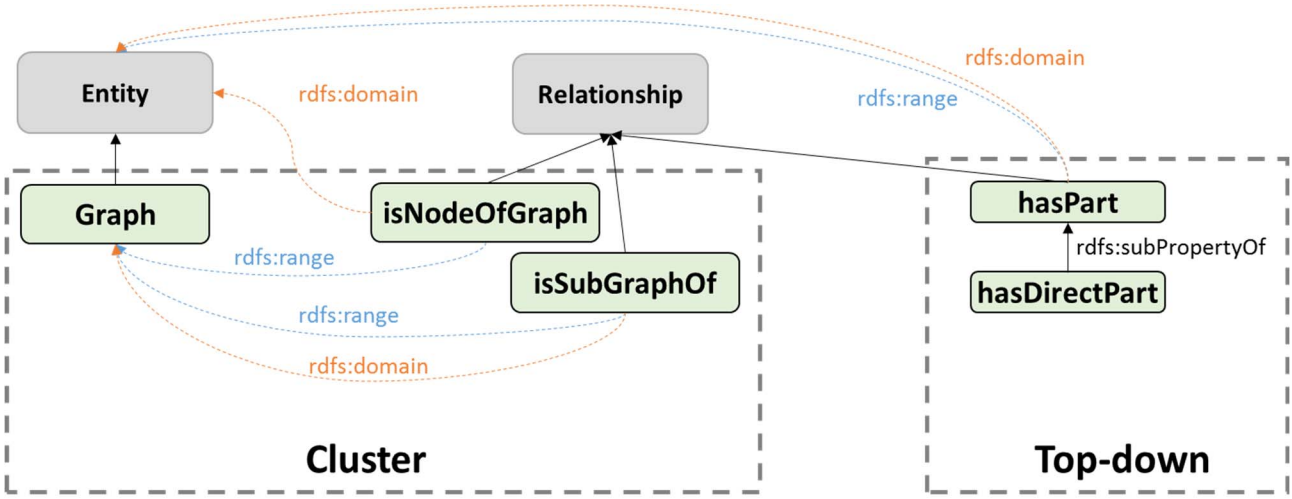


Figure 17: Cross-Domain System Composition

6.3.6.1 Top-down system composition

The "**hasPart**" and "**hasDirectPart**"relationships define *typically* the decomposition of tightly-coupled, and self-contained (non-physically-distributed) systems into subsystems that are typically designed as components of the system of which they are a part. These relationships are "top-down" in that their domain is the overall container system and the range is the contained component subsystems. They may be used to describe e.g. the structure of a building, the components of a traditionally-engineered manufactured system.

As its name suggests, "**hasDirectPart**" defines parts of the closest level of composition in the system, while "**hasPart**" allows the assertion of parts, parts of parts, for as many nested levels as required.

The information model defines "**hasPart**" as a transitive relationship. If this reasoning feature is supported by the system implementing the NGSI-LD API, the inference of parts of parts at all levels becomes implicit. However, this is not mandatory.

6.3.6.2 Bottom-up system composition and clustering

In addition to classical self-contained systems described by the previous "hasPart" relationships, there is a need to describe systems that are *typically* assembled organically or incrementally in a bottom-up fashion and may integrate non-physically-related objects, or may, in the extreme, correspond to loose clusterings. This type of system composition does *typically* correspond to physically-distributed systems, though this is not a requirement per the definition. A city or complex building is an obvious example for this, where the subsystems have not been designed from the outset to be part of the larger system, and may be added or removed without compromising the function of the larger system.

NGSI-LD defines the **Graph** class to capture this type of bottom-up system grouping or loose clustering, implicitly understood as a subgraph of a larger graph of either a larger graph system or the overall graph that captures the whole environment. Contrary to other NGSI-LD entities, nodes of **Graph** type, do not stand directly for physical counterpart entities with a one-to-one mapping. The graph is an informational abstraction making up the scaffolding of a system. The special "**isNodeOfGraph**" relationship is used to define, in an appropriately bottom-up direction, the composition of their constituent subsystems into these systems.

The mapping of NGSI-LD entities to graphs is typically many-to-one and one-to-many.

Figure 18 shows a set of such system graphs, capturing a structural overlay on the example introduced in Figure 1.

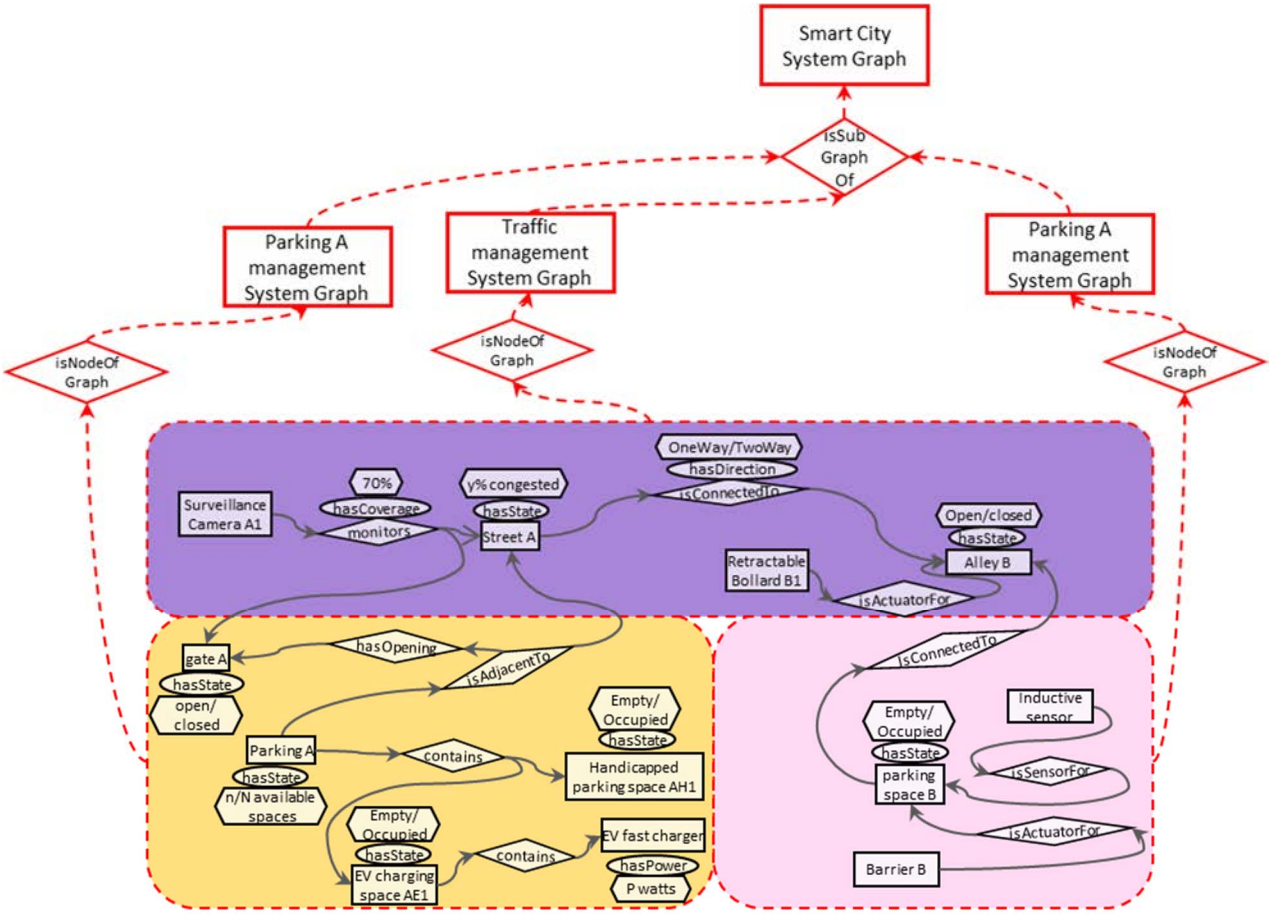


Figure 18: System Clustering Example (1)

Figure 19 shows a more complicated set of intersecting systems, capturing an overlay on the example introduced in Figure 18.

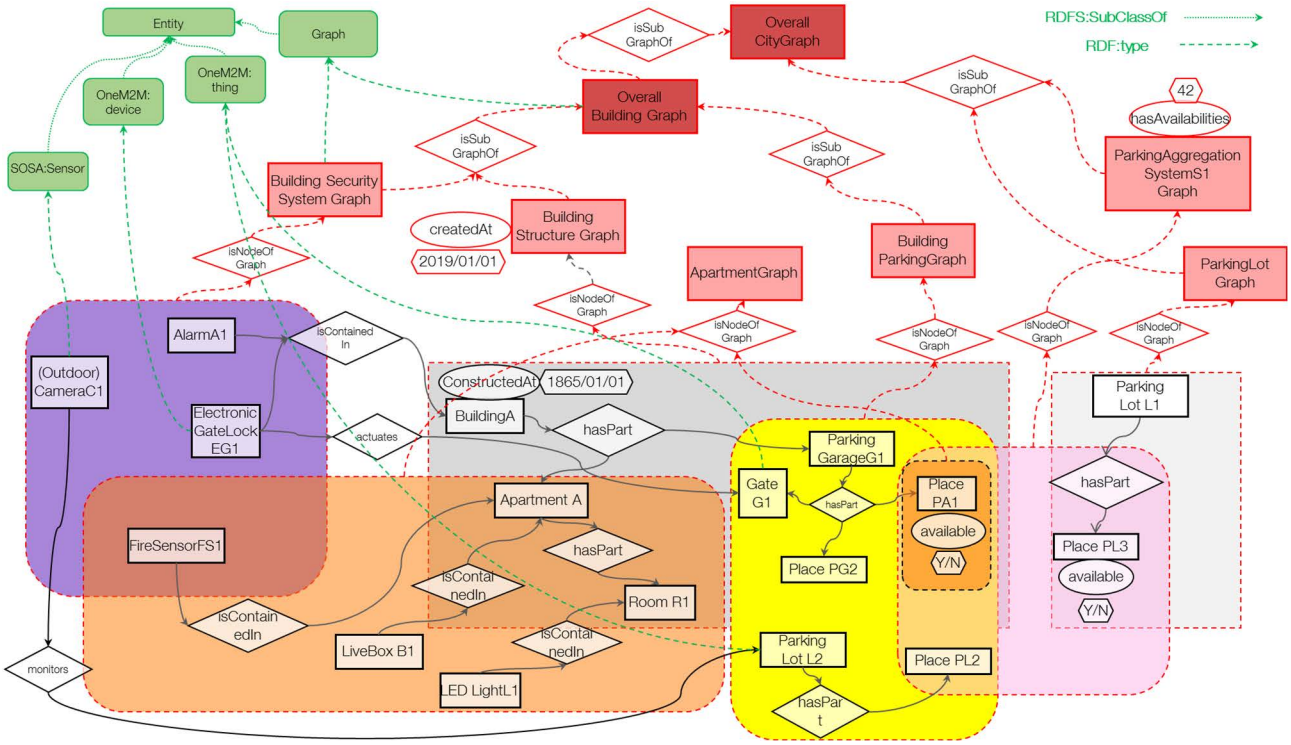


Figure 19: System Clustering Example (2)

The example consists of a building and its parts (using "hasPart" relationships) forming the structure of the building, in addition to other devices that are included in the building. The building has a garage and an apartment. A parking place within the garage belongs to the apartment, thus forming one system together. The building is equipped with a security system containing security devices. Besides, there is a separate public parking that also appears in the example.

The example shows how it is possible to use "**isNodeOfGraph**" to define systems that capture special concerns (e.g. **Building Security Graph** in which case it is desired to separate it from the **Building Structure Graph**), and how nesting/overlap between systems is possible (e.g. **Parking Aggregation System S1** comprise parking places from two different systems: the public parking, and the private parking space from the building).

Annex A (informative): Guidelines for Entity Typing

A.0 Introduction

Typing is used to categorize real-world entities, as represented by NGSI-LD entities, according to supposedly permanent criteria that define these entities against other kinds/species/varieties/makes of entities. NGSI-LD uses types defined as classes in RDF/RDFS/OWL ontologies, corresponding to what is called a "Tbox" in description logic, while the association of entity instances to these classes/types corresponds to the "Abox". These types are a way to attach permanent pieces of "meta-information" to these entities, differently from what is done with the use of regular attributes (properties and relationships) of the corresponding entity instance.

However there is no single universal criterion to choose between those characteristics of entities that are best expressed by typing, and those that are best expressed by normal attributes. Types are just pieces of information, and any typing information could also be expressed with attributes. Consistency, readability, manageability and complexity should be considered when choosing to handle information in either way.

Typing simplifies data input and facilitates checking of data consistency. Without typing, data representation would be much less concise and less easy to understand, because typing avoids replication of pieces of information across all instances of some category of entities that share similar characteristics, precisely because these characteristics may be inherited from the definition of the corresponding classes.

As much as possible, any characteristic that cannot vary from one entity instance to another should be defined through the typing of this instance, whereas all instance-dependent characteristics should be defined by properties and attributes.

For example, the characteristics of a room defined as a kitchen should be defined in its type in as much as they distinguish it from, say, a bathroom, in a generic way. Its area should obviously be defined as a per-instance property, and whether it is adjacent to the living room should obviously be defined by a per-instance relationship. Having said that, there are still many non-obvious modelling choices remaining, typically concerning whether it is useful to define subclasses of the generic kitchen class (e.g. to distinguish between open-space vs traditional kitchens).

In general, it is usually better to use relationships or multiple typing (see following clause) than to define ad hoc classes that might be too specific, too dependent on culture, fashion, etc. In general, using a property with predefined values to capture this kind of subcategorization is not a good idea either (see clause A.2).

A.1 Additional implementation requirements

Typing should not be restricted to assigning only one type to each entity instance. Implementations that support multi-typing (assigning multiple types to a single entity) enhance the repertoire of data modelling and avoid the definition (and potential multiplication *ad infinitum)* of subclasses to characterize different variants of a generic class of entities.

For future versions of the NGSI-LD API it is recommended to consider support for the following improvements in the cross-domain ontology:

- Multi-typing
- Sub-classification (i.e. allowing classes to be subclasses of others)
- Transitivity of sub-classification in order to allow propagating along a list or hierarchy of sub-classifications

Multi-typing example:

A trailer (also known as Recreational Vehicle/Caravan/MotorHome) would be defined with single typing as a separate class of its own, with characteristics that correspond to being both a home space and being mobile at the same time. It would thus replicate definitions of relevant attributes from classes "Vehicle" and "Home". Defining it with dual typing, the definitions are inherited from the two classes rather than having to redefine both of them again inside a new class.

Sub-classification example:

Sub-classification allows, for example to define "Vehicle" as a subclass of "Mobile". In such case every instance that is explicitly typed as "Vehicle" is implicitly also "Mobile" and inherits all typical attributes of a mobile entity, without having to redefine them (e.g. it has a speed and an instantaneous position).

A.2 Modelling recommendations

Instead of associating similar attributes to different entities that belong to a common category, a class can be defined to transfer all these attributes implicitly to all the instances that belong to it. In case of modification of an attribute, there would be one local placeholder to change, the attribute associated with class, instead of changing the attribute in all the instances explicitly.

The following recommendations should help to better structure and handle the data:

- Whenever a class can be defined as a subclass of existing classes, defined in public, curated and widely shared ontologies, this should be done. Further additional restrictions can be added to the class itself after su-classification.
- Classes are not necessarily mutually exclusive, so avoid granularity that makes every single class define a mutually exclusive set of instances. Allowing instances to belong to multiple classes is enough to describe it.
- Whenever properties are used to pigeonhole instances into sub-categories, it is normally preferable to use typing instead. This is often the case when property values are enumerations. For example, instead of using a property to define for each instance of an entity whether it is Mobile, Movable or Stationary, inherit the corresponding entity classes defined in the propose NGSI-LD cross-domain ontology for each of these values rather that assigning them as properties.

A.3 Using OWL/RDFS/RDF modelling

A.3.0 Introduction

This clause is for those who are more familiar with Object-Oriented modelling.

NGSI-LD derives its formal foundation from OWL/RDFS/RDF which are themselves based on description logic. Description-logic-based modelling differs in many ways from object-oriented modelling. Some of these differences are obvious, others less so, and it is important to understand them. A few of these differences are listed below, borrowing partially from a W3C publication [i.6].

A.3.1 OWL/RDFS/RDF modelling

- Classes are just sets of individuals, and each individual can belong to multiple classes.
- Class membership is not static and may be changed (by inference or by explicit typing) at runtime.
- Classes themselves can also be created and changed at runtime.
- Reasoners may be used for (re-)classification and consistency checking at any time.
- An inconsistent and/or incomplete model can still be used, etc.
- Properties in RDF (and relationships in NGSI-LD) are not statically attached to classes and may be added to individuals independently of their class definition.
- Instances can have arbitrary values for any property. Range and domain constraints can be used for type checking and type inference, but not for restrictions.

- Classes defined in a public ontology file (a T-Box) can be referred to from any other class or instance definition anywhere.
- Open World Assumption: If there is not enough information to prove a statement true, then it may be true or false.

A.3.2 Object-Oriented modelling

- Classes are regarded as types that are assigned explicitly as full-fledged templates for instances.
- In many object-oriented languages (not all of them) each instance has only one class as its type, and some languages that allow multiple typing do still place restrictions on it (e.g. the types vs interfaces distinction in Java).
- The list of classes should be defined initially and should be fully known at compile-time; instances cannot change their type dynamically.
- Compilers are used to check consistency of instances with classes, but only at build-time. Compilers prevent the use of inconsistent programs.
- Properties are defined locally to a class (and to all its subclasses through inheritance).
- Range constraints are used for type consistency checking. Values are correctly typed according to the property constraints in the class.
- Classes can encapsulate their members to private access.
- Closed World Assumption: If there is not enough information to prove a statement true, then it is assumed to be false.

Annex B (informative): Relationship to other cross-domain ontologies and upper ontologies

B.0 Introduction

Several existing ontologies, specified by Standard Developing Organizations or other bodies, do overlap partially with the current work. As per received best practices of ontology-based modelling, it is best to avoid redefining ontologies that have already been defined by other groups, especially if they are well established and already widely used by a technical community. The purpose in this annex is to provide cross-referencing between the NGSI-LD ontology and a few of these other ontologies, and to reuse definitions from these, where such opportunity exists. In this way, consistency between the NGSI-LD ontology and these existing ontologies is ensured, and also the consistency of data that would use these ontologies jointly at some level.

Although each ontology definition may have its own motivation - maybe different from the initial motivation of ISG CIM – intersection of interests should be carefully defined. Mapping/alignment between ontologies is a process in which such interests are considered by providing either equivalence relations or sub-classification/super-classification relations between different classes and vocabularies across ontology definitions. Alignment with other ontologies should be done at the correct level (top ontologies, structural ontologies, domain specific ontologies), which define which relations should be used and with which classes.

B.1 Mapping to oneM2M

oneM2M is a partnership project for IoT (originally defined as "machine to machine communication" in the Telecom world). OneM2M provides an OWL ontology that can be partially mapped to the ISG CIM cross-domain ontology, as illustrated in Figure B.1.

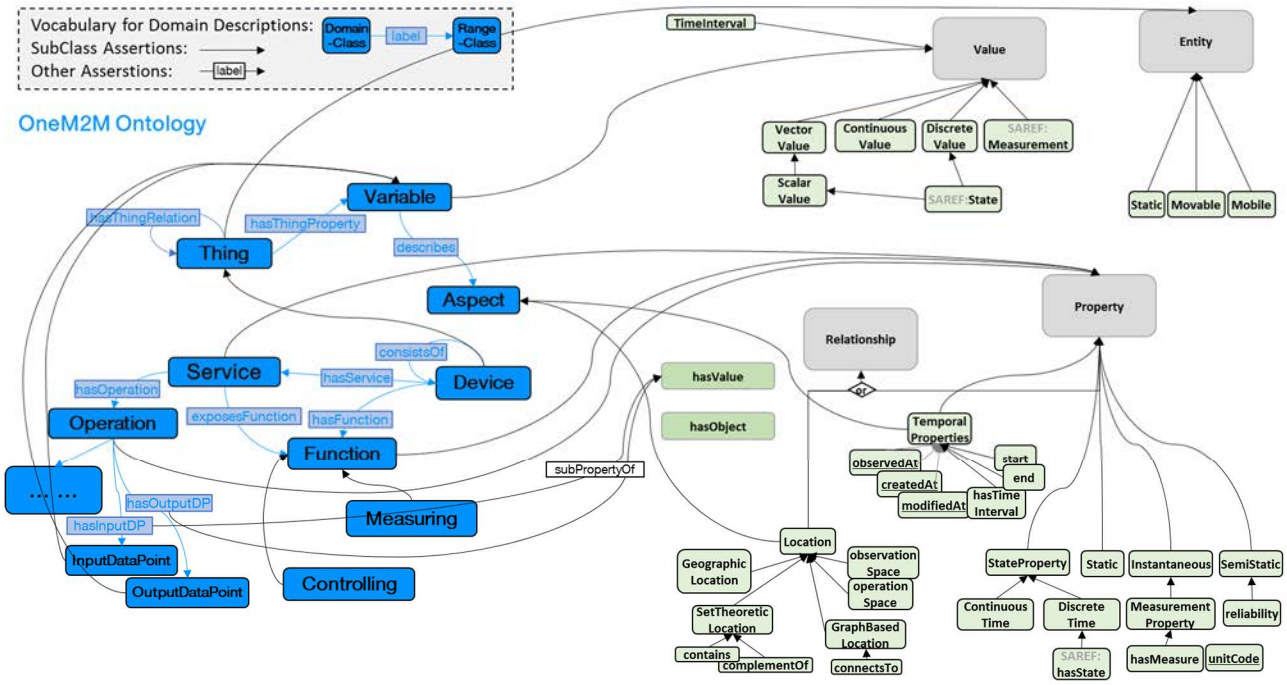


Figure B.1: Mapping NGSI-LD meta-model and cross-domain ontology to oneM2M base ontology

B.2 Mapping to W3C WoT Thing Description

W3C Web of Things (WoT) Thing Description provides both cross-domain and domain-specific vocabularies, and is tentatively mapped to the NGSI-LD cross-domain ontology as represented in Figure B.2.

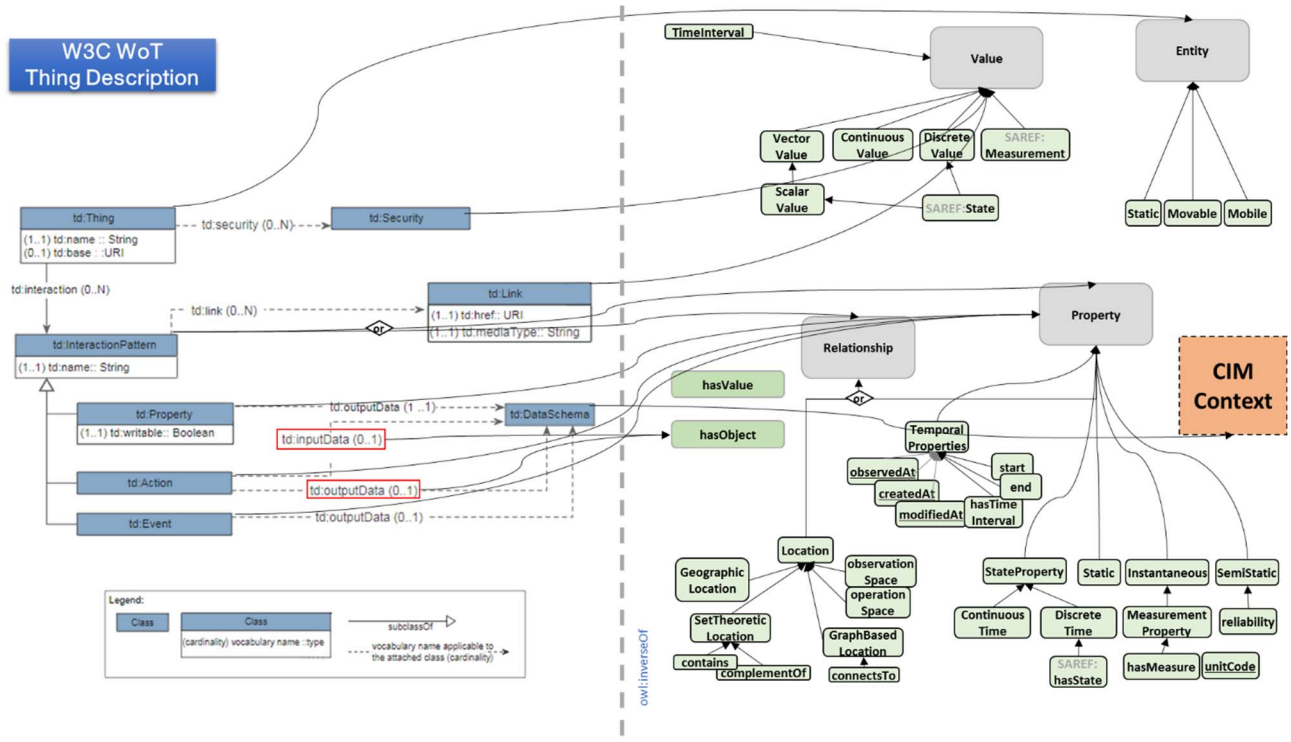


Figure B.2: Mapping NGSI-LD to W3C WoT Thing Description

B.3 Mapping to W3C Time Ontology

The NGSI-LD cross-domain ontology uses a time-specific vocabulary to provide temporal information (either general, or as metadata associated to properties/relationships). Such temporal vocabulary can be taken from W3C Time Ontology rather than redefining them. This also allows interested users/systems who are using W3C Time Ontology to align with the NGSI-LD cross-domain ontology, and can be itself considered as an extension for temporal vocabularies for people who start by using the NGSI-LD ontology.

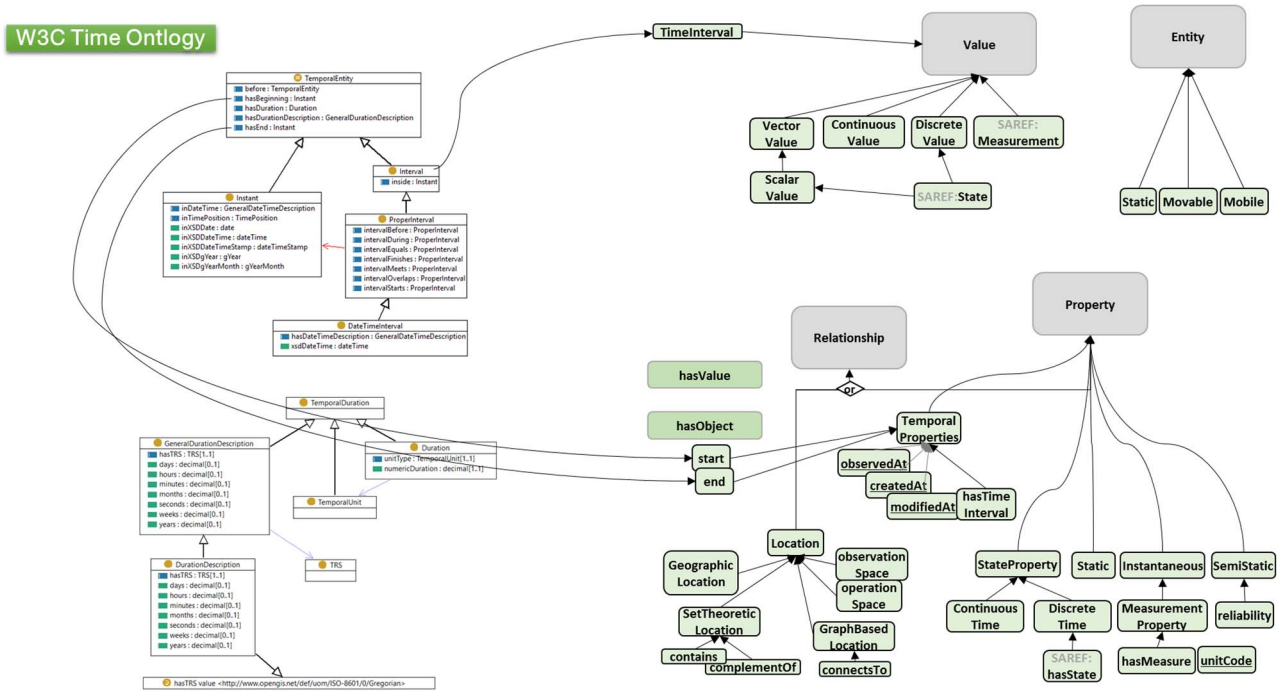


Figure B.3: Mapping NGSI-LD to W3C Time Ontology

B.4 GSMA NGSI-LD-Entities

GSMA NGSI-LD-Entities are a set of data definitions for harmonizing the data from IoT and related context data sources. They have been developed through consultation with mobile operators and industry associated as part of the GSMA IoT project (https://www.gsma.com/iot/iot-big-data/) and are considered as examples, not as a comprehensive set.

The harmonized data models are expected to evolve over time, potentially new entities will be added and entity definitions changed.

The GSMA NGSI-LD-Entities ontology is available at: https://github.com/GSMADeveloper/NGSI-LD-Entities.

B.5 Mapping to SAREF

The SAREF suite of ontologies, defined under the auspices of the ETSI SmartM2M Technical Committee, provide definition of generic classes that can be mapped to the NGSI-LD cross-domain ontology, but they do also further define domain-specific vocabularies in various domains (city, building, agriculture, etc.). Figure B.4 shows the mapping of generic classes of SAREF to the NGSI-LD cross-domain ontology.

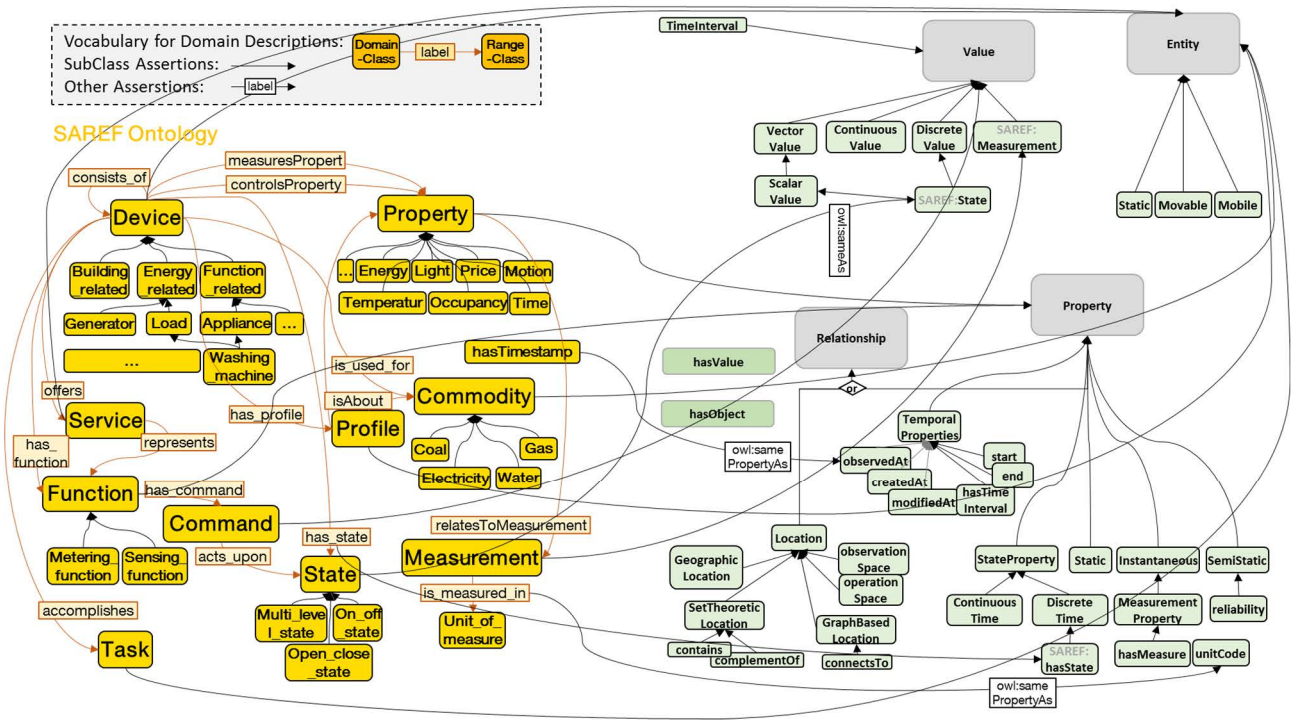


Figure B.4: Mapping NGSI-LD to SAREF

Annex C (informative): Distinguishing real-word entities from NGSI-LD entities

C.0 Introduction

In NGSI-LD, both attributes (relationships and properties) and semantic referencing by way of classes are used to assign information to real-world entities that are represented by NGSI-LD informational entities, so that these NGSI-LD entities are just informational proxies/representatives of the real-world "stuff" they stand for. In most cases, this is so obvious that it need not even be mentioned: for example the location attribute of an entity always corresponds to a real-world location of a real-world entity, NOT to e.g. the internet location defining the hosting of the corresponding NGSI-LD entity by some server, etc. In a few cases, the distinction is much less obvious, and it should be made clear and explicit. For example attributes like "createdAt" or "modifiedAt" could be misunderstood to apply either to real-world entities OR to their informational NGSI-LD representation as maintained in a platform.

C.1 W3C View

In RDF and linked data in general, a URI is used to identify a resource that may be a purely informational resource, or a real world physical thing, or a person, a living being, or something in between, like a legal entity, what is technically called a "Non-Informational Resource" (NIR). In principle Alice *as a physical person* should be identified by a different URI from the address of her home page (in web1.0 style), or the address of the profile of Alice on a social network (both of the latter being purely informational resources). Bob may like or dislike Alice, but this is a totally distinct piece of information than saying he likes or dislikes her web page of some social network profile of hers. The distinction between these two different types of resources is a recurring theme of discussion [i.7] that has been recognized as crucial by the W3C. The W3C has advised using either of two potential solutions [i.8]:

1) The first method is to use hash URIs (i.e. URIs with fragment identifiers, like http://Alice_home_page.example.org#Alice) for non-informational resources. URIs without "#", like http://Alice_home_page.example.org, should be understood as referring to informational resources.

2) The second method is to follow a simple http rule. If an "http" resource responds to a GET request with a 2xx response, then the resource identified by that URI is an information resource; if it responds with a 303 (See Other) response, then the resource identified by that URI could be a non-informational resource.

C.2 NGSI-LD View

Some properties in the NGSI-LD API have specifically been defined such that they are never to be reified and do not use the "hasValue" construct. Those properties include: *observedAt*, *createdAt*, *modifiedAt*, *start*, *end* and *unitCode*. Those properties are direct instances (using *rdf:type*) of the class **Property.**

Consider the following explanatory example:

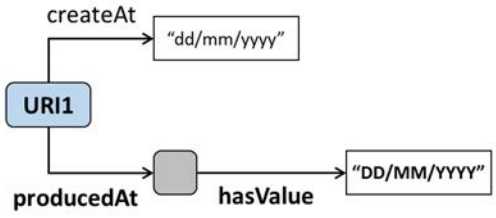


Figure C.1: Example of different kinds of information representation

The two "createdAt" and "producedAt" properties might seem to convey similar ideas, yet "createdAt" is a cross-domain property that applies to the creation of a record in some database i.e. has a special use in the NGSI-LD information model as a non-reifiable property and thus a purely informational descriptor, while "producedAt" is understood to apply to the actual physical production of the real world "thing" that the NGSI-LD entity identified by URI1 stands for, so that "producedAt" is a domain-specific property that is reified by attaching a blank node to its object (shown in bold in the figure).

Annex D (informative): OWL-DL representation of the Information Model

```
@prefix : <https://uri.etsi.org/ngsi-ld/v1/ontology#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@base <https://uri.etsi.org/ngsi-ld/v1/ontology> .
<https://uri.etsi.org/ngsi-ld/v1/ontology> rdf:type owl:Ontology .

#################################################################
#    Object Properties
#################################################################

###  https://uri.etsi.org/ngsi-ld/v1/ontology#ContinuousTime
:ContinuousTime rdf:type owl:ObjectProperty ;
                rdfs:subPropertyOf :StateProperty ;
                rdfs:range :ContinuousValue .


###  https://uri.etsi.org/ngsi-ld/v1/ontology#CoordinateBasedLocation
:CoordinateBasedLocation rdf:type owl:ObjectProperty ;
                         rdfs:subPropertyOf :LocationProperty .


###  https://uri.etsi.org/ngsi-ld/v1/ontology#DiscreteTime
:DiscreteTime rdf:type owl:ObjectProperty ;
              rdfs:subPropertyOf :StateProperty ;
              rdfs:range [ rdf:type owl:Class ;
                           owl:unionOf ( :ContinuousValue
                                         :DiscreteValue
                                       )
                         ] .


###  https://uri.etsi.org/ngsi-ld/v1/ontology#GeoProperty
:GeoProperty rdf:type owl:ObjectProperty ;
             rdfs:subPropertyOf :CoordinateBasedLocation ;
             rdfs:range :Geometry .


###  https://uri.etsi.org/ngsi-ld/v1/ontology#GraphBasedLocation
:GraphBasedLocation rdf:type owl:ObjectProperty ;
                    rdfs:subPropertyOf :LocationRelationship .


###  https://uri.etsi.org/ngsi-ld/v1/ontology#Instantaneous
:Instantaneous rdf:type owl:ObjectProperty ;
               rdfs:subPropertyOf :Property ;
               rdfs:range :Instantaneous .


###  https://uri.etsi.org/ngsi-ld/v1/ontology#LocationProperty
:LocationProperty rdf:type owl:ObjectProperty ;
                  rdfs:subPropertyOf :Property .


###  https://uri.etsi.org/ngsi-ld/v1/ontology#LocationRelationship
:LocationRelationship rdf:type owl:ObjectProperty ;
                      rdfs:subPropertyOf :Relationship .


###  https://uri.etsi.org/ngsi-ld/v1/ontology#MeasurementProperty
```

```
:MeasurementProperty rdf:type owl:ObjectProperty ;
                     rdfs:subPropertyOf :Instantaneous .


###  https://uri.etsi.org/ngsi-ld/v1/ontology#Property
:Property rdf:type owl:ObjectProperty ;
          rdfs:range [ rdf:type owl:Class ;
                       owl:unionOf ( :Property
                                     :Value
                                   )
                     ] .


###  https://uri.etsi.org/ngsi-ld/v1/ontology#Relationship
:Relationship rdf:type owl:ObjectProperty ;
              rdfs:domain :Entity ;
              rdfs:range [ rdf:type owl:Class ;
                           owl:unionOf ( :Entity
                                         :Relationship
                                       )
                         ] .


###  https://uri.etsi.org/ngsi-ld/v1/ontology#SemiStatic
:SemiStatic rdf:type owl:ObjectProperty ;
            rdfs:subPropertyOf :Property ;
            rdfs:range :SemiStatic .


###  https://uri.etsi.org/ngsi-ld/v1/ontology#SetBasedLocation
:SetBasedLocation rdf:type owl:ObjectProperty ;
                  rdfs:subPropertyOf :LocationRelationship .


###  https://uri.etsi.org/ngsi-ld/v1/ontology#StateProperty
:StateProperty rdf:type owl:ObjectProperty ;
               rdfs:subPropertyOf :Property .


###  https://uri.etsi.org/ngsi-ld/v1/ontology#Static
:Static rdf:type owl:ObjectProperty ;
        rdfs:subPropertyOf :Property ;
        rdfs:range :Static .


###  https://uri.etsi.org/ngsi-ld/v1/ontology#connectsTo
:connectsTo rdf:type owl:ObjectProperty ;
            rdfs:subPropertyOf :GraphBasedLocation ;
            rdf:type owl:TransitiveProperty ;
            owl:propertyChainAxiom ( :connectsTo
                                     :hasObject
                                   ) .


###  https://uri.etsi.org/ngsi-ld/v1/ontology#hasDirectPart
:hasDirectPart rdf:type owl:ObjectProperty ;
               rdfs:subPropertyOf :hasPart ;
               owl:propertyChainAxiom ( :hasDirectPart
                                        :hasObject
                                      ) .


###  https://uri.etsi.org/ngsi-ld/v1/ontology#hasMeasure
:hasMeasure rdf:type owl:ObjectProperty ;
            rdfs:subPropertyOf :MeasurementProperty ;
            rdfs:range :Measurement .
```

```
###  https://uri.etsi.org/ngsi-ld/v1/ontology#hasObject
:hasObject rdf:type owl:ObjectProperty ;
           rdfs:domain :Relationship ;
           rdfs:range :Entity .


###  https://uri.etsi.org/ngsi-ld/v1/ontology#hasObservationZone
:hasObservationZone rdf:type owl:ObjectProperty ;
                    rdfs:subPropertyOf :LocationRelationship ;
                    rdfs:range [ rdf:type owl:Class ;
                                 owl:unionOf ( :Relationship
                                               :Zone
                                             )
                               ] ;
                    owl:propertyChainAxiom ( :hasObservationZone
                                             :hasObject
                                           ) .


###  https://uri.etsi.org/ngsi-ld/v1/ontology#hasOpertationZone
:hasOpertationZone rdf:type owl:ObjectProperty ;
                   rdfs:subPropertyOf :LocationRelationship ;
                   rdfs:range [ rdf:type owl:Class ;
                                owl:unionOf ( :Relationship
                                              :Zone
                                            )
                              ] ;
                   owl:propertyChainAxiom ( :hasOpertationZone
                                            :hasObject
                                          ) .


###  https://uri.etsi.org/ngsi-ld/v1/ontology#hasPart
:hasPart rdf:type owl:ObjectProperty ;
         rdfs:subPropertyOf :Relationship ;
         rdf:type owl:TransitiveProperty ;
         owl:propertyChainAxiom ( :hasPart
                                  :hasObject
                                ) .


###  https://uri.etsi.org/ngsi-ld/v1/ontology#hasState
:hasState rdf:type owl:ObjectProperty ;
          rdfs:subPropertyOf :StateProperty ;
          rdfs:range :State .


###  https://uri.etsi.org/ngsi-ld/v1/ontology#isContainedIn
:isContainedIn rdf:type owl:ObjectProperty ;
               rdfs:subPropertyOf :SetBasedLocation ;
               rdf:type owl:TransitiveProperty ;
               owl:propertyChainAxiom ( :isContainedIn
                                        :hasObject
                                      ) .


###  https://uri.etsi.org/ngsi-ld/v1/ontology#isNodeOfGraph
:isNodeOfGraph rdf:type owl:ObjectProperty ;
               rdfs:subPropertyOf :Relationship ;
               rdfs:domain :Entity ;
               rdfs:range :Graph ;
               owl:propertyChainAxiom ( :isNodeOfGraph
                                        :isSubGraphOf
                                      ) .


###  https://uri.etsi.org/ngsi-ld/v1/ontology#isSubGraphOf
:isSubGraphOf rdf:type owl:ObjectProperty ;
```

```
              rdfs:subPropertyOf :Relationship ;
              rdf:type owl:TransitiveProperty ;
              rdfs:domain :Graph ;
              rdfs:range :Graph .


###  https://uri.etsi.org/ngsi-ld/v1/ontology#location
:location rdf:type owl:ObjectProperty ;
          rdfs:subPropertyOf :GeoProperty .


###  https://uri.etsi.org/ngsi-ld/v1/ontology#observationSpace
:observationSpace rdf:type owl:ObjectProperty ;
                  rdfs:subPropertyOf :GeoProperty .


###  https://uri.etsi.org/ngsi-ld/v1/ontology#operationSpace
:operationSpace rdf:type owl:ObjectProperty ;
                rdfs:subPropertyOf :GeoProperty .


#################################################################
#    Data properties
#################################################################

###  https://uri.etsi.org/ngsi-ld/v1/ontology#TemporalProperty
:TemporalProperty rdf:type owl:DatatypeProperty .


###  https://uri.etsi.org/ngsi-ld/v1/ontology#createdAt
:createdAt rdf:type owl:DatatypeProperty ;
           rdfs:subPropertyOf :TemporalProperty ;
           rdfs:range xsd:dateTime .


###  https://uri.etsi.org/ngsi-ld/v1/ontology#hasValue
:hasValue rdf:type owl:DatatypeProperty ;
          rdfs:domain :Property .


###  https://uri.etsi.org/ngsi-ld/v1/ontology#modifiedAt
:modifiedAt rdf:type owl:DatatypeProperty ;
            rdfs:subPropertyOf :TemporalProperty ;
            rdfs:range xsd:dateTime .


###  https://uri.etsi.org/ngsi-ld/v1/ontology#observedAt
:observedAt rdf:type owl:DatatypeProperty ;
            rdfs:subPropertyOf :TemporalProperty ;
            rdfs:range xsd:dateTime .


###  https://uri.etsi.org/ngsi-ld/v1/ontology#unitCode
:unitCode rdf:type owl:DatatypeProperty ;
          rdfs:range xsd:string .


#################################################################
#    Classes
#################################################################

###  https://uri.etsi.org/ngsi-ld/v1/ontology#ContinuousValue
:ContinuousValue rdf:type owl:Class ;
                 rdfs:subClassOf :State .


###  https://uri.etsi.org/ngsi-ld/v1/ontology#DiscreteValue
:DiscreteValue rdf:type owl:Class ;
```

```
               rdfs:subClassOf :State .


###  https://uri.etsi.org/ngsi-ld/v1/ontology#Entity
:Entity rdf:type owl:Class ;
        owl:disjointWith :Property ,
                         :Relationship ,
                         :Value .


###  https://uri.etsi.org/ngsi-ld/v1/ontology#Geometry
:Geometry rdf:type owl:Class ;
          rdfs:subClassOf :Value .


###  https://uri.etsi.org/ngsi-ld/v1/ontology#Graph
:Graph rdf:type owl:Class ;
       rdfs:subClassOf :Entity .


###  https://uri.etsi.org/ngsi-ld/v1/ontology#Instantaneous
:Instantaneous rdf:type owl:Class ;
               rdfs:subClassOf :Property .


###  https://uri.etsi.org/ngsi-ld/v1/ontology#LineString
:LineString rdf:type owl:Class ;
            rdfs:subClassOf :Geometry .


###  https://uri.etsi.org/ngsi-ld/v1/ontology#Measurement
:Measurement rdf:type owl:Class ;
             rdfs:subClassOf :Value .


###  https://uri.etsi.org/ngsi-ld/v1/ontology#Mobile
:Mobile rdf:type owl:Class ;
        owl:equivalentClass [ rdf:type owl:Restriction ;
                              owl:onProperty :location ;
                              owl:allValuesFrom :Instantaneous
                            ] ;
        rdfs:subClassOf :Entity .


###  https://uri.etsi.org/ngsi-ld/v1/ontology#Movable
:Movable rdf:type owl:Class ;
         owl:equivalentClass [ rdf:type owl:Restriction ;
                               owl:onProperty :location ;
                               owl:allValuesFrom :SemiStatic
                             ] ;
         rdfs:subClassOf :Entity .


###  https://uri.etsi.org/ngsi-ld/v1/ontology#Point
:Point rdf:type owl:Class ;
       rdfs:subClassOf :Geometry .


###  https://uri.etsi.org/ngsi-ld/v1/ontology#Polygon
:Polygon rdf:type owl:Class ;
         rdfs:subClassOf :Geometry .


###  https://uri.etsi.org/ngsi-ld/v1/ontology#Property
:Property rdf:type owl:Class ;
          owl:disjointWith :Relationship .
```

```
###  https://uri.etsi.org/ngsi-ld/v1/ontology#Relationship
:Relationship rdf:type owl:Class ;
              owl:disjointWith :Value .


###  https://uri.etsi.org/ngsi-ld/v1/ontology#ScalarValue
:ScalarValue rdf:type owl:Class ;
             rdfs:subClassOf :VectorValue .


###  https://uri.etsi.org/ngsi-ld/v1/ontology#SemiStatic
:SemiStatic rdf:type owl:Class ;
            rdfs:subClassOf :Property .


###  https://uri.etsi.org/ngsi-ld/v1/ontology#State
:State rdf:type owl:Class ;
       rdfs:subClassOf :Value .


###  https://uri.etsi.org/ngsi-ld/v1/ontology#Static
:Static rdf:type owl:Class ;
        rdfs:subClassOf :Property .


###  https://uri.etsi.org/ngsi-ld/v1/ontology#Stationary
:Stationary rdf:type owl:Class ;
            owl:equivalentClass [ rdf:type owl:Restriction ;
                                  owl:onProperty :location ;
                                  owl:allValuesFrom :Static
                                ] ;
            rdfs:subClassOf :Entity .


###  https://uri.etsi.org/ngsi-ld/v1/ontology#Value
:Value rdf:type owl:Class .


###  https://uri.etsi.org/ngsi-ld/v1/ontology#VectorValue
:VectorValue rdf:type owl:Class ;
             rdfs:subClassOf :Value .


###  https://uri.etsi.org/ngsi-ld/v1/ontology#Zone
:Zone rdf:type owl:Class ;
      rdfs:subClassOf :Entity .
```

Annex E (informative): Change History

Date	Version	Information about changes
16 April 2018	V0.0.1	ToC agreed in ISG CIM ordinary call
7 May 2019	V0.0.2	Early Draft reviewed by ETSI and Lindsay Frost This is the date of the change of the template with guidance : Latest changes made on 2019-01-08 https://portal.etsi.org/Portals/0/TBpages/edithelp/Docs/ETSI_GS_skeleton_with_guidance_text.docx just the skeleton : Latest changes made on 2019-01-08 https://portal.etsi.org/Portals/0/TBpages/edithelp/Docs/ETSI_GS_skeleton.docx
17 May 2019	V0.0.3	Removed unused informative references. Corrected typos
8 July 2019	V1.0.0	ETSI Technical Officer formatting & editing rule checking
February 2021	V1.1.2	Diverse revisions
March 2023	V1.1.3	Figure revision and minor text revisions
May 2023	V1.2.1	Technical Officer review for ***editHelp!*** publication pre-processing

History

Document history		
V1.1.1	July 2019	Publication
V1.2.1	June 2023	Publication