ETSI White Paper No. 42

Guidelines for Modelling with NGSI-LD

1st edition – March 2021

ISBN No. 979-10-92620-36-6

Main Author/Editor:
Gilles Privat, Orange labs

ETSI
06921 Sophia Antipolis CEDEX, France
Tel +33 4 92 94 42 00
info@etsi.org
www.etsi.org

Contributors

Ahmed Abid, EGM
Alexey Medvedev, Deakin University, Australia
Alireza Hassani. Deakin University, Australia
Franck Le Gall, EGM
Giuseppe Tropea, CNIT
Juan Antonio Martinez, University of Murcia
Lindsay Frost, NEC Labs
Martin Bauer, NEC Labs

Acknowledgement of supporting EU research programmes, current and past, is gladly made to: Project AutoPilot Grant ID 731993, Project CPaaS.io Grant ID 723076, Project Fed4IoT Grant ID 814918, Project Fiware4Water Grant ID 821036, Project iFishIENCi Grant ID 818036, Project IMPAQT Grant ID 774109, Project IoF2020 Grant ID 731884, Project IoT-Crawler Grant ID 779852, Project MIDIH Grant ID 767498, Project SynchroniCity Grant ID 732240.

Contents

Executive Summary

This ETSI White Paper is intended to complement the NGSI-LD information model normative specification (see ETSI GS CIM 006 V1.1.1 [1]). It provides a set of practical guidelines on how to model a domain-specific system, process, or environment, how to associate entity instances to types/classes, how to use relationships and properties. These guidelines are based on both the NGSI-LD meta-model and the NGSI-LD cross-domain ontology as a common denominator set of classes cutting across domain-specific ontologies and taxonomies.

This White Paper is also intended to be complementary to the NGSI-LD Primer (see ETSI GR CIM 008 [2]), which mainly explains how to use the NGSI-LD API (see ETSI GS CIM 009 [3]) (e.g. creating or updating entity instances, queries, subscriptions, etc.) in a more "hands-on" practical way, especially for the initial use cases considered(see ETSI GR IM 002 [4]). The main body of the present document limits the use of the NGSI-LD information model to those features that are compatible with the NGSI-LD API. The use of features of the information model that are not (yet) supported by the NGSI-LD API is explained in the Annex.

The intended reader categories, not prioritized, are:

1. software architects/developers with a background in object-oriented programming/modelling and/or UML
2. software architects/developers with a background in relational databases and classical entity/relationship modelling
3. data/information scientists and software practitioners in the fields of big data, data lakes, deep-learning-style AI
4. Information scientists with a background in classical knowledge representation with linked data, RDF-based models, description logics, first-order logic, reasoning, "classical" (symbolic) AI

This White Paper explains how NGSI-LD modelling relates to the modelling approaches familiar to these professionals and how it differs, at both practical and theoretical levels. This document is NOT an "NGSI-LD for dummies". It requires prior general knowledge in the above-mentioned reference fields.

1 A quick summary of the NGSI-LD information model

1.1 Property Graph Meta-model

The NGSI-LD information model is derived from Property Graphs (PGs in the following), which have emerged as the common-denominator data-model of most graph databases since they became popular in the mid-2000s.

PGs (a.k.a. attributed graphs, see "RDF versus attributed graphs: The war for the best graph representation", 5) are usually informally defined as follows:

- A PG is a directed graph, made up of nodes (vertices) connected by unidirectional arcs (i.e. directed edges)
- Nodes and arcs both may have multiple optional attached properties (i.e. attributes)
- Properties (similar to attributes in object models) have the form of arbitrary key-value pairs. Keys are strings and values are arbitrary data types. By contrast to RDF graphs, properties are not arcs of the graph[1].
- A relationship is an arc [2] of the graph, which always has an identifier, a start node, and an end node.

The NGSI-LD meta-model formally defines the following NGSI-LD foundational concepts (Entities, Relationships, Properties) on the basis of RDF/RDFS/OWL, and partially on the basis of JSON-LD (see JSON-LD 1.1, A JSON-based Serialization for Linked Data : https://www.w3.org/TR/2020/CR-json-ld11-20200316/):

- *An NGSI-LD Entity* is the informational representative of something (a *referent*) that is supposed to exist in the real world, outside of the NGSI-LD platform. This referent need not be something strictly physical (it could be a legal or administrative entity), or self-contained (it may be a distributed system-level construct). Any instance of such an entity is supposed to be uniquely identified by a URI, and characterized by reference to one or more NGSI-LD Entity Type(s). In property-graph language, it is a node.
- *An NGSI-LD Property* is an instance that associates a characteristic, an NGSI-LD Value, to either an NGSI-LD Entity, an NGSI-LD Relationship, or another NGSI-LD Property.
- *An NGSI-LD Relationship* is a directed[3] link between a subject (starting point), that may be an NGSI-LD Entity, an NGSI-LD Property, or another NGSI-LD Relationship, and an object (end-point), that is an NGSI-LD Entity.

[1] This may appear to contradict the name itself, which should thus not be understood as "graph of properties", but "graph with (attached) properties", or "propertied graphs", to parallel the phrase "attributed graphs".

[2] "edge" is often used here in place of "arc", yet in graph theory (from pure mathematics) the two terms are NOT interchangeable : an edge connects vertices in an *undirected* graph, whereas an arc belongs in a *directed* graph (arcs have an origin node and an end node, whereas edges are symmetrical). A graph, as defined in pure mathematics, cannot mix edges and arcs, because directed and undirected graphs are two very different kinds of mathematical abstractions (if V is the set of vertices, a directed graph is a subset of V_xV, whereas an undirected graph is as subset of $P_2(V)$)

[3] An NGSI-LD relationship is directed by default. If it is meant to stand for a symmetric type of link, it must be instantiated as two arcs of opposite directions. Note again that this is aligned with the strong distinction made in

- *An NGSI-LD value* is a JSON value (i.e. a string, a number, true or false, an object, an array), *or* a JSON-LD typed value 6 (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).
- *An NGSI-LD type* is an OWL class that is a subclass of either the NGSI-LD Entity, NGSI-LD Relationship, NGSI-LD Property or NGSI-LD Value classes defined in the NGSI-LD meta-model

1.2 Cross-domain ontology[4]

The NGSI-LD cross-domain ontology, built upon the NGSI-LD meta model specified above, defines in turn a set of generic NGSI-LD entity types, relationship types and property types (formal concepts and constructs, with associated constraints) intended to serve as common terms between domain-specific models, addressing the generic temporal and structural description of physical systems across these domains. The cross-domain ontology defines sets of primitive concepts relating to temporality, mobility, system states and system composition, as listed in the following subsections. The elements in figure 1 below are explained in the following text.

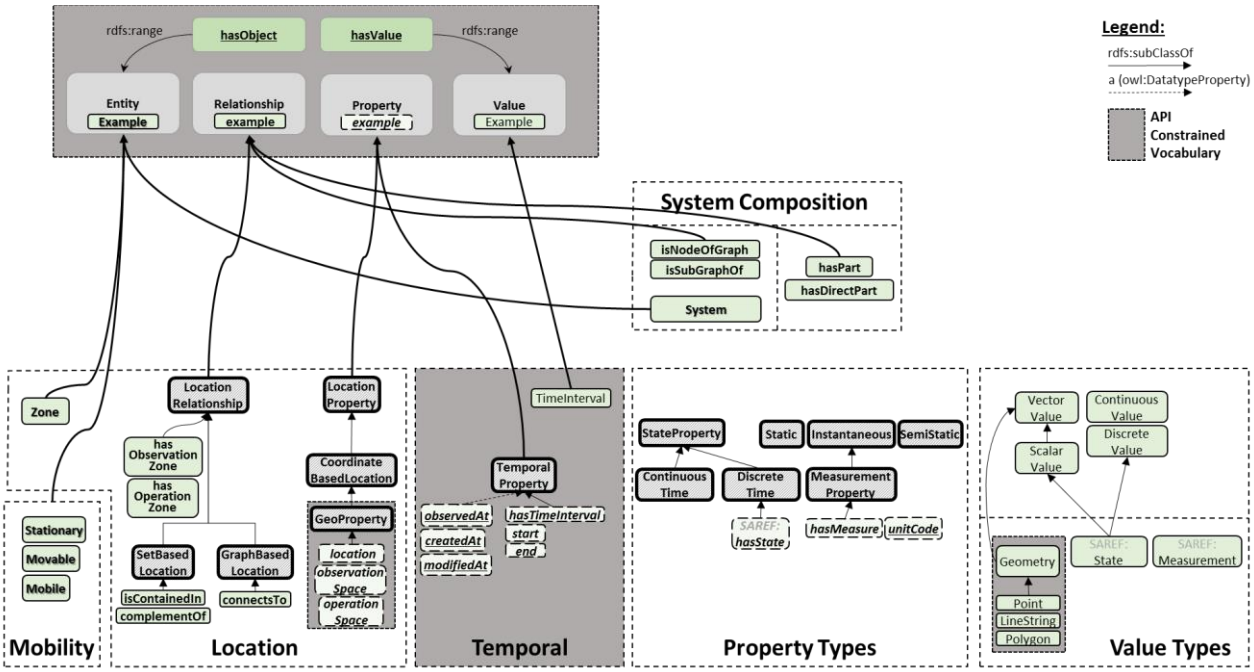


Figure 1 : NGSI-LD Cross-Domain Ontology, with referenced Meta-model[5]

pure mathematics between directed and undirected graphs, neither of which may mix arcs and edges because they are entirely different mathematical constructs : arcs are couples (dyadic tuples) drawn from the Cartesian product VxV of the vertex set, whereas edges are dyadic sets drawn from $P_2(V)$. By contrast, some graph databases may mix directed and undirected relationships, and the distinction is normally rooted in the type of the corresponding relationship.

[4] See ETSI GS CIM 006 V1.1.1 [1] for complete specification

[5] From Figure 11 in ETSI GS CIM 006 V1.1.1 [1]

1.2.1 Temporal concepts (from section 4.8 of ETSI GS CIM 009 [3])

- observedAt
  - *Time at which a certain Property or Relationship became valid or was observed. For example, a temperature Value was measured by the sensor at this point in time.*
- createdAt
  - *Time at which the Entity, Property or Relationship was entered into an NGSI-LD system (this is usually NOT the time at which the referent Entity itself was created!)*
- modifiedAt
  - *Time at which an Entity, Property or Relationship was last modified in an NGSI-LD system, e.g. in order to correct or supersede a previously entered value.*
- MeasurementProperty

  Static
  - *Does NOT vary AT ALL, e.g. the position of a building*

  SemiStatic
  - *May vary from time to time, e.g. position of a piece of furniture*

  Instantaneous
  - *Varies constantly, e.g. the position of a smartphone*

1.2.2 Mobility classes

- Stationary
  - *Entity with static position*
- Movable
  - *Entity with semi-static position*
- Mobile
  - *Entity with instantaneous position that normally varies constantly*

1.2.3 Location classes

- Location

  hasOperationZone
  - *a location relationship, the source of which is typically an actuator, the target of which is a zone, i.e. an entity that delimits its own space (e.g. a room, a city)*

  hasObservationZone
  - *a location relationship, the source of which is typically a sensor , the target of which is a zone*
- GraphBasedLocation

  connectsTo
  - *a relationship between 2 entities corresponding to a physical junction or link through a network, or a physically-based connection, e.g. a room and a corridor connected by a door, or two crossing connected by a street.*
- SetBasedLocation

  isContainedIn
  - *relationship between two entities that may capture either a rough notion of location (e.g. a piece of furniture being inside a room, without further precision)*

  complementOf
  - *relationship between two entities, whereby is stated that one is NOT contained in the other, which means it is contained in the complement of the target entity with regard to a reference space.*
- CoordinatesBasedLocation

  observationSpace

- *same as operationZone, only it is a property instead of a relationship, because the target is a geometry (complex value) instead of an entity*

operationSpace

- *same as operationZone, only it is a property instead of a relationship, because the target is a geometry (complex value) instead of an entity*

1.2.4 System state description classes (from section 6.3.2 of ETSI GS CIM 006 V1.1.1 [1])

- StateProperty
  - *Instantaneous property of an entity, which 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*

  ContinuousTime
  - *Corresponds to system states which may vary continuously in time, such as e.g. the level of water in a water tank*

  DiscreteTime
  - *Corresponds to system states whose variations are triggered by discrete events occurring asynchronously in general (like e.g. trucks offloading their contents in a warehouse), or synchronously (like a computer register being updated on periodic clock cycles)*
- State Values

  ContinuousValue
  - *A state variable which may vary continuously, usually represented by a floating-point number, like e.g. the filling level of a water tank represented as a percentage ; may correspond to a either a continuous-time or discrete-time variable*

  DiscreteValue
  - *A state variable which may vary only in discrete increments, like e.g. the number of persons in a room ; may correspond only to a discrete-time variable*

  VectorValue/ScalarValue
  - *A vector-valued state has several homogeneous components of the same type, like e.g. the 3 spatial coordinates of a mobile object. Individual scalar state-variables of a different nature are, together with vector-valued variables, part of the complete state of a system, which is always multidimensional and inhomogeneous in this sense (like e.g. comprising the speed and the position of a mobile object)*

1.2.5 System composition classes (from 6.3.6.2 in ETSI GS CIM 006 V1.1.1 [1])

- System
  - *An entity that is described in NGSI-LD as a white box, i.e. with a further decomposition into component sub-entities captured with their relationships through other nodes of a subgraph (typically with at least a few hasPart/hasDirectPart relationships). A system node acts as the "root" node of the corresponding subgraph[6]. A black box or grey box artefact captured through a single node (even if it has a set of properties) would not be characterized as a system in the NGSI-LD sense , even if it is a system from an engineering viewpoint*
- Graph
  - *This is the superclass of hypernodes, special entities that stand for a large-scale subgraph, which typically stands for a distributed system or system of systems, such as a physical network*

[6] In graph-theoretic terms this subgraph need not be a rooted tree, most of the time it will include transversal relationships between the branches

- isNodeOfgraph
  - *relationship between a node and a hypernode, the latter standing for a distributed system or system of systems incorporating the former as a component or subsystem*
- isSubGraphOf
  - *relationship between a hypernode and another hypernode, the latter standing for a distributed system or system of systems incorporating the former as a subsystem*
- hasPart
  - *relationship between two entities, the latter being a permanent and mandatory component of the former system and having been designed especially for this purpose*
- hasDirectpart
  - *relationship between two entities, the latter being a permanent and mandatory primary (i.e. without any intervening subsystem) component of the former system and having been designed especially for this purpose,*
- isContainedIn
  - *relationship between two entities, the former typically being a non-permanent and non-mandatory component of the former system, and having not been designed especially for this purpose ; may also denote a simple set-based location relationship, without any component-to-system meaning*

1.3 Domain-specific ontologies

Domain-specific ontologies (see Figure 2) define complete sets of classes/concepts so as to improve interoperability and efficiency within a given domain such as e.g. buildings, cities, industry, or generic sub-domains thereof (e.g. the suite of extensions defined by SAREF [7] for different domains in SAREF4ENER, SAREF SAREF4ENVI, SAREF4BLDG, SAREF4CITY, SAREF4INMA, SAREF4AGRI, and others under development). They need not be full-fledged OWL ontologies and may be just thesauri (such as AGROVOC thesaurus standardized by FAO for agriculture) or taxonomies/controlled vocabularies (such as schema.org). Users are encouraged to reuse appropriate domain-specific ontologies when modelling. These ontologies are, however, *not* part of the NGSI-LD specification and are not directly addressed in the main part of this document. Caution should be exercised, whenever they are used jointly with the cross-domain ontology defined in the specification, because some of the topmost classes of these domain-specific ontologies may not be aligned with identically-named generic types of the NGSI-LD cross-domain ontology, which may define the same concepts in a more general way. For example the notion of "state" may be defined more narrowly in a domain-specific ontology than is the case in the NGSI-LD cross-domain ontology. Alignment of these ontologies should, if possible, be undertaken, but that is beyond the scope of this document.

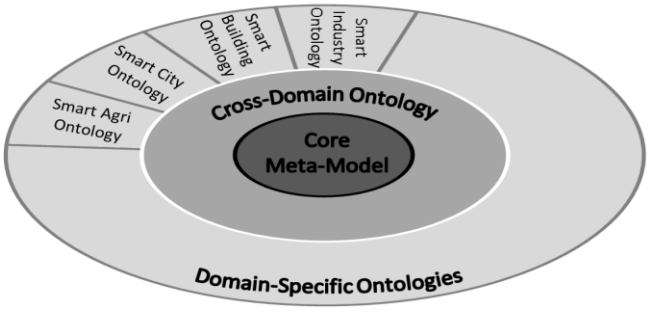


Figure 2: Example domain-specific ontologies complementing the cross-domain ontology

The current document addresses modelling of a domain *instance (*e.g. a specific building, a specific city) primarily at the individual level, whether those ontologies address exclusively the class/concept level. Such instance models should, however, always be *shareable*. The use of generic types/classes/concepts drawn from both generic and domain-specific ontologies, is meant to support this shareability, and this is the whole point of ontological modelling.

2 Guidelines for Typing

2.1 Introduction

Typing, as used by NGSI-LD, applies to entities, relationships, properties, and values. It is based on ontological modelling as standardised since the early 2000s by the semantic web project (see Berners-Lee et al [8]) on the basis of earlier work on RDF and description logic). This modelling differs in subtle ways from the typing used in e.g. object-oriented modelling, or other programming language typing styles. The notions of class and type are used in both cases, with slightly different meanings, as explained in the following (see also "A Semantic Web Primer for Object-Oriented Software Developers", W3C Working Group Note, March 2006 [9]).

Speaking in the broadest sense, typing of entities and relationships, as used by NGSI-LD, assigns them to classes which broadly categorize them as real-world things or relationships, according to supposedly permanent criteria that differentiate these things from other kinds/species/varieties/makes of similar or related things or relationships.

RECOMMENDATION-01. *NGSI-LD users need to attach types to all the entities, relationships and properties that they intend to instantiate in their implementations.*

RECOMMENDATION-02. *These types should be directly or indirectly derived from classes defined in shared OWL ontologies, thesauri, or taxonomies[7].*

Such types, deriving from ontological classes (see "Ontology development 101: A Guide to Creating Your First Ontology". Knowledge Systems Laboratory 32 (1. January 2001), Stanford University [10]), are simply one means to attach permanent pieces of "meta-information" to NGSI-LD entity or relationship instances, i.e. to "classify them", in a way which differs from what is done by the use of attributes (properties). In the NGSI-LD API specification (see ETSI GS CIM 009 [3]), query functions are defined that can very efficiently filter on the basis of a few specially-defined NGSI-LD types of entities/relationships, whereas operations referencing the nearly infinite number of possible user-defined attributes (properties) are more constrained (or much slower).

A trade-off has to be made, however, as to how many levels of inheritance are suitable between user-defined types and generic, widely shared ontological classes which they ultimately inherit. The NGSI-LD API (v1.3.1 ETSI GS CIM 009 [3]) uses 17 of the cross-domain classes defined in the NGSI-LD cross-domain ontology (see ETSI GS CIM 006 V1.1.1 [1]).

[7] OWL classes correspond to SKOS concepts and RDF types, all of them matching the more foundational definition of "concepts" in description logic, where the axioms and relationships that bind these concepts together are captured what is called a "Tbox" (*Terminological component*), while the association of entity instances/individuals to these concepts/classes/types is captured as part of the "Abox" (*Assertion component*).

RECOMMENDATION-03. *These types must reference directly or indirectly the NGSI-LD meta-classes (entity, relationship, property) and should reference directly or indirectly, both the NGSI-LD cross-domain ontology and a domain-specific ontology, thesaurus or taxonomy which has achieved consensus within a given vertical domain or technical area of competence such as e.g. buildings, civil engineering or agriculture.*

The next sections provide some guidance on how to define (or choose from existing sets) the use-case specific or domain-specific types/classes.

2.2 Using typing vs. using relationships or properties

Specific types, defined as subclasses of more generic classes, usually have additional properties or relationships that the superclass does not have, or have restrictions different from those of the superclass. There is, however, no single universal criterion to choose between those characteristics of entities that are best expressed by typing, and those that are best expressed by assigning properties.

Typing allows potential checking of data consistency (though full consistency cannot be enforced if external classes are used). 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 referenced from the definition of the corresponding super-classes.

RECOMMENDATION-04. *Any characteristic feature that is intrinsic to a category of entities, does not vary from one entity instance to another within this category, and may be used to differentiate this category from others, should be assigned to individual instances through typing.*

RECOMMENDATION-05. *All extrinsic and instance-dependent features should be defined as properties.*

For example, the characteristic features of a room defined as a kitchen should set by its type inasmuch as they distinguish it from, say, a bathroom, in a generic way. Its area, and, even more obviously, its state (whether it is empty or occupied) should be defined as a per-instance properties, and whether it is adjacent to the living room should obviously be defined by a per-instance relationship.

RECOMMENDATION-06. *Characteristics defined by continuous-valued numbers, or with many possible values such as colours, should be defined through properties and do not normally justify the creation of new classes, except when such a distinction is fundamental to the domain[8].*

Further modelling choices may be less obvious, for example whether it is useful to define subclasses of the generic kitchen class to e.g. distinguish between open-space vs. traditional kitchens.

RECOMMENDATION-07. *It is usually better to use properties than to define sub-classes that might be too specific, too dependent on local cultures, or temporary trends.*

In general, using a property with predefined values to capture this kind of subcategorization is not a good idea either (see the clause “Guidelines for Use of Properties” in the following). Yet if a distinction is key to our domain, sub-classing it may also be warranted. If a distinction is important in the domain and we

[8] A classic example of this is in the oenology domain, where the distinction between white wine and red wine is fundamental to the whole domain and warrants the definition of corresponding subclasses in a wine-domain ontology

think of the objects with different values for the distinction as different kinds of objects, then we should create a new class for the distinction. A stool would, in this view, not just be a chair with three legs, but a different category of seating furniture altogether. If distinctions within a class form a natural hierarchy, then we should also represent them as sub-classes. If a distinction within a class may be used in another domain (as a restriction, or with multiple typing), then it is also better to define it as a subclass than by using a property.

To summarize: instead of associating similar properties to different entities that belong to a common category, a class can be defined to associate all those attributes[9] implicitly to all the instances that belong to it. In case of modification of a property, there would be one local placeholder to change, the attribute associated with the class, instead of changing the attribute in all the instances explicitly.

2.3 Using NGSI-LD-specific types vs. classes borrowed from ontologies or taxonomies

As per the received best practices of ontological modelling, an NGSI-LD information model should in principle make use, as much as possible, of existing classes drawn from standardised, curated and widely shared OWL ontologies, SKOS thesauri, or RDFS taxonomies (such as schema.org). These classes, be they generic or domain-specific, may be customised to the needs of NGSI-LD modelling by defining subclasses using transitive subclassing, as explained in the following.

Such an NGSI-LD-defined type is always, by default, a subclass of the topmost types of the NGSI-LD meta-model (NGSI-LD entity, relationship, property). It may and should, as much as possible, be a subclass of one of the cross-domain classes from the NGSI-LD cross-domain ontology, such as the mobile/movable/stationary distinction, which are defined by NGSI-LD in a generic rather than domain-specific way.

2.4 Multi-typing and transitive sub-classing issues

The current version of the NGSI-LD API (ETSI GS CIM 009 [3]) does not support multi-typing (assigning multiple types to a single entity). The NGSI-LD API is by design agnostic to the choice of a domain-specific ontology, so that it remains flexible. An analysis of usage of Multi-Typing in a potential future version of the NGSI-LD API is provided in the Annex and an example is provided here as motivation.

A 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 a vehicle at the same time (Figure 3). It would thus replicate definitions of relevant attributes from classes "Vehicle" and "Home". If characterized with dual typing instead (Figure 5 in Annex), the definitions would be directly inherited from the two classes rather than having to redefine both again inside a new type. Multi-typing also avoids having to define other narrowly specific classes for slightly different categories such as e.g. a “towed trailer” (a.k.a. caravan) which is also a kind of home, is not an autonomous vehicle but is nonetheless mobile.

Transitive sub-classing would make it possible, supposing we define this “Motorhome” type of vehicle as a subclass of "Mobile", that every instance of this type inherits typical attributes of a mobile entity,

[9] these attributes are, however, just “slots”, i.e. placeholders for property-values or relationship-targets that are defined only at the instance level ; there is nothing, in ontological modeling, like static class variables defined in some OO languages like Java, which are the same for all instances of the class

namely an instantaneous position, and possibly a speed, without having to redefine them. Transitive sub-classing is also used by default when a specific entity type defined within NGSI-LD inherits the generic definition of an NGSI-LD Entity from the NGSI-LD meta-model.

2.5 Recommendations for NGSI-LD typing

This section assumes the use of single-typing, as available in NGSI-LD v1.3.1 (see ETSI GS CIM 009 [3]). See Annex for a discussion of the advantages of the use of multi-typing, not currently supported in the API).

The basic principle in single-type modelling is that a complete assortment of NGSI-LD-specific types has to be defined, as many as may be needed to match the different categories of entities being modelled.

All types thus defined for use in NGSI-LD automatically belong to at least one of the root types of the NGSI-LD metamodel (relationships, entities, properties). They should furthermore also be assigned, if available, the type (class) from the relevant classes in the NGSI-LD cross-domain ontology, rather than use similar concepts as they might be defined in a more domain-specific way by existing external ontologies. Finally, they may inherit any multiplicity of domain-specific classes that are relevant to their definition. This is shown in the examples from Figure 3 with types prefixed as “MyNGSI-LD- Types”. This use of typing will be more natural for developers who are familiar with typing in classical programming languages, especially object-oriented languages, because types in these languages are closer to templates that do fully define the specificities of any given class.

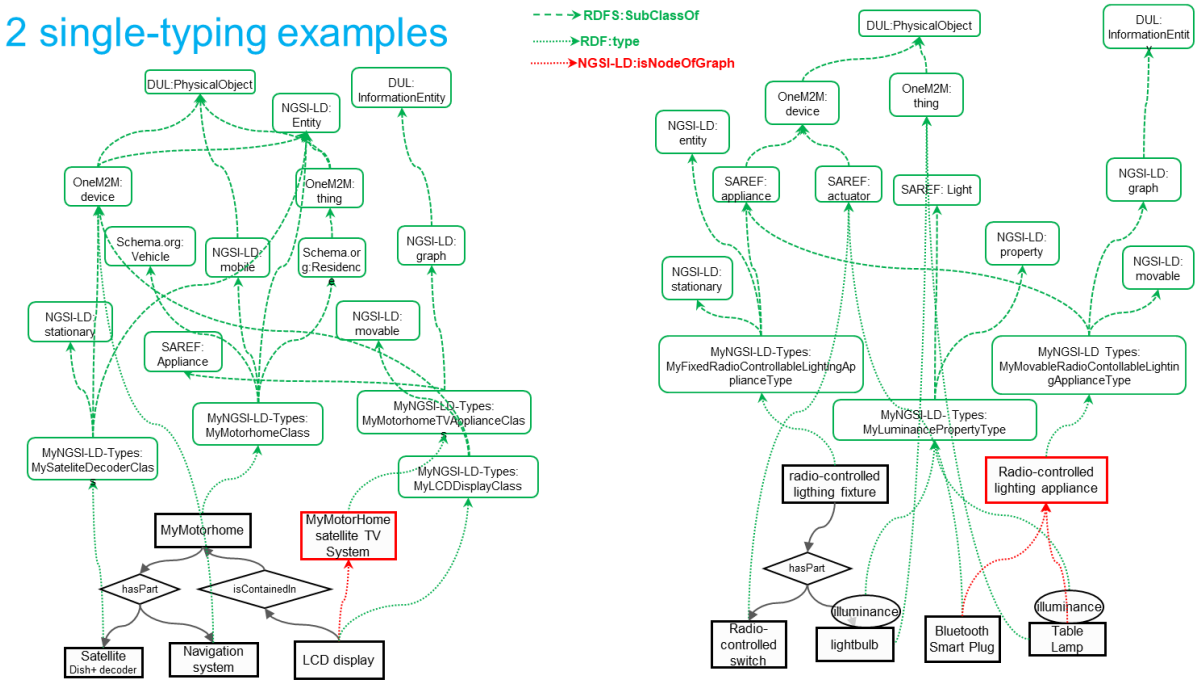


Figure 3: Examples of definitions of specific classes with single-typing and transitive subclassing

2.6 NGSI-LD types compared to types in object-oriented models and RDFS/OWL classes

This section is intended for readers who are more familiar with Object-Oriented modelling or RDF/RDFS/OWL modelling and would like to see a comparison.

NGSI-LD derives its formal foundation from the use of RDF types and RDFS/OWL classes, 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 document [9] and from an IEEE article "Think like a graph" [11]).

Property graphs, from which NGSI-LD is also derived, are actually closer in several regards to object-oriented modelling use than “pure” RDF. Therefore NGSI-LD typing is compared with both in the following table provided as an approximate overview.

Table 1: Comparison of different uses of typing in different modelling styles[10]

RDF/RDFS/OWL types/classes	NGSI-LD typing	Object-Oriented typing
Classes are just sets of individuals that are supposed to have something in common[11]	NGSI-LD types are *partial templates* for instances of entities or relationships	Classes are regarded as generalization of data types, assigned explicitly as *full-fledged templates* for object instances.
Each individual can belong to multiple classes : this is considered to be the foundational tenet of ontological modelling	The current version of the NGSI-LD API does not support multi-typing, but the NGSI-LD information model does not forbid it in principle, and implementations which do not use the API may support it[12].	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 to do still place restrictions on it (e.g. the classes vs. interfaces distinction in Java).
Class membership is not static and may be changed (by inference or by explicit typing) at runtime	The current version of the API does not support dynamic typing	The list of classes should be defined initially and should be fully known at compile-time; instances cannot change their type dynamically.
Reasoners may be used for (re-)classification and consistency checking at any time	Reasoners should be used with caution because the high expressivity of JSON-LD may lead to undecidability (which is also the case with OWL-Full)	Compilers are used to check consistency of instances with classes, but only at build-time

[10] This table intends to provide only a *practical* comparison, not to lay out formal definitions of fundamental concepts from OWL or OO modeling

[11] this “something” is precisely what is captured by the intentional definition of the class. Just as for a concept, a class definition captures what distinguishes a category from a broader, narrower, or related concept. Extensional definitions (listing all members of a class), if they are at all possible, are not very useful, except for trivial cases, and do not in general suffice for the proper definition of a class. RDFS/OWL Classes are closer to type theory than to set theory in that they do not satisfy the extensionality axiom of set theory : two classes could have the same extension but still be different classes through their intentional definitions.

[12] see Annex for specific recommendations with multiple-typing

An incomplete model can still be used.	An incomplete model can still be used.	Consistency is checked at compile time; an inconsistent program cannot be compiled and thus cannot be run
Properties in RDF are not statically attached to classes and may be added to individuals independently of their class definition.	NGSI-LD properties and relationships are not statically attached to classes and may be added to entities/relationships independently of their type definition.	Attributes are defined locally to a class (and to all its subclasses through inheritance).
Instances can have arbitrary values for any property. Range and domain constraints are used for type checking and type inference.	Instances can have arbitrary values for any property. Range and domain constraints could be used for type checking and type inference.	Range constraints are used for type consistency checking. Values are correctly typed according to the property constraints in the class.
Classes defined in a public ontology file can be referred to from any other class or instance definition anywhere. They are supposed to be fully defined and disambiguated by using a namespace prefix Consistency of their use may be checked by running a reasoner	Classes defined in a public ontology file can be referred to through transitive subclassing, but it should not be assumed that they will be known from a platform implementing the NGSI-LD API. Consistency cannot be enforced if classes are not drawn from a curated set of preferred ontologies All types have a unique URI	Classes are local to a given program. Consistency is checked and enforced a compile time
No Unique Name Assumption: *an instance may have multiple names assigned from its different sources, which may be reconciled a posteriori (using an owl:sameAs object property between the 2 instances)*	**No Unique Name Assumption:** *an instance may have multiple names assigned from its different sources, which may be reconciled a posteriori (using an owl:sameAs object property between the 2 instances)*	**Unique Name Assumption (UNA) :** *an instance has only one name (within the scope addressed)*
Open World Assumption (OWA)*: If there is not enough information to prove a statement true, then it may be true or false***.**	**Open World Assumption (OWA)***: If there is not enough information to prove a statement true, then it may be true or false***.**	**Closed World Assumption (CWA):** *If there is not enough information to prove a statement true, then it is assumed to be false***.**

3 Guidelines for use of relationships

Relationships between two NGSI-LD graph nodes (NGSI-LD entities as defined above) should be created to:

- stand for actual physical relationships between the physical entities that the nodes stand for. These relationships may correspond to either:
    - an actual permanent physical connection through e.g. a tube, pipe, cable, road, street, gate, doorway, etc.
    - a connection supporting the transport of some kind of physical stuff, like e.g. a link between stakeholders in a logistics/supply chain network, a link between containers in a waste collection network

- a wireless connection through an electromagnetic link of some kind, be it radio, IR or optical
- a physical containment or adjacency relationship, like a room being inside a building, or near adjacent to a corridor
- a purely logical or informational connection

- denote an actual social or legal relationship between the entities that the nodes stand for, like a connection between friends in a social network, or contractual relationships between companies
- denote inclusion of a node or graph in a subgraph when the target of the relationship is a node of the *graph* class (as defined in the NGSI-LD cross-domain ontology)

Relationships with origin in a relationship or property are allowed[13]. This kind of construct may e.g. be used when information such as provenance or ownership needs to be captured : the target of the corresponding relationship is the legal entity or person who vouches for, owns, or created the relationship or property in question. (see Annex 2 for examples).

The alternatives for use of relationship vs entities in modelling a physical network are laid out in Figures 4 and 5 below. Figure 4 gives three alternatives for the modelling of a road network. Using vertices to capture both ways (streets, roads, etc. whatever they may be) and crossings, with purely logical relationships to connect them.

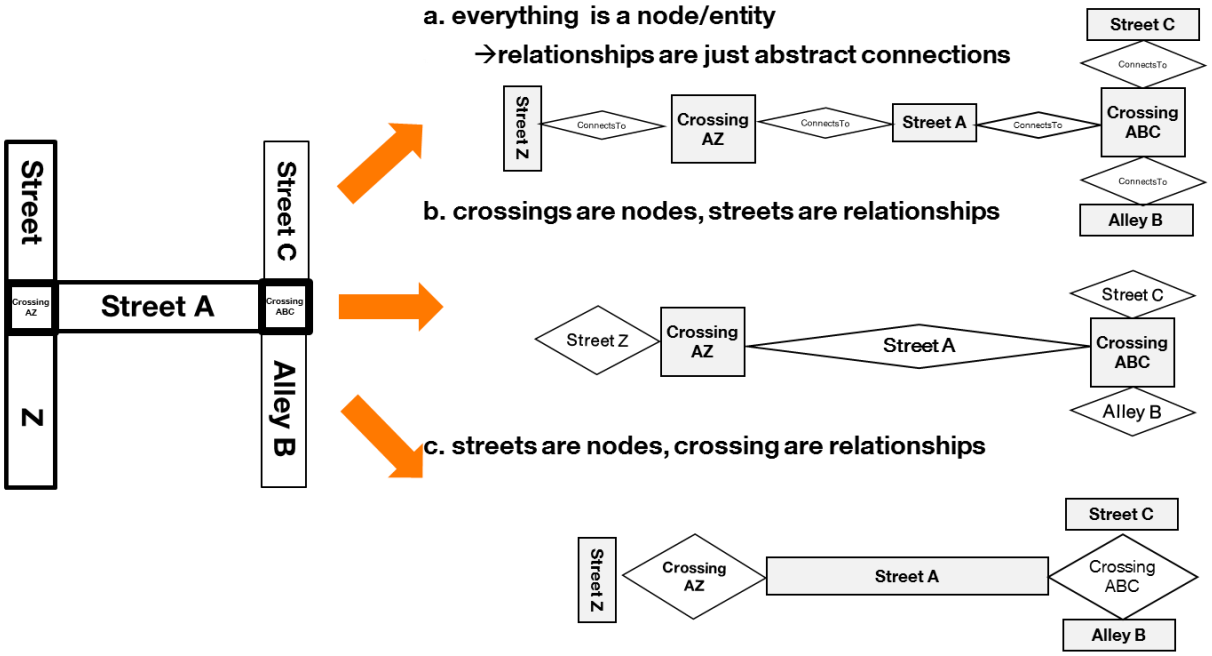


Figure 4: Alternatives for modelling a road network with NGSI-LD

[13] They are actually required for compatibility with the NGSI-LD API, but they may be un-supported by some NGSI-LD implementations, because they are not normally used in graph databases, because the customary Property Graph model does not vouch for them. They should be used with caution, when it is not possible or suitable to create an additional node as their point of origin

Figure 4a appears as a rather heavyweight and cumbersome solution (see "Think like a graph" [11]) but is in fact the only solution that is compatible with the API, because the API does not support queries through relationships. Capturing ways as edges (Figure 4b) and crossings as vertices would otherwise be the preferred solution (see “Using a conceptual model to transform road networks from OpenStreetMap to a graph database” [12]), because it matches the normal use of a graph model in classical graph modelling. Obviously, it does also align with Occam’s razor principle. If solution a. is used for compatibility with the API, a more graph-oriented representation like b. could still be extracted by transformation to run classical graph algorithms for e.g. shortest path or maximal flow computations.

A somewhat counter-intuitive solution (Figure 4c) captures ways as vertices and crossings as relationships. This does create as many entities as solution 4a , but creates other problems.

For another example in a water network, we present both the “everything is a node” solution (Figure 5.a), where relationships are just abstract *un-instantiated* logical connections à la RDF and the more classical-graph-like solution (Figure 5.b), where relationships clearly stand for actual physical pipes, i.e. physical connections.

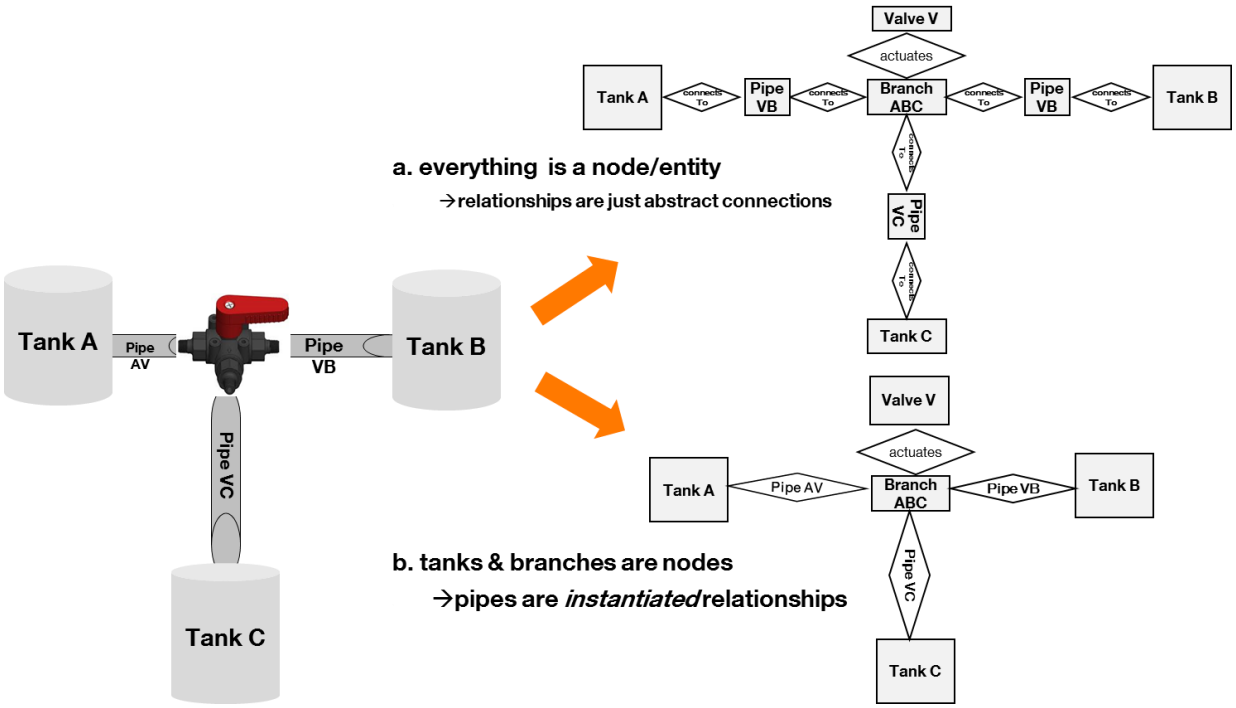


Figure 5: Example of relationship used to model a physical connection in a water network

4 Guidelines for use of properties

NGSI-LD properties may apply to NGSI-LD entities, relationships, or other properties[14]. The use of NGSI-LD properties is very similar to that of an attribute in classical key-value or object-oriented models. The target of a property can only be an NGSI-LD value, which is slightly more general than an RDF literal

[14] Properties of relationships are an integral part of the customary PG graph model, but properties of properties are not. They may not unsupported by implementations of NGSI-LD that rely on graph databases.

because it may be a complex data structure. It should not however, be identified by an URI, in which case an entity should be created and a relationship should be used instead[15].

Properties are often used to create certain subcategories, with values from enumerated types as their target. This is better than defining sub-types that would be too narrow or culturally dependent. When these subcategories are, however, essential to the domain being defined (such as for the distinction between, say, red wine and white wine), they are better defined with subtypes than with enumerated properties.

5 Summary of Recommendations

RECOMMENDATION-01.	NGSI-LD users need to attach types to all the entities, relationships, and properties that they intend to instantiate in their actual implementations.
RECOMMENDATION-02.	These types should be directly or indirectly derived from classes defined in shared OWL ontologies, thesauri, or taxonomies .
RECOMMENDATION-03.	These types must reference directly or indirectly the NGSI-LD meta-classes (entity, relationship, property) and should reference directly or indirectly, both the NGSI-LD cross-domain ontology and a domain-specific ontology, thesaurus or taxonomy which has achieved consensus within a given vertical domain or technical area of competence such as e.g. buildings, civil engineering or agriculture.
RECOMMENDATION-04.	Any characteristic feature that is intrinsic to a category of entities, does not vary from one entity instance to another within this category, and may be used to differentiate this category from others, should be assigned to individual instances through typing.
RECOMMENDATION-05.	All extrinsic and instance-dependent features should be defined as properties.
RECOMMENDATION-06.	Characteristics defined by continuous-valued numbers, or with many possible values such as colours, should be defined through properties and do not normally justify the creation of new classes, except when such a distinction is fundamental to the domain.
RECOMMENDATION-07.	It is usually better to use properties than to define sub-classes that might be too specific, too dependent on local cultures, or temporary trends.

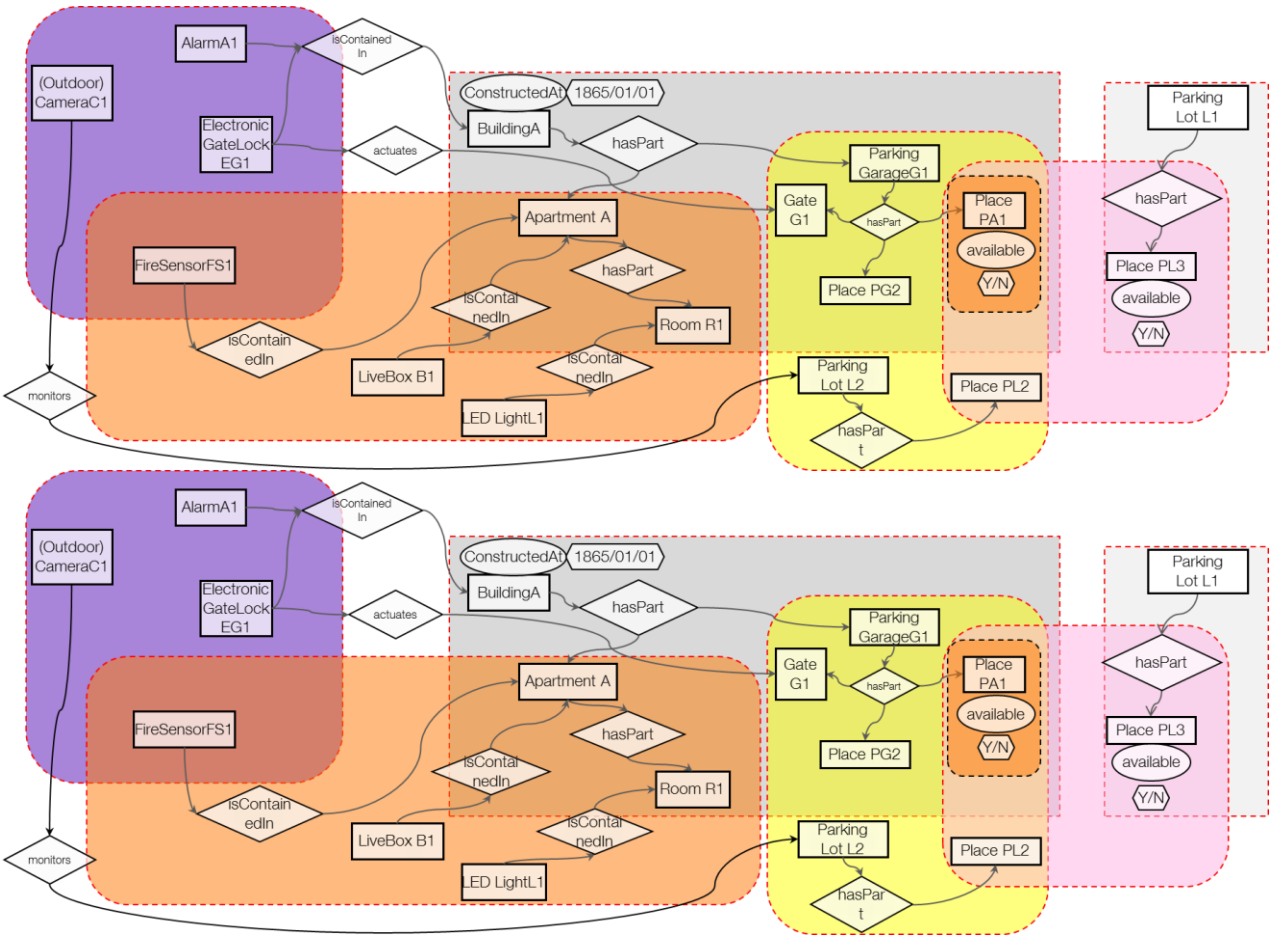
Further suggestions are given in the FAQ section, later in this document, however it is difficult to create universally applicable recommendations.

[15] URIs are legal values, only URIs referring to entities should not be used as Property values as Relationships are intended for this purpose

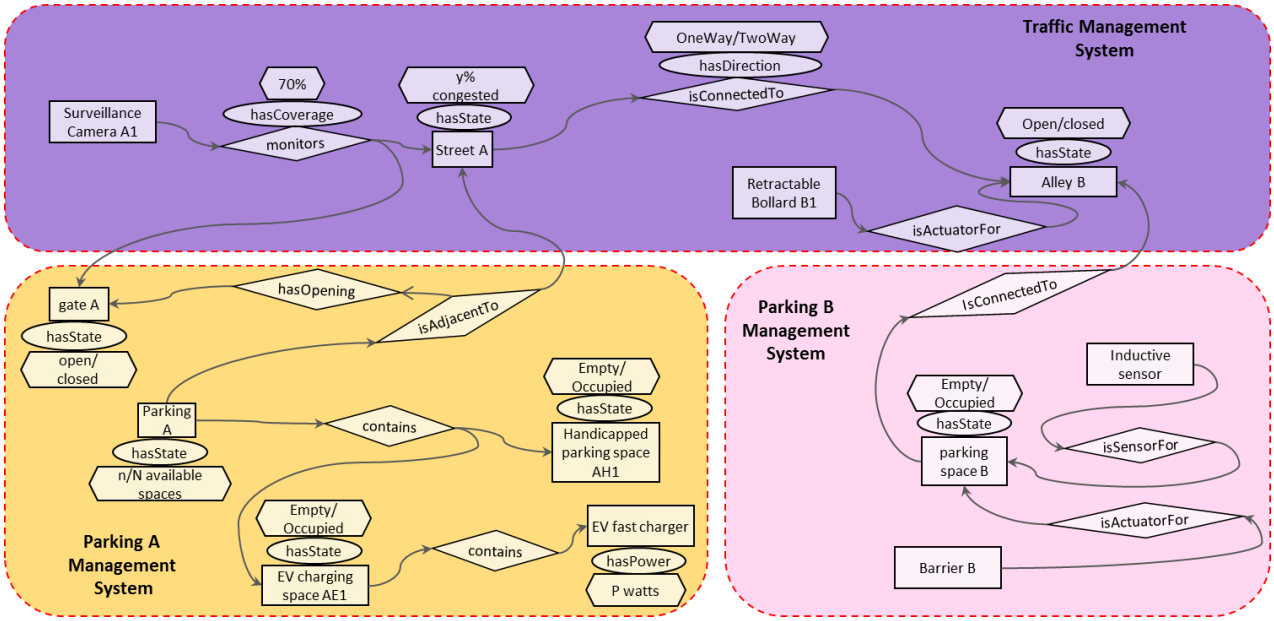
6 Example Graphs

Overall graph that brings together the simpler examples used in the previous sections of the document to illustrate various modelling choice alternatives

6.1 Smart Building example



6.2 Smart City example



6.3 SmartCity IoT data example

This use case example presents how IoT data can be modelled in the NGSI-LD context. This subsection is mainly based on material contributed by one of the authors (Franck Le Gall).

Let's start by a simple use case that consists on a sensor that provides the temperature of a room. In this case we have two main entities which are a room and a sensor but the main important question where we should put the temperature value provided by the sensor: In the Sensor entity or in the Room entity. Is there any recommendation?

Both possibilities are depicted in the figure below:

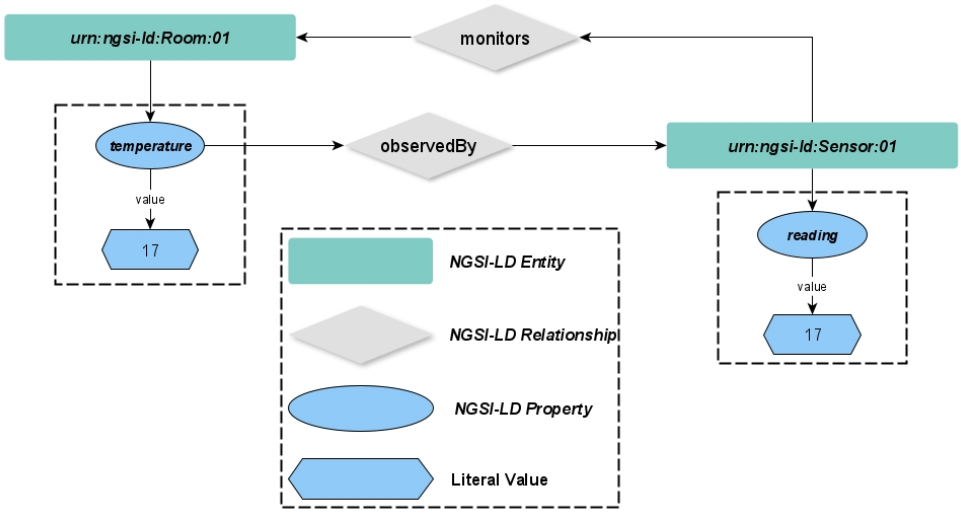


Figure 6: Room Temperature NGSI-LD data model

The answer to this question "where we should put the temperature value provided by the sensor" depends on how data will be queried.

- If we are querying the temperature of the room, it is better to have the value of the temperature in the room entity. So, one query can be executed to answer to this request. Otherwise, when the value of temperature is putted in the sensor entity. We have to check all the temperature sensors that monitor this room. Then we query the temperature value of the found sensor. Here we are running at least two queries to answer to this simple request.
- If we are querying all values of a given sensor if the temperature value is in the room entity so the query will be translated to check all values of the relationship “observedBy” of temperature properties. If temperature values are putted in the sensor entity, then the query it will be easier.

Generally, when working on contexts the main important idea is to put the temperature of the room in the room entity and then use the principle of relationship of a property to link it with its provider.

Below an NGSI-LD example of the Room entity with a temperature value when the temperature is put as a property in the room entity.

```
{
    "id": "urn:ngsi-ld:Room:01",
    "type": "Room",
    "temperature":
        {
                "type": "Property",
                "value": 17,
                "observedBy": {
                          "type": "Relationship,
                          "object":"urn:ngsi-ld:Sensor:01"
                          }
    },
    "@context": ["https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-
context.jsonld"]
}
```

In IoT systems generally a sensor system may provide several types of values. In this case, we recommend to model this use case as a system of sensors where an individual sensor is attached to a concentrator which connects together several types of sensors through a concentrator. This use case will be modelled as the figure below:

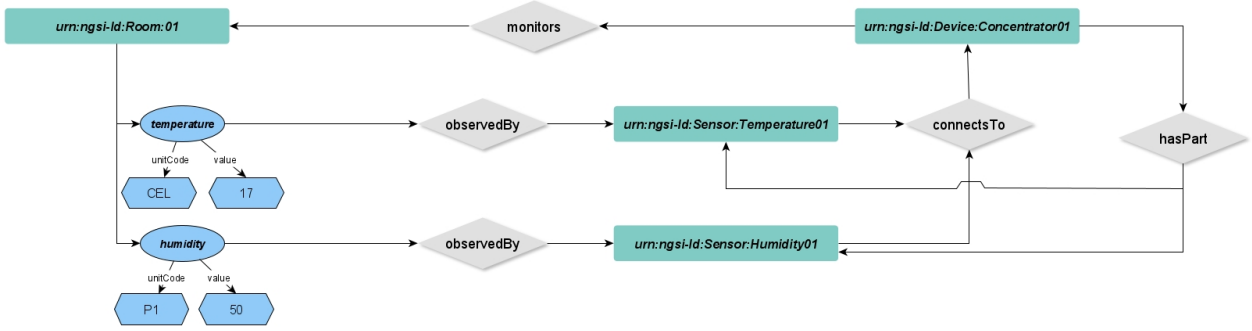


Figure 7: Modelling a Concentrator Device in NGSI-LD

The NGSI-LD relationship "connectsTo" is added between the concentrator and device Entities. Temperature and humidity properties are observed by the corresponding sensor. To introduce more the NGSI-LD modelling recommended steps, serval examples are presented below

7 NGSI-LD modelling FAQ

7.1 Meta-Model and semantics fundamentals

Since NGSI-LD graphs can be converted to RDF (through JSON-LD serialization) and are formally defined based on RDF/RDFS/OWL, why not just use an RDF graph throughout, instead of those pesky property graphs?

- *Property graphs have been a de facto standard since quite a long time, and their track record in the industry demonstrates their versatility and scalability*
- *RDF graphs are inherently less expressive than property graphs (PG). The difference is not one of degree, but of kind, PGs corresponding to second-order logic [16]while RDF is limited to first order logic. The conversion of property graphs to RDF requires the use of reification[17], which, to express statements about statements, entails an (up to fourfold) expansion in the number of statements and obfuscates the underlying structural semantics of the property graph*
- *RDF graphs are inherently less efficient than property graphs as an information storage mechanism, because they code entity attributes as properties, thus in the same way as actual graph arcs, even though they are dead-ends and meaningless arcs from a graph-modelling viewpoint*

How is it that the semantics of an NGSI-LD graph is not reducible to RDF-style semantics? I thought the whole idea of semantics is just derived from RDF and nothing ever existed before RDF

- *The kind of referential semantics expressed by RDF is the most basic to apply in information science. It applies locally (independently of context) on a per resource basis, similar to the semantics of the simplest sign systems, as long as they don't account for syntax, e.g. context-*

[16] second-order logic allows predicates to be the subject of other predicates
[17] see questions on reification in interoperability and conversion issues

free lexical semantics. Many more foundational theories of formal semantics pre-existed RDF, such as the semantics of natural languages or the operational semantics of programming languages. All these apply globally and do account for context through the proper use of sign composition syntax. The semantics of property graphs, as well as those of all graph models that have been used centuries before RDF existed, apply globally to the structure of the graph as a whole, in that it matches the structure of a physical network, or more generally of physical or logical or social relationships

Are NGSI-LD entities limited to represent physical entities? Couldn't they be purely informational entities in their own right?

- *The referent entity that an NGSI-LD entity stands for is supposed to exist somehow "in the real world", independently of the NGSI-LD platform that maintains its representation. It need not be something strictly physical: it may be for example a legal entity, or any shared informational construct that is maintained and persisted in its own right outside of the NGSI-LD- platform such as a movie, an event, etc.*
- *NGSI-LD nodes from the "graph" class are different from regular nodes in that they identify clusters of other NGSI-LD nodes that make up a subgraph, and they stand for a system with one level of indirection, i.e. through the nodes they group and the relationships that connect them. These systems cover all the spectrum between the following:*
- *loose federations of things, coupled as systems by belonging to the same NGSI-LD subgraph: e. g. supply-chain networks, vehicle sharing systems, short-term rental system*[18]
- *physically distributed systems and physical networks : e.g. telecom networks at the infrastructure level, water or electricity distribution networks, surveillance systems, etc.*
- *federations of independent entities operated by the same stakeholder, at any scale, like e.g. a like a desktop computer and various peripherals, a logistical network, a waste collection network, or, in different vein, a sharing system grouping physical assets*
- *systems of systems in the sense of the customary definition a bottom-up assemblage of subsystems which are operationally independent and have not have been designed to work together, but which do happen to work together to provide a functionality that is more than the sum of those provided by the individual subsystems separately; examples are the Internet at large, or a city, be it smart or not!*

Isn't an NGSI-LD relationship the same thing as an OWL "ObjectProperty", and an NGSI-LD property the same thing as an OWL "DatatypeProperty"?

- *An OWL Object property is still an RDF property, and cannot, as such, be the subject of another property without being reified. An NGSI-LD relationship is, by contrast, a first-class citizen of the NGSI-LD meta-model, and can be the subject of properties, and even other relationships (depending on implementation)*

When modelling physical networks (like road networks, electrical grids, water distribution systems, etc.) with NGSI-LD, should I represent the arcs/edges (actual pieces of physical connection) of these networks (respectively sections of roads, cables, or pipes between crossings or branches in previous examples) as NGSI-LD entities, or NGSI-LD relationships?

[18] in this last case, the system is purely informational and does not have a physical counterpart as such, only the entities it federates have such a counterpart

- *If the graph model is to be used as a basis to run classical graph algorithms (like maximal flow shortest paths, etc.), or complex systems analysis algorithms from spectral graph theory, the structure of the graph should match that of the physical network it models, which means arcs of the graphs should be used to represent sections of the network between branches.*
  The Property Graph model makes it possible to attach properties to these arcs, which may correspond to the width of a road section, the bandwidth of a cable, the diameter or max flow of a pipe, etc.

7.2 Typing issues

I don't find a suitable type in existing ontologies for a new entity that I need to characterize: should I define a new ad hoc type of mine, customized to serve as a perfect-fit template for this entity?

- *Ontological types are meant to be shared. Defining customized or ad hoc types to be used in one single environment or application normally defeats their purpose, but a compromise can be made if multiple typing is not supported, as is the case in the current version of the API. If existing types defined in shared public ontologies are not specific enough to define our new-found category of entity, and if multiple typing is not used for compatibility with the current version of the NGSI-LD API, a specific type may be defined, as the intersection (via inheritance) of several "known" classes from shared ontologies. Supposing for instance that we wish to define a light bulb as being a "smart" radio-network-actionable light bulb, it would thus be defined by creating an "intersection type" inheriting both the traditional lighting appliance category (which defines it as providing a lighting service, just like, say, a candle), and the connected device, or networked actuator categories.*

May I "borrow" properties from unrelated types in unrelated ontologies to add them to a customized type?

- *It is perfectly allowed and even advisable to do so*

To specialize an existing class to match possible variants, is it better to use a property with enumerated value type as a range, or to use subtyping?

- *Subtypes should be defined only when the corresponding subcategory is fundamental to the domain being modelled. Thus it would make no sense whatsoever to define a subclass of red shirts or white shirts within the shirt class: a colour property will do. It makes sense, however, to define a subclass for red wine or white wine if the wine domain is being modelled, as this distinction is fundamental to the whole domain, and the corresponding number of subcategories is limited (which obviously would not be the case for the colour of shirts…)*

May I use multiple types to customize a type that does not fit so well my entity?

- *If multiple typing is supported, it is the preferred way to do so (see Annex 1) and may be done directly. If it is not supported, customized types have to be defined, which would themselves subtype multiple existing types from already-existing ontologies, together with the types from the NGSI-LD cross-domain ontology.*

May I use UML or OO types/classes in place of classes defined in ontologies?

- *Type definitions should be available from a public repository and conform with RDF/RDFS/OWL formalism. If full-fledged OWL ontologies are not available for my domain, an RDFS taxonomy (such as schema.org) or SKOS thesaurus may be used.*

7.3 Interoperability and conversion issues

Is it possible to convert any RDF/linked data dataset or any JSON-LD model instance to NGSI-LD?

- *Direct conversion is always possible by transforming RDF/OWL datatype properties into NGSI-LD properties and object properties into NGSI-LD relationships, using the same types as in RDF. Such a dataset will not draw, however, all the potential benefit from the higher expressivity of the property graph model.*

Is it possible to convert the other way around, from NGSI-LD to RDF/linked data?

- *Conversion from a high-expressivity information model to a low-expressivity one entails some inevitable loss of information or complexification /expansion*
- *Basically NGSI-LD properties are mapped to RDF/OWL datatype properties and NGSI-LD relationships are mapped to object properties, using the same types in RDF*
- *BUT: property predicates are not identified individually as instances in RDF, they are defined only by their type, which means that NGSI-LD identifiers of properties and relationships will get "lost in translation"*
- *Properties of relationships and properties of properties CANNOT directly be expressed in RDF : this is where reification comes in*

What does reification mean?

- *Reification makes an RDF statement (a subject →predicate → object triple) into the subject of another statement*

is the kind of reification used by NGSI-LD (for conversion to RDF) the same as RDF reification?

- *There are multiple ways to reify RDF statements. Standard RDF reification defines a statement as a resource of type* `rdf:Statement`*, with the attached properties* `rdf:subject, rdf:predicate, and rdf:object`*, so that a total of 4 additional statements (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!*
- *NGSI-LD reification is equivalent to RDF reification using so-called "blank nodes : if we want to reify the statement*

  [CameraA → monitors → StreetA]

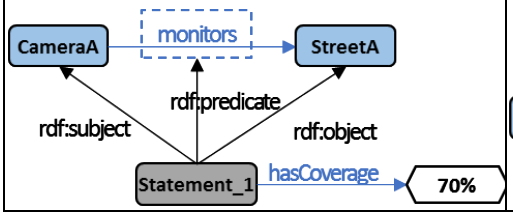
in order to make it the subject of another statement:

[[statement_1] → hasCoverage → 70 %],

we add a blank node to obtain an RDF-reified equivalent of the example property graph with three triples as shown in the diagram below. This blank node does not show up using JSON-LD serialization, because all NGSI-LD relationships are "pre-reified" using the special relationship "hasObject"

"heavyweight classical RDF reification	*lightweight blank-node based reification used by NGSI-LD*

<pre>{"@id": "CameraA",
"monitors": {"@id": "StreetA"}}
{"@id": "Statement_1",
"subject":
{"@id": "CameraA"},
"predicate":
{"@id": "monitors"},
"object":
{"@id": "StreetA"},
"hasCoverage": "70%" }</pre>
	<pre>{"@id": "CameraA",
	"monitors":
	{"hasObject":"@id": "StreetA},
	{"hasCoverage":"70%" }}</pre>

Is NGSLI-LD reification related to reification in object-oriented modelling

- *It is loosely related in the sense that OO reification makes explicit or accessible something that is hidden or implicit in the model*

Is NGSI-LD reification related to reification in Marxist philosophy ?

- *Only in the most basic etymological sense of "transforming into a thing" ("Verdinglichung"), or making a subject into an object, which as is a form of human alienation…*

Is it possible to directly convert the native formats of existing graph databases to NGSI-LD?

- *Most graph databases use the property graph as their implicit data model, but they may not implement all features of NGSI-LD as it defines this model: for example properties of properties or relationships of relationships*

Do object-attributes of an object-oriented model map to NGSI-LD properties?

- *Yes*

Do the relationships of an entity-relationship model map to NGSI-LD relationships?

- *Not necessarily*

Do the entities of an entity-relationship model map to NGSI-LD entities?

- *Entities of an ER model map to entire tables of a relational database, whereas entities of NGSI-LD would normally map to rows of such a database*

Do the relations of a relational database map to NGSI-LD relationship?

- *Not at all: relations of a relational database are n-ary, whereas relationships of a graph database are binary ; n-ary relationships may also be captured by an hypergraph, which is an entirely different species of mathematical construct altogether*

7.4 System-level modelling issues

When should I single out a set of entities (graph nodes) as a subgraph (using the "isNodeofGraph" relationship targeting a corresponding "graph" entity)?

- *Typically these nodes should make up a system, in a way that does not duplicate other relationships that are captured directly by the graph. For example, a traditional self-contained top-down hierarchical system will have "hasPart" relationships between all of its constituent subsystems, possibly at different levels. This kind of system is well represented by the uppermost node in the hierarchy, corresponding to the root of a rooted-directed tree (a.k.a. arborescence) : it need not be captured as an additional "graph" node, which would be redundant. For a distributed system like a physical network, or a set of entities that make up a system in any indirect way (e.g. managed by the same information system), another construct is necessary to single out a set of nodes that or may not be linked together as a subgraph: this is where the "isNodeofGraph" relationship comes into play.*

Why should I use this subgraph decomposition instead of just using a specific property to single out the same nodes? or a specific type?

- *A property would not make it possible to capture properly systems that are nested at multiple levels, which is essential in the description of complex systems. The proposed solutions support this using transitivity of relationships: a graph can be in turn a subgraph of a higher-level graph*

Do the relationships and properties attached to nodes that are part of a subgraph automatically become part of this subgraph?

- *Properties attached to nodes of a subgraph are obviously part of this subgraph; as for relationships, they are also if both their start and end nodes are part of the same subgraph; otherwise see the following*

Where should I draw the border of these subgraphs for relationships that cross from one subgraph to another?

- *There is no general rule: these relationships could belong to only one subgraph, to both, or none…*

7.5 Using the NGSI-LD Cross-domain ontology

7.5.1 Temporal classes

Do the createdAt/modifiedAt properties apply to physical entities, or to their representation in an NGSI-LD-compatible platform or database?

- *By contrast to "regular" NGSI-LD properties, these special properties apply to the NGSI-LD entity informational abstraction, (or database entry that supports it), NOT to the real-world entity that the NGSI-LD entity stands for. As a consequence of this, they cannot be reified*

7.5.2 Mobility classes

Is this fixed/movable/mobile distinction crisp and absolute or fuzzy and relative?

- *It is only a practical distinction, with fuzzy boundaries, definitely not a theoretical one (which would not make sense, from the viewpoint of, fundamental physics). Typically, a smartphone or car is mobile, a piece of furniture is movable and a wall is stationary. This relates to the location properties of the corresponding entities.*

Couldn't the same distinction be made with properties instead of typing (class inheritance)?

- *Making the distinction with a class makes it easier to inherit specific properties of either classes of entities (thus mobile entities may have a speed)*

7.5.3 Location classes

Why is it that location can be defined either by properties or relationships? Isn't it enough to define all entity locations as properties corresponding to GPS coordinates?

- *GPS coordinates are useless and meaningless inside a building, for example. The "Containedin" relationship provides a more practical reference for location by giving the room, (and by transitivity, the floor) into which a mobile or movable entity is contained*

Is it possible to use both at the same time?

- *Yes, just as it is usual to combine relative coordinates (within a local referential system) with absolute (e.g. geocentric) coordinates of the referential itself.*

Doesn't "setBasedLocation" overlap with system composition with this "isContainedIn" relationship?

- *The "isConatinedIn" relationship normally denotes set-based location, but may also be used to characterize a loose, purely bottom-up type of system composition*

Can the "ConnectsTo" relationship class be used as a superclass for any kind of link/edge in a physical network (e.g. sections of streets, roads, railways, cables, pipes, tubes, etc)

- *Yes : it is precisely intended to be used like this, and it can be used more generally to support "graph-based location" where such a physical network does not exist materially, for example inside a building (a room may thus connect to a corridor through a door, information which may be used to navigate inside a building.*

7.5.4 System-State properties

Why do so-called “state” properties have to be singled out as such? What is so special about those state properties that they should deserve this emphasis?

- *The notion of state is at the root of the theory of dynamical systems: the state of a dynamical system summarizes the entire history of the system, so that the future states and outputs of the system depend only on the present state and future inputs*
- *A state property maintained by an NGSI-LD entity has to mirror a component/dimension of the actual physical state of the physical system that the entity stands for, with strictly bounded latency. The need to satisfy this strong constraint, so that the state property may be used to perform an action on the system, is the reason why state properties have to be singled out*

How do I tell the difference between a “state” and a “non-state” property of an entity?

- *A non-state property can be e.g. a sensor measurement or reading, whose current value does not directly depend on previous values; a possible dependence may come from the state of an external system/environment that is not addressed in modelling, like e.g. the lighting of a room from daylight). By contrast, a state property obeys a state equation that defines its dependence on previous state values of the target system being modelled : for example, the occupancy of a room at time t depends on its occupancy at time t-dt and the number of people who have left or entered during the interval dt. Obviously, a non-variable property of a system like the size of the room is not a state value either.*

How is “continuousValue” related to “ContinuousTime”, and “discreteTime” to “discreteValue” for those state properties?

- *A continuous-value state dimension of a system may have either continuous-time or discrete-time dynamics. A discrete-value state dimension can have only a discrete-time dynamic. Most of the system dimensions s addressed in NGSI-LD modelling will be discrete-time, or at least modelled as such.*
- *A few textbook examples are for*
- *continuous-time & continuous-value system dimension: water level in a water tank*
- *discrete-time & continuous-value system dimension: filling percentage of a large warehouse by parcels offloaded from trucks and containers*
- *discrete-time & discrete-value system dimension: number of people inside a room*

7.5.5 System-composition classes

What kind of subgraph can be the source of the “isSubGraphOf” relationship?

- *These graphs have to be defined as being the target of “isNodeOfGraph” relationships, as subgraphs standing for systems, which are described to be subsystems of a larger system in this way*

When should I use the “hasPart” relationship rather than “isNodeofGraph”?

- *A self-contained system designed through traditional top-down engineering from tightly coupled subsystems like a building, a car, a plane, a machine-tool, etc. should be described using “hasPart” relationships if these subsystems/parts meet both of the following criteria :*

- *the overall system would not work properly or would fail completely if one of these parts were removed,*
- *the parts are specially designed to fit into the overall system (like a wall in a building, the foot of a chair, the keyboard of a laptop computer).*
- *A system whose parts are assembled as a looser federation, like a computer made up of separate central unit, display, keyboard and mouse, or a physically distributed system, should be described using the "isNodeofGraph" relationship (it normally meets the first criterion above, but not the second)*

When should I use the 'isContainedIn" relationship rather than "isNodeofGraph"?

- *"isContainedIn" is a relationship that captures both set-based location relationship and bottom-up system-composition relationship, in the NGSI-LD cross-domain ontology. It may be used to characterize a loose assembly of physical entities as being contained in the same container, like e.g. all the pieces of furniture inside a room. These entities do not necessarily make up a system as such, in the proper sense of the term. If some of these entities do provide a common functionality or service that is more than the sum of the services they provide individually, like e.g. all the pieces of equipment+furniture making up a personal workstation in an open-plan office space , they may be singled out and characterized further with an "isNodeofGraph" relationship.*

8 References

1. ETSI GS CIM 006 V1.1.1 (2019-07), “Context Information Management (CIM) Information Model (MOD0)”: https://www.etsi.org/deliver/etsi_gs/CIM/001_099/006/01.01.01_60/gs_CIM006v010101p.pdf
2. ETSI GR CIM 008, "Context Information Management (CIM) ; NGSI-LD Primer". https://www.etsi.org/deliver/etsi_gr/CIM/001_099/008/01.01.01_60/gr_CIM008v010101p.pdf
3. ETSI GS CIM 009, "Context Information Management (CIM); NGSI-LD API". https://www.etsi.org/deliver/etsi_gs/CIM/001_099/009/01.04.01_60/gs_cim009v010401p.pdf
4. ETSI GR CIM 002, "Context Information Management (CIM); Use Cases (UC)": https://www.etsi.org/deliver/etsi_gr/CIM/001_099/002/01.01.01_60/gr_CIM002v010101p.pdf
5. Margitus, M., Tauer, G., & Sudit, M. "RDF versus attributed graphs: The war for the best graph representation". In *Information Fusion (Fusion), 2015 18th IEEE Internat. Conf.* (pp. 200-206)
6. JSON-LD 1.1, “A JSON-based Serialization for Linked Data” : https://www.w3.org/TR/2020/CR-json-ld11-20200316/
7. SAREF, Smart Applications REFerence ontology, see
   (a) ETSI SAREF Portal https://saref.etsi.org;
   (b) ETSI TS 103 264 V3.1.1 (2020-02), SmartM2M; Reference Ontology and oneM2M Mapping: https://www.etsi.org/deliver/etsi_ts/103200_103299/103264/03.01.01_60/ts_103264v030101p.pdf ;
   (c) ETSI TS 103 267 V2.1.1 (2020-02), SmartM2M; Smart Applications; Communication Framework: https://www.etsi.org/deliver/etsi_ts/103200_103299/103267/02.01.01_60/ts_103267v020101p.pdf
8. Berners-Lee, T., Hendler, J., & Lassila, O. (2001). “The semantic web”. *Scientific American*, *284*(5)
9. Knublauch, Holger, Daniel Oberle, Phil Tetlow and Evan Wallace, "A Semantic Web Primer for Object-Oriented Software Developers", W3C Working Group Note, March 2006. https://www.w3.org/TR/sw-oosd-primer
10. Noy, Natalya and Deborah L. McGuinness, "Ontology development 101: A Guide to Creating Your First Ontology“. Knowledge Systems Laboratory 32 (1. January 2001), Stanford University. https://www.researchgate.net/publication/243772462_Ontology_Development_101_A_Guide_to_Creating_Your_First_Ontology
11. Liu, Z., Zhou, P., Li, Z., & Li, M. (2018). "Think like a graph: Real-time traffic estimation at city-scale". *IEEE Transactions on Mobile Computing*, *18*(10), 2446-2459.
12. Steinmetz, D., Dyballa, D., Ma, H., & Hartmann, S. (2018, October). “Using a conceptual model to transform road networks from OpenStreetMap to a graph database”. In *International Conference on Conceptual Modeling* (pp. 301-315). Springer

Annex A: Use of features and capabilities not supported by the API e.g. Multi-Typing

A.1 Introduction

As it was discussed in previous sections, NGSI-LD is trying to balance the benefits of object-oriented and ontological approaches to modelling. The NGSI-LD is flexible and currently does not impose many constraints on the definitions or usage of user-provided types or instances: there is no attempt to impose real-world domain knowledge on the Property Graph in the system.

However, very many real-world modelling examples rely on consensus-driven choices of constraints so as to improve interoperability. Examples abound in areas of Building Information Models, Electronic Health Records for patients, etc. This annex considers some of the features that need to be considered to apply such domain-knowledge constraints to modelling.

One of the important and challenging issues, which requires substantial analysis, is the question of multiple typing i.e., shall we allow the creation of instances which have simultaneously several assigned types. At the current time, the NGSI-LD API does not support assigning multiple types to an entity.

A motivating example that illustrates the utility of having an instance which possesses the properties of two different types (classes) is provided below. To create an instance of an entity which people would identify as a “MotorHome”, we need to borrow properties from both the types *Home* and *Vehicle*. The available choice is between assigning multiple types directly to the instance or creating a new type with a distinguishable name (e.g. MotorHome), which is assigned properties from two parent types (e.g. *Home* and *Vehicle*), and then create an instance of this newly defined type (single type).

Below, we provide a comparison of the aforementioned approaches.

A.1.1 The comparison of multiple inheritance and multiple typing

Allowing the assignment of multiple types to an instance helps to remove some of the modelling complexities. The main benefit is the elimination of a need for the creation of a new type, or the extending of an existing type, for each use case. However, multiple typing also creates potential issues, which need to be addressed. In this section, we analyse and compare the benefits and disadvantages of multiple typing approach compared with the use of single typing enhanced by multiple inheritance.

NGSI-LD is flexible regarding the definition of new types and any modeller who has a ‘write’ permission can create instances which have no meaning or are self-contradictory. For example, if an instance is allowed to be assigned the types *home*, *vehicle*, and *dog,* then while home and vehicle can be combined to make a “MobileHome”, a *dog* type is incompatible in this list. However, there is no way to check such maladroit assignments and prevent such entities from joining the common search space.

Unfortunately, in cases where an instance is created by referencing an existing type and just adding attributes from other types (classes), it becomes impossible to search for that subset of entities by means of a narrow type. For example, if an entity *MotorHome1* is only a mixture of various properties found for types *home* and type *vehicle,* and is assigned to both types *home* and *vehicle*, how can a search specifically find all the *MotorHomes*?

Such examples are ‘flat’; the hierarchy stops at such instances; they cannot be used as examples for further filtering. It is not possible to define a simple query to find the subset of all things that have the same kinds of properties as *MotorHome1.*

Such instances cannot be reused, whereas once a single type is created, it can be used as a template for other similar instances.

A.1.2 The utility of combining Multi-Typing with an Ontological Validation

If it is possible to impose an ontological structure, for example as a strict hierarchical taxonomy of types, or just a number of "belongsTo" relationships between types, then the uses of multi-typing expand.

Searching by a narrow type becomes straightforward. E.g. search for all instances of type ‘*MotorHome*’. Also, it is possible to search for all entities of type *Vehicle* and downwards (in the hierarchy), thus finding motorhomes when searching for cars. It is also possible to search upwards, thus finding cars and houses when searching for caravans. The hierarchy can be continued, e.g. a type *ElectricMotorHome* can be inherited from the *MotorHome* type. The *MotorHome* class can be reused to create more instances

If the "imposed ontological structure" also defines a schema of allowed properties and allowed values, then it also becomes possible to validate correctness when new instances are created.

In summary, the described single typing enhanced by multiple inheritance approach is based on the existence of a reference ontology, which is used to search downwards and upwards in the hierarchy. This ontology is used by the platform during the processes of creating new entities and in queries (i.e. API call).

A.1.3 Issue of extending ontological structures.

Real-life systems need the possibility of extensions. In the current API, there is no constraint on adding new types, however if there is an imposed ontology then there needs to be a means to add new types into the ontology.

In case, when a modeller creates a new type that inherits from existing types (e.g. *MotorHome* from *Vehicle* and *Home*), the type becomes available for creating instances. However, we need to prevent inappropriate use of this feature. If someone creates a type *Dog* and inherits it from types *Home* and *Vehicle* (potentially any of the existing types), it will have the result that everyone who searches for cars (including downwards in the hierarchy), will be getting instances of a type *Dog*.

This could be avoided by introducing double references (up and down) in the search hierarchy. A query, which is asking for instances of type *Dog* (and upwards) is getting all the instances of this type, plus instances of a *Home* and a *Vehicle*. It can be assumed that a person who queries for this type *Dog* is aware of and agrees to such a type. However, if the OWNERS/MAINTAINERS of the types *Home* and *Vehicle* (or the MAINTAINER of the whole ontology to which those types belong) did not approve the inheritance, then the system can be specified such that a search for a Vehicle instance will not return a *Dog* instance. In this example, nevertheless, modellers can form their own special types within shared ontologies, whereby such special types are not directly integrated into the ‘imposed’ ontology which is the assumed foundation of the merged dataset of all entities in the system.

A.2 Recommendations for NGSI-LD typing with multiple typing

The following recommendations apply if a form of multiple typing is supported in an NGSI-LD system (which means it does not use the AMPI) and they can be considered for future specifications of the NGSI-LD API.

Contrary to the case of single-typing, multiple typing alleviates the need to create specific "single-typing" classes that exactly match the requirements of the targeted domain and the entity instances to be modelled and may themselves inherit multiple classes from shared ontologies. Instead, with multiple typing, a similar result may be achieved by directly referencing these classes from the instances being categorized, thus picking and choosing known features from a variety of known ontologies, including the NGSI-LD cross-domain and metamodel ontologies. Choices could be made from many ontologies, thesauri, taxonomies, and vocabularies, be they generic or domain-specific, high-level, mid-level or low-level. Classes are not, in general, mutually exclusive, so multiple-typing avoids a granularity that amounts to define every single type by a small, mutually exclusive set of instances, cross-referencing multiple classes being a more versatile and adaptable way to describe their peculiarities than pigeonholing them into narrowly defined categories

With multiple-typing, new instances should be created by :

- referencing, directly or indirectly, at least one of the root classes of the NGSI-LD meta model (entity, relationship, property)
- referencing, directly or indirectly, generic classes from the NGSI-LD cross-domain ontology (for e.g. defining whether the instances being addressed are mobile, movable, or stationary). This is more generic than the use of more specific concepts as they might be defined in domain-specific ontologies.
- referencing multiple classes, chosen from generic or use-case-specific ontologies, to characterize specific features or peculiarities of these instances. With multiple typing, this is preferable to the use of properties or narrowly defined subcategories.

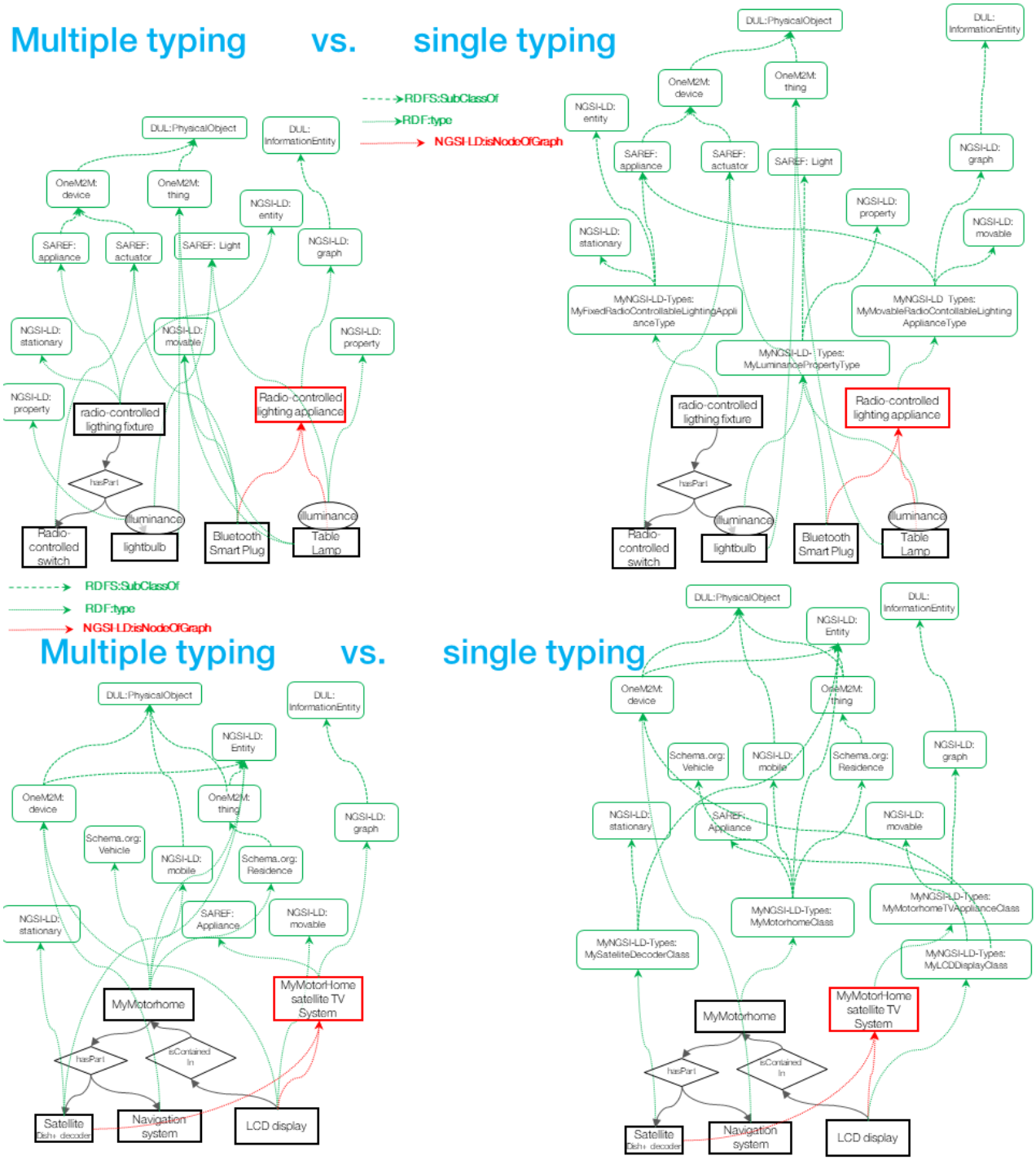


Figure A.1: Examples of different modelling choices depending on the use of multiple typing (on the left) vs. single typing (on the right)

Annex B: Examples for Smart Farming

B.1 Acquaculture

Following the same previous modelling principles (Values are added to the), the Smart Aquaculture use case consists of deploying an Aqua-box (device) in a fish containment in order to monitor various water parameters such as the dissolved oxygen, pH, and salinity. A Feeder is installed in this fish containment and it is executed by a service that we call feeder operation. Following the previous modelling steps this use case will be modelled using 4 main NGSI-LD entities:

- **FishContainment** presents the entity that contains all measures,
- **AquaBox** is the device deployed in the Fish containment to monitor it.
- **DissolvedOxygenSensor** which will provide the measure of dissolved Oxygen in the Fish Containment. It is connected to the Aquabox using the NGSI-LD Relationship “connectsTo”.
- **SalinitySensor** which will provide the measure of salinity in the Fish Containment. It is connected to the Aquabox using the NGSI-LD Relationship “connectsTo”.
- **PHSensor** which will provide the measure of pH in the Fish Containment. It is connected to the Aquabox using the NGSI-LD Relationship “connectsTo”.
- **FeedingOperation** in the Smart Aquaculture use case there are several other operations such as the Feeding Operation which will executes the feeder with inputs such as the food name and food quantity as NGSI-LD properties.
- **Feeder:** Is a machine attached to fish containment using the relationship “connectsTo” it is executed by the feeding operations

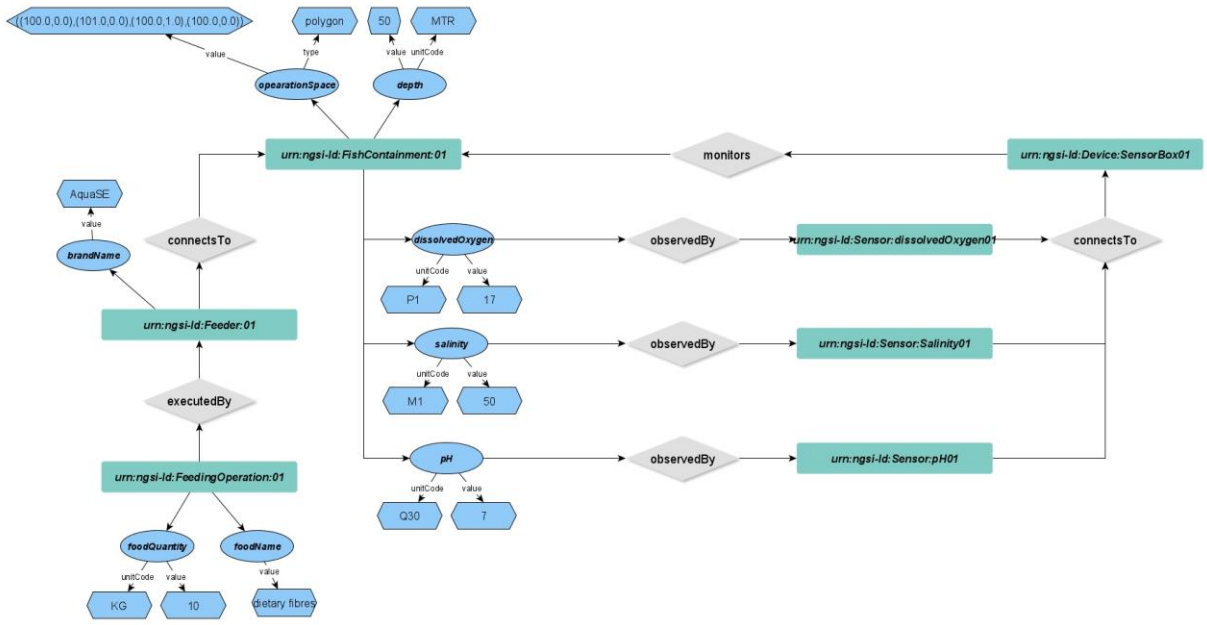


Figure B.1: Smart Aquaculture NGSI-LD data model

B.2 Smart agriculture

In the scope of agriculture, there are a lot of aspects that can be modelled, ranging from a possible Application for irrigation management, the field composition, through the irrigation devices to vegetation and water indices provided by satellite images. DEMETER project has made a great effort defining what they call the Agriculture Information Model, which considers different entities such as: AgriApp, AgriCrop, AgriFarm, AgriParcel, AgriCrop, AgriSoil, AgriParcelOperation, and AgriParcelRecord to name a few.

In this use case, we have selected a subset of these entities so that different properties of the soil could be described properly. The following diagram presents an example of such instantiation thanks to the NGSI-LD representation.

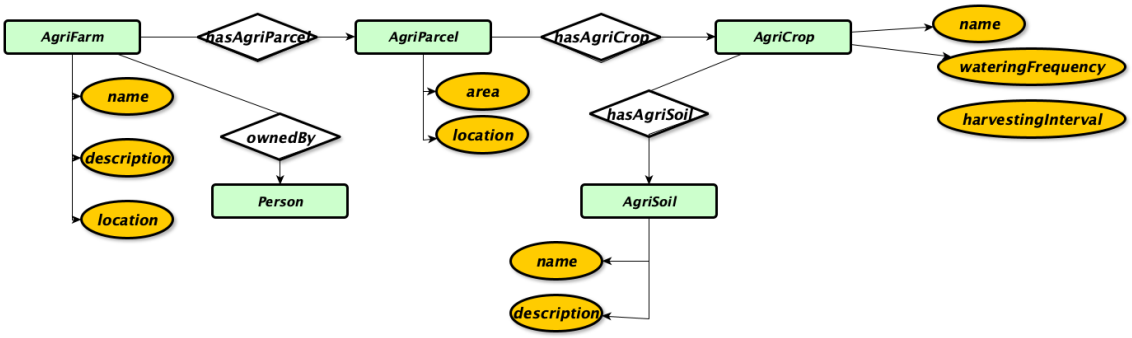


Figure B.2 : Smart Agriculture NGSI-LD model example

A farm represented by AgriFarm contains a harmonised description of a generic farm made up of buildings and parcels. The Data Model has the following properties:

- id: unique identifier (required)
- type: NGSI Entity type. It must be equal to “AgriFarm” (required)
- dateCreated: "2017-01-01T01:20:00Z",
- dateModified: "2017-05-04T12:30:00Z",
- name: Name of the farm,
- seeAlso: Other useful URL,
- description: General description of the farm,
- relatedSource: Reference application,
- location: Point of GPS coordinates of the farm,
- landLocation: Geometry defining the boundaries of the farmland,
- address: address of the farm,
- contactPoint: Contact information of the farm, i.e. email, telephone, etc.,
- ownedBy: Owner (Person or Organization) of the farm,
- hasAgriParcel: Relevant AgriParcel object

AgriParcel: This entity contains a harmonised description of a generic parcel of land. The Data Model has the following properties:

- id: unique identifier (required)
- type: NGSI Entity type. It must be equal to “AgriParcel” (required)
- dateCreated: "2017-01-01T01:20:00Z",
- dateModified: "2017-05-04T12:30:00Z",
- location: Point of GPS coordinates of the parcel (required)
- area: Area of the parcel (required)
- category: Category of the parcel (i.e. Arable),
- ownedBy: Owner (Person or Organization),
- seeAlso: Other useful URL,
- relatedSource: Reference application,
- belongsTo: Holder (Person or Organization),
- hasAgriParcelParent: Relevant AgriParcel object
- hasAgriParcelChildren: Relevant AgriParcel object
- hasAgriCrop: required, Relevant AgriCrop object
- cropStatus: Status of the crop ("seeded", "justBorn", "growing", "maturing", "readyForHarvesting"),
- lastPlantedAt: Date of the last planted crop,
- hasAgriSoil: Relevant AgriSoil object
- hasDevice: List of connected devices

AgriCrop: This entity contains a harmonised description of a generic crop. The Data Model has the following properties:

- id: unique identifier (required)
- type: NGSI Entity type. It must be equal to “AgriCrop” (required)
- dateCreated: "2017-01-01T01:20:00Z",
- dateModified: "2017-05-04T12:30:00Z",
- name: Name of the crop (required)
- alternateName: Alternative name (Scientific name) of the crop,
- agroVocConcept: URL of the FAO details of the crop,
- seeAlso: Other useful URL,
- description: General description of the crop,
- relatedSource: Reference application,
- hasAgriSoil: Relevant AgriSoil object
- hasAgriFertiliser: Relevant AgriFertilizer object(s)
- hasAgriPest: Relevant AgriPest object
- plantingFrom: Date Range of the planting,
- harvestingInterval: Date Range of the harvesting,
- wateringFrequency: Frequency of watering (choosing from: "daily", "weekly", "biweekly", "monthly", "onDemand", "other")

AgriSoil: This entity contains a harmonised description of a generic soil. The Data Model has the following properties:

- id: unique identifier (required)
- type: NGSI Entity type. It must be equal to “AgriSoil” (required)
- dateCreated: "2017-01-01T01:20:00Z",
- dateModified: "2017-05-04T12:30:00Z",
- name: Name of the soil (required)
- alternateName: Alternative name (Scientific name) of the pest,
- agroVocConcept: URL of the FAO details of the soil,
- seeAlso: Other useful URL,
- relatedSource: Reference application,
- hasAgriProductType: List of product type,
- description: Description of the soil

B.3 Modelling Water (sub) Network and Topologies in NGSI-LD

Water Network Management NGSI-LD data model

The proposed water network management data model is intended to model the main physical entities of a potable water network. The chosen approach derives from the model used by the EPANET simulation platform[19] with the final purpose to execute simulation based on the actual state (pressure, water quality parameters) of the network so to execute scenarios such as predictive evaluation or what-if analysis.

The proposed NGSI-LD data model for Water Network Management is detailed in the FIWARE smart data models repository: https://github.com/smart-data-models/dataModel.WaterNetworkManagement. In this repository, physical entities of the network (Curve, Junction, Pattern, Pipe, Pump, Reservoir, Tank and Valve) are described through their NGSI-LD properties and relationships.

B.4 IoT Enhancement

The next step toward modelling the Water Network management is to connect observations coming from the IoT environment to their appropriate placement in the modelled water network. In fact, IoT devices are installed in several locations and are providing observations about Water quality and more generally about the current state of the Network. These observations have to be connected to the appropriate NGSI-LD entities within the graph representing the water network.

Devices in this context are following the FIWARE Device Data model (https://github.com/smart-data-models/dataModel.Device). Observations are inserted as internal properties of the observed entity. For

[19] EPANET is a simulation tool for water networks allowing to run pressure driven or demand driven scenarios. https://github.com/USEPA/EPANET2.2

example the water Level value provided by a Device installed in a Tank will be added into the "*Tank*" entity as an NGSI-LD Property called "*waterLevel*".

Use Case Example: Water Management Network Physical Deployment

The use case consists on a set of devices measuring water related parameters in a simple Water Network. Examples of observed parameters are chlorine level, the water level in a Tank, the water flow, and the water pressure in different locations of the water network. The main use case elements are depicted in Figure B.3 (below). The water structure in this use case is composed of a Tank, Valves (V_1 and V_2) and a Pump.

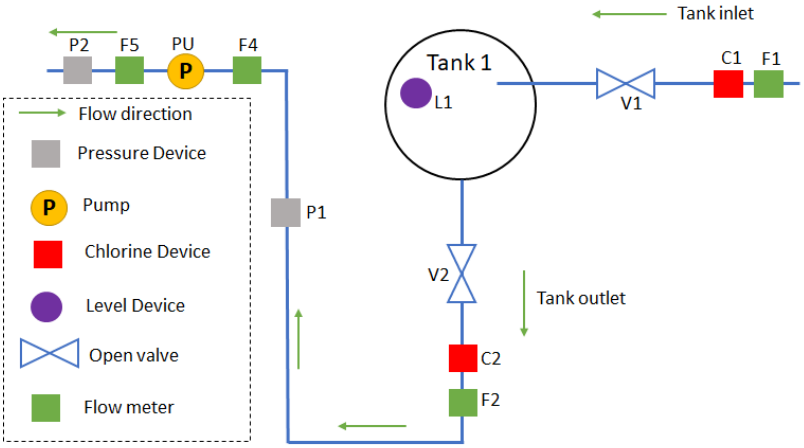


Figure B.3: Water Management Network Use Case

V_1 and V_2 are connected to the Tank for water inlet and outlet respectively. The device L_1 is providing the water level of the Tank. Devices C_1, F_1 and C_2, F_2 are providing chlorine and Flow of the Water inlet and outlet of the Tank. Devices P_1, F_4 and C_2, F_5 are providing pressure and Flow of the Water inlet and outlet of the Pump.

As Tanks, Reservoirs, Junctions, Valves and Pumps present existing objects in a physical real word they are modelled as NGSI-LD entities. A Pipe in a physical word is relating two Water Nodes (Tanks, Reservoirs, and Junctions). This is modelled in NGSI-LD as a relationship relating two entities. However in real use cases a pipe may have a set parameters and temporal properties which evolves over time since it is possible to install on it several type of devices such as Flow meters or chlorine devices. For this purpose a Pipe is also modelled as an NGSI-LD entity.

The pipe P1 is relating the Tank T1 to the Valve V1. The Relationship between these entities depends mainly on the manner of querying stored data in the broker. In the following we present two manner for modelling this use case. The first one is based on the NGSI-LD recommended cross domain ontology. The second one follows more the domain of use case and proposes new Relationships for relating entities

Modelling the water Physical Deployment following the NGSI-LD cross domain ontology

"ConnectsTo" is a recommended NGSI-LD relationship from the cross-domain ontology. It could be used to relate the pipe P1 to the Tank T1 and the Valve V1. The "connectsTo" relationship in the Pipe entity will refer to both Tank and Valve using the multi-specification of NGSI-LD version 1.3. The NGSI-LD Entity Relationship graphical model of this use cases is depicted in Figure B.4.

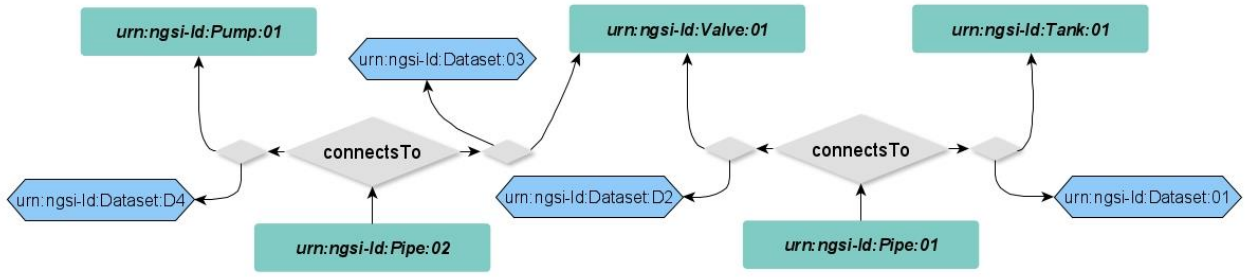


Figure B.4: Entity Relationship use case data model using connectsTo Relationship

An example of serialization of the Pipe P1 entity in NGSI-LD is depicted in the Listing (below)

```
{
    "id": "urn:ngsi-ld:Pipe:01",
    "type": "Pipe",
    "connectsTo": [
        {
                "type": "Relationship",
                "object": "urn:ngsi-ld:Tank:01",
                "datasetId": "urn:ngsi-ld:Dataset:01"
        },
        {
                "type": "Relationship",
                "object": "urn:ngsi-ld:Valve:01",
                "datasetId": "urn:ngsi-ld:Dataset:D2"
        }
     ],
    "@context": [
    "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context.jsonld"
    ]

}
```

Modelling the water Physical Deployment using an extended domain ontology

From a practical point of view and following the water domain experts, the previous modelling approach is not efficient since the information of the water direction in the Pipe P1 entity is excluded and the Relationship connectsTo shows only that the Pipe is connecting the Tank to Valve or the Valve to the Tank with no more details.

The cross domain ontology is a recommended ontology, but the NGSI-LD data model could be extended to define new attributes in order to follow the needs of real uses cases. For this purpose, we defined two relationships “startsAt” and “endsAt” that extend the “connectsTo” relationship by enhancing the detail of the water direction semantically.

The NGSI-LD Entity Relationship graphical model of this use cases is depicted in Figure B.5 (below):

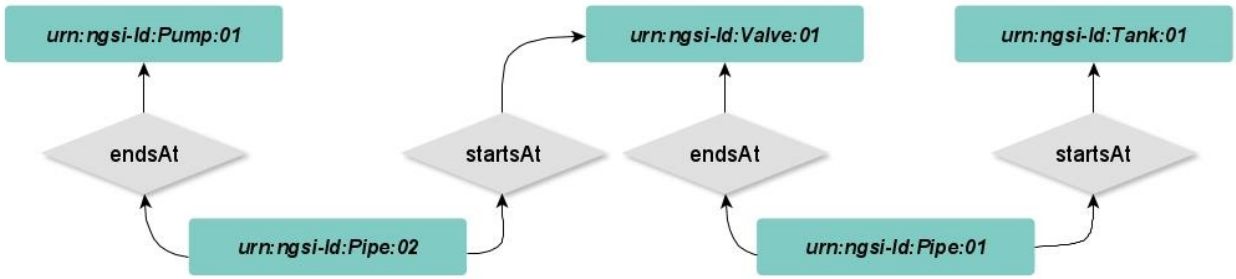


Figure B.5 : Entity Relationship use case data model using startsAt/endsAt Relationships

The Pipe P1 in this case is serialized as follow:

```
{
    "id": "urn:ngsi-ld:Pipe:01",
    "type": "Pipe",
    "startsAt":
        {
                "type": "Relationship",
                "object": "urn:ngsi-ld:Tank:01"
        },
    "endsAt":
        {
                "type": "Relationship",
                "object": "urn:ngsi-ld:Valve:01"
        }
     ],
    "@context": [
    "https://schema.lab.fiware.org/ld/context"
    ]
}
```

Modelling the IoT for water Physical Deployment

Devices L_1, C_1, F_1, C_2, F_2, P_1, F_4, C_2 and F_5 in the previous example follow the FIWARE Device Model (https://github.com/smart-data-models/dataModel.Device). Provided values of these devices are inserted in the defined entities according to their emplacement. In the remaining of this use case description, we concentrate on the Tank Entity. The Tank follows the Tank data model of the Water Management Network of the FIWARE smart data models repository.

The figure below presents the Tank model with 2 properties: The "*waterLevel*" which is observed by the device L1 and the "*chlorine*" with 2 different values (for inlet and outlet) modelled using the multi-attribute aspect of NGSI-LD which are observed by devices C1 and C2.

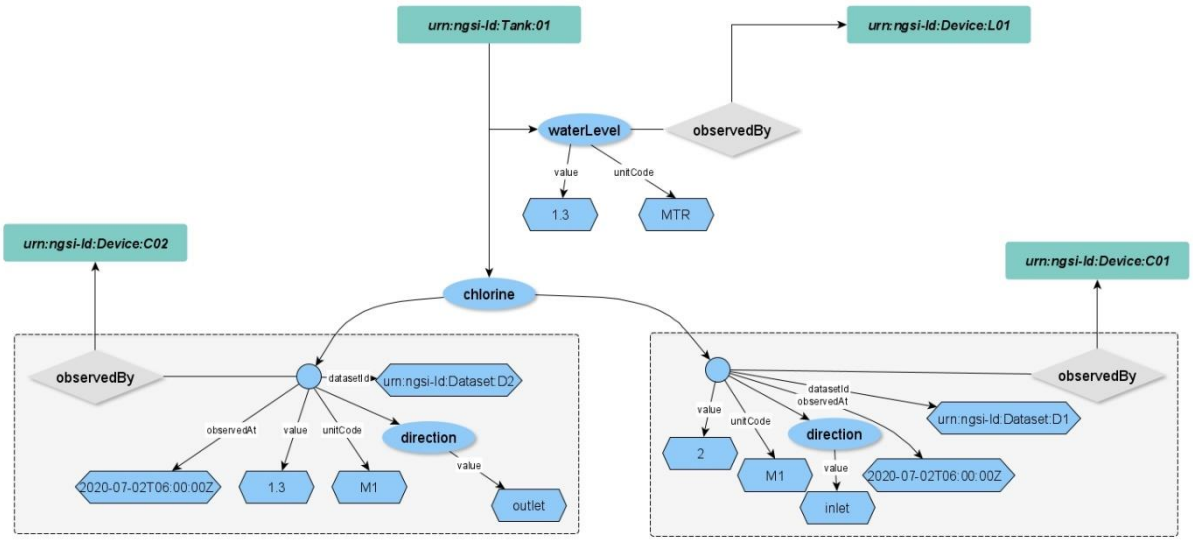


Figure B.6: Integrating IoT data within the water network data model

Modelling the Water Network Topology

The existing water Networks are not only defined through physical entities (Curve, Junction, Pattern, Pipe, Pump, Reservoir, Tank and Valve), they are also often organised as several sets of connected sub-networks where the physical entities may be a (sub) part of them. In a water distribution network, this is generally called sectorisation or district metered area (DMA). These sub-networks can also be defined by their emplacements (A sub-network by a zone, by city, by road….), by their functionalities (A sub-network for Treatment Plant) or by any other needs according the real use cases.

It is thus important to represent these sectorisations within the model to ease its management. As an example, a predictive simulation can be executed against any of the sector depending on the operational needs. In the following, an approach is proposed to make such a modelling using capabilities of v1.3.1 of the NGSI-LD specification.

To model the Water Network Topologies in NGSI-LD, we define these rules:

- All Water Networks and Water Sub-Networks are defined as "WaterNetwork" NGSI-LD graph entities.
- To relate a sub-Network to its components in both directions , we use the "isNodeOfGraph" NGSI-LD relationship from the cross-domain ontology and we introduce an additional reciprocal relationship : "comprises".
- To relate a Network to its component subnetworks in both directions, we use the "isSubGraphof" NGSI-LD relationship from the cross-domain ontology and we and we introduce an additional reciprocal relationship : 'hasSubGraph".
- In order to support the multi-sub-Network belonging, the NGSI-LD multi-attribute aspect is supported.
-

    a. ***"isNodeOfGraph" and "comprises" Relationships***

The “isNodeOfGraph” is recommended relationship issued from the NGSI-LD cross-domain ontology used for describing a bottom-up federated/distributed kind of system composition. Let’s define two water Networks: “Treatment Plant” and “DMA1”. The Tank T1 and the Pipe P1 belong to the Treatment Plant water Network. The Pipes P1 and P1, the Valve V1 and the Pump Pu1 belong to the DMA1 water Network. A graphical presentation of this data model is depicted in Figure B.7 (below).

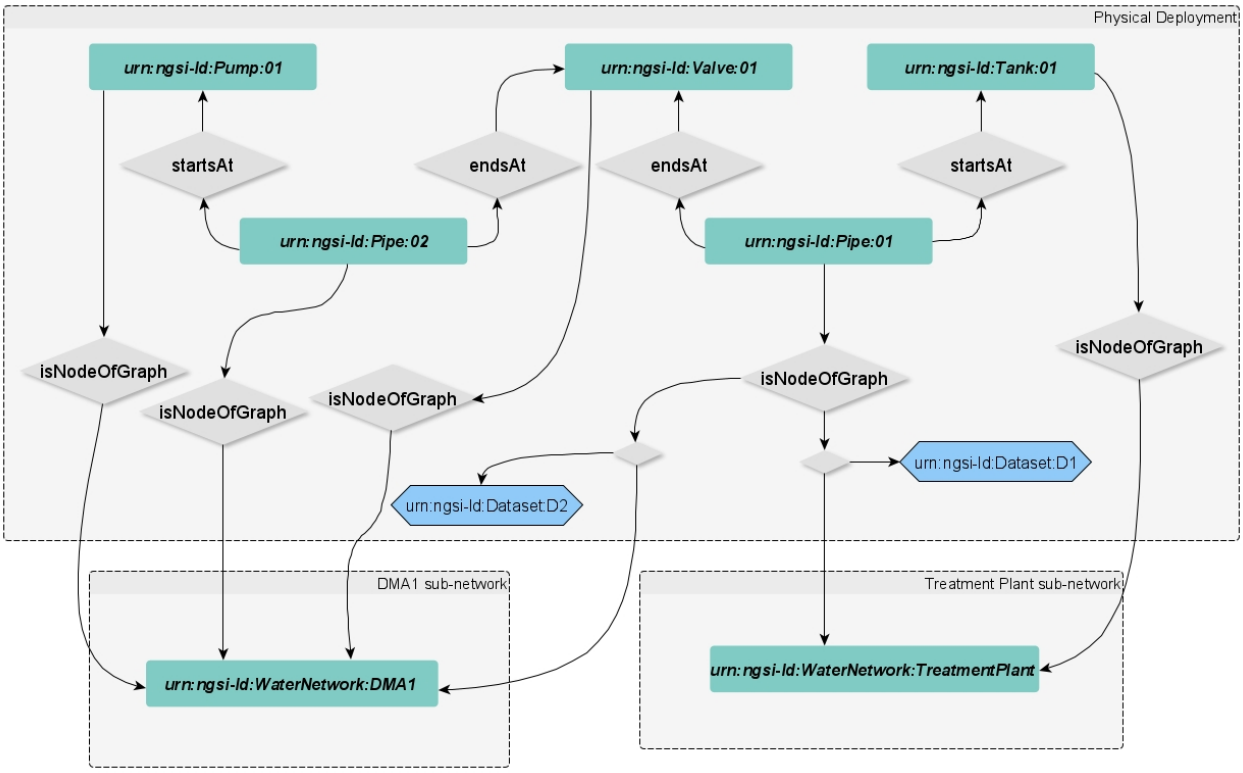


Figure B.7: Modelling Water Network Topology using endograph Relationship

The Pipe P1 in this case is serialized as follow:


```
{
    "id": "urn:ngsi-ld:Pipe:01",
    "type": "Pipe",
    "isNodeOfGraph": [
        {
                "type": "Relationship",
                "object": "urn:ngsi-ld:WaterNetwork:TreatmentPlant",
                "datasetId": "urn:ngsi-ld:Dataset:01"
        },
        {
                "type": "Relationship",
                "object": "urn:ngsi-ld:WaterNetwork:DMA1
                "datasetId": "urn:ngsi-ld:Dataset:D2"
        }
     ],
    "@context": [
    "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context.jsonld"
    ]
```


From a practical point of view and following the water domain experts, the objective of water topology modelling is to model simulation scenarios. A simulation scenarios in the Water domain is always processed in a water network. Thus the Water Network NGSI-LD entities may have the details about all physical component of the water network. This detail is missing in this model since the Relationship endograph is serialized and stored in the water component entities such as the Pipe P1.

For this purpose we introduce the "*comprises*" NGSI-LD relationship, used for defining the components of a water Network. The "DMA1" in this case is composed of Pipe1, Valve1, Pipe2 and Junction1. The Treatment Plant is composed of Pipe1, Valve1.

A graphical presentation of this data model is depicted in Figure B.8 (below).

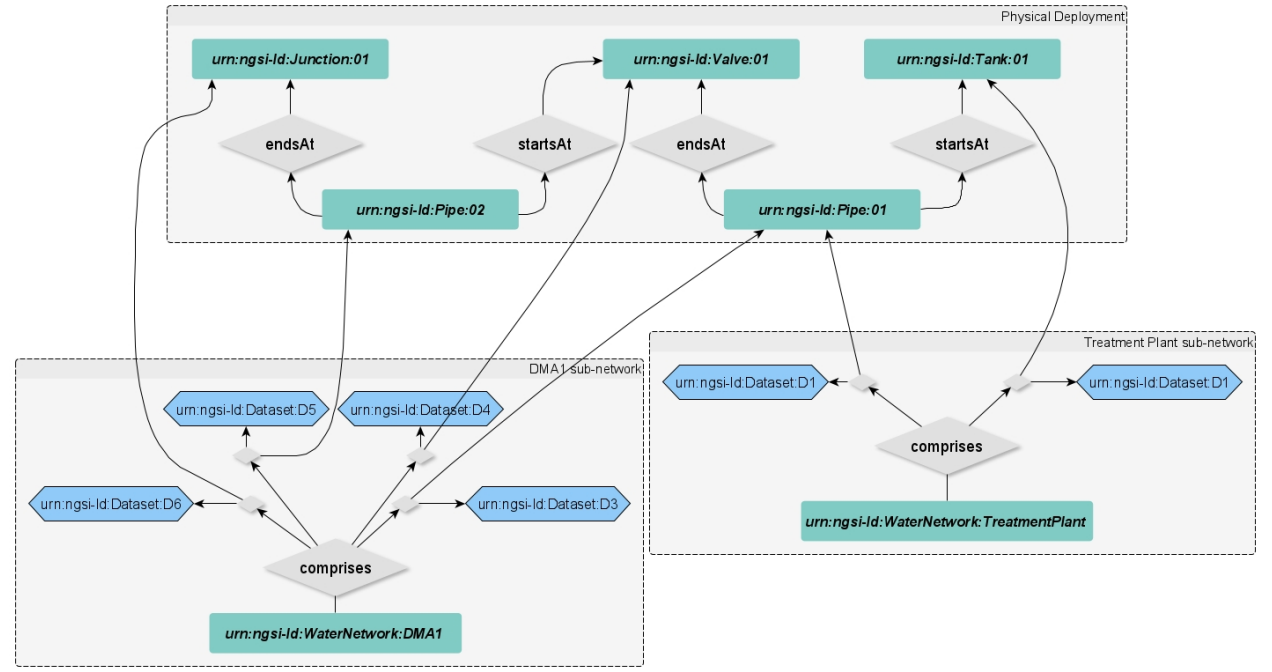


Figure B.8 Modelling Water Network Topology using comprises Relationship

The Treatment Plant Water Network entity in this case is serialized as follows:

```
{
    "id": "urn:ngsi-ld:WaterNetwork:TreatmentPlant",
    "type": "WaterNetwork",
    " comprises": [
        {
                "type": "Relationship",
                "object": "urn:ngsi-ld:Pipe:01",
                "datasetId": "urn:ngsi-ld:Dataset:01"
        },
        {
                "type": "Relationship",
                "object": " urn:ngsi-ld:Tank:01"
                "datasetId": "urn:ngsi-ld:Dataset:D2"
        }
     ],
    "@context": [
    "https://schema.lab.fiware.org/ld/context"
    ]
```

b. *"isograph" and "hasSubGraph" Relationhips*

The previously defined water networks have to be linked to their super network to create the complete topology of the water network. The network is composed of a set of sub-networks and each sub-network may also be composed of a set of (sub) sub-networks. Let's suppose that the previously defined water sub-network "*DMA1*" and "*TreatmentPlant*" are belonging to the "*SophiaWaterNetwork*".

The "isSubGraphOf" is recommended relationship issued from the NGSI-LD cross-domain ontology used for describing system composition of top-down-designed systems. A graphical presentation of this data model is depicted in Figure (below). As for the previous example, the isSubGraphOf will be serialized and stored in the Water Network entities DMA1 and Treatment Plant. Thus, the component of Sophia water network is missed since that the "isSubGraphOf" is in the sub-networks entities.

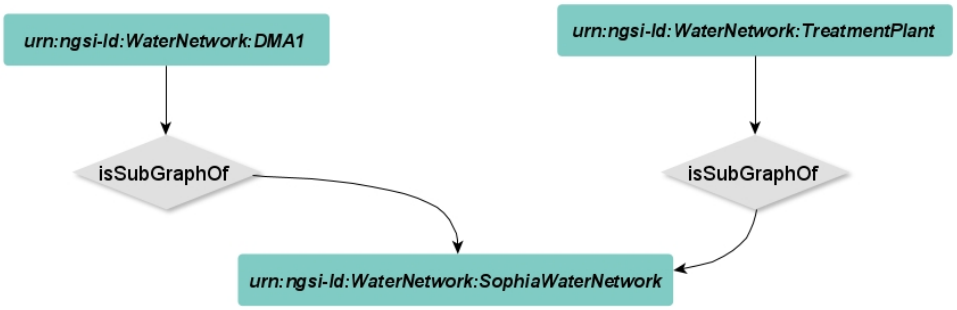


Figure B.9: Modelling Water Network Topology using isSubGraphOf Relationship

We introduced the NGSI-LD relationship "*hasSubGraph*" which is intended to relate a water network to its sub-networks. The sub-networks in the case are modelled in the Sophia water Network entity.

A graphical presentation of this data model is depicted in Figure B.10 (below).

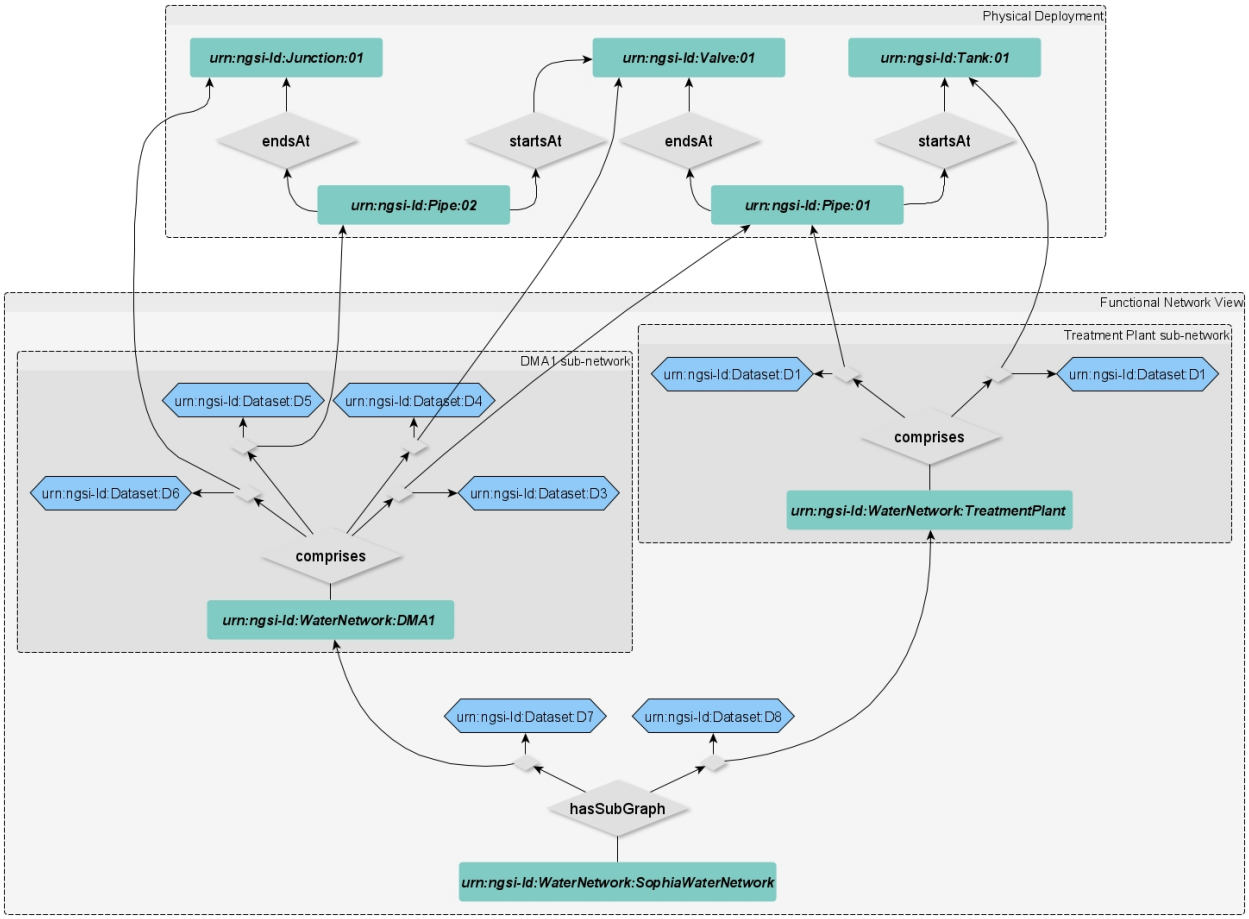


Figure B.10: Modelling Water Network Topology using hasSubGraph Relationship

The Sophia Water Network entity in this case is serialized as follow:

```
{
    "id": "urn:ngsi-ld:WaterNetwork:SophiaWaterNetwork",
    "type": "WaterNetwork",
    " hasSubGraph": [
        {
                "type": "Relationship",
                "object": "urn:ngsi-ld:WaterNetwork:DMA1",
                "datasetId": "urn:ngsi-ld:Dataset:07"
        },
        {
                "type": "Relationship",
                "object": "urn:ngsi-ld:WaterNetwork:TreatmentPlant"
                "datasetId": "urn:ngsi-ld:Dataset:D8"
        }
     ],
    "@context": [
    "https://schema.lab.fiware.org/ld/context"
    ]
```

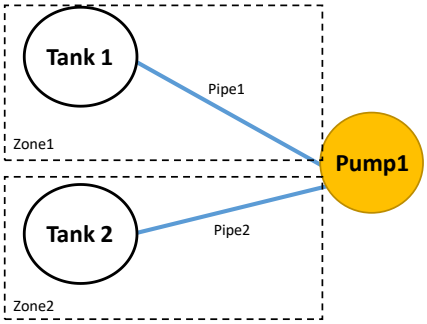
c. Generalities

Summarizing the previous NGSI-LD Specifications of modelling the water network topology:

- A Network entity is linked to its sub-Networks (semantically, considered as Networks) via the extended NGSI-LD relationship "*hasSubGraph*" or via the recommended Relationship "isSubGraphOf" in a federated system composition format.
- A Network entity is linked to its components (junction, tank….) using the NGSI-LD relationship "comprises".
- A component entity is linked to its Water Networks (junction, tank….) using the NGSI-LD recommended relationship "isNodeOfGraph".
- A Network can be composed of a set of networks (via the relationship "*hasSubGraph*") and a set of components (via the relationship "*comprises*").

An Illustrative Example:

The water network called "*SophiaWaterNetwork*" is now composed of a Pump and tow sub-networks called "*Zone1*" and "*Zone2*". For simplicity each zone is only composed of a Pipe and a Tank. This example is illustrated in Figure B.11 below:



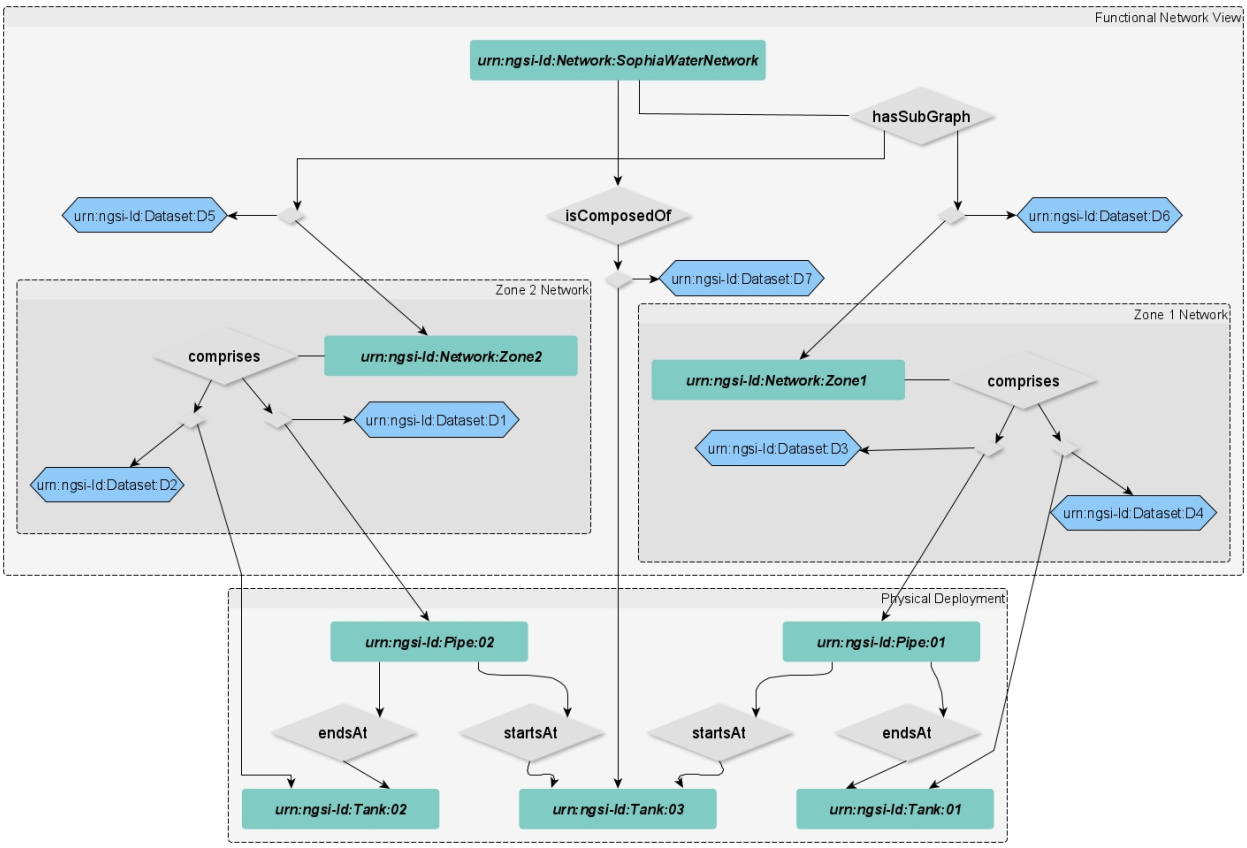
Sophia Water Network

Figure B.11

Following the previous specifications for the network topology design, the NGSI-LD data model will be composed of:

- Physical Deployment which contains the water network components (Pump, Pipes and Tanks) and their Properties and Relationships.
- The "*SophiaWaterNetwork*" Entity have two sub-Networks "*Zone1*" and "*Zone2*" linked to them via the "*hasSubGraph*" relationship and one Water component (*Pump1*) linked to it via the "*comprises*" relationship.
- The (sub) Networks "*Zone1*" and "*Zone2*" are linked to their component via the "*hasPart*" relationship.

- The NGSI-LD graphical representation of this model is depicted in the Figure B.12 (below).



-

Figure B.12: Modelling the use case example using specific domain ontology

Following the NGSI-LD recommended ontology the data model is depicted in the Figure B.13 below.

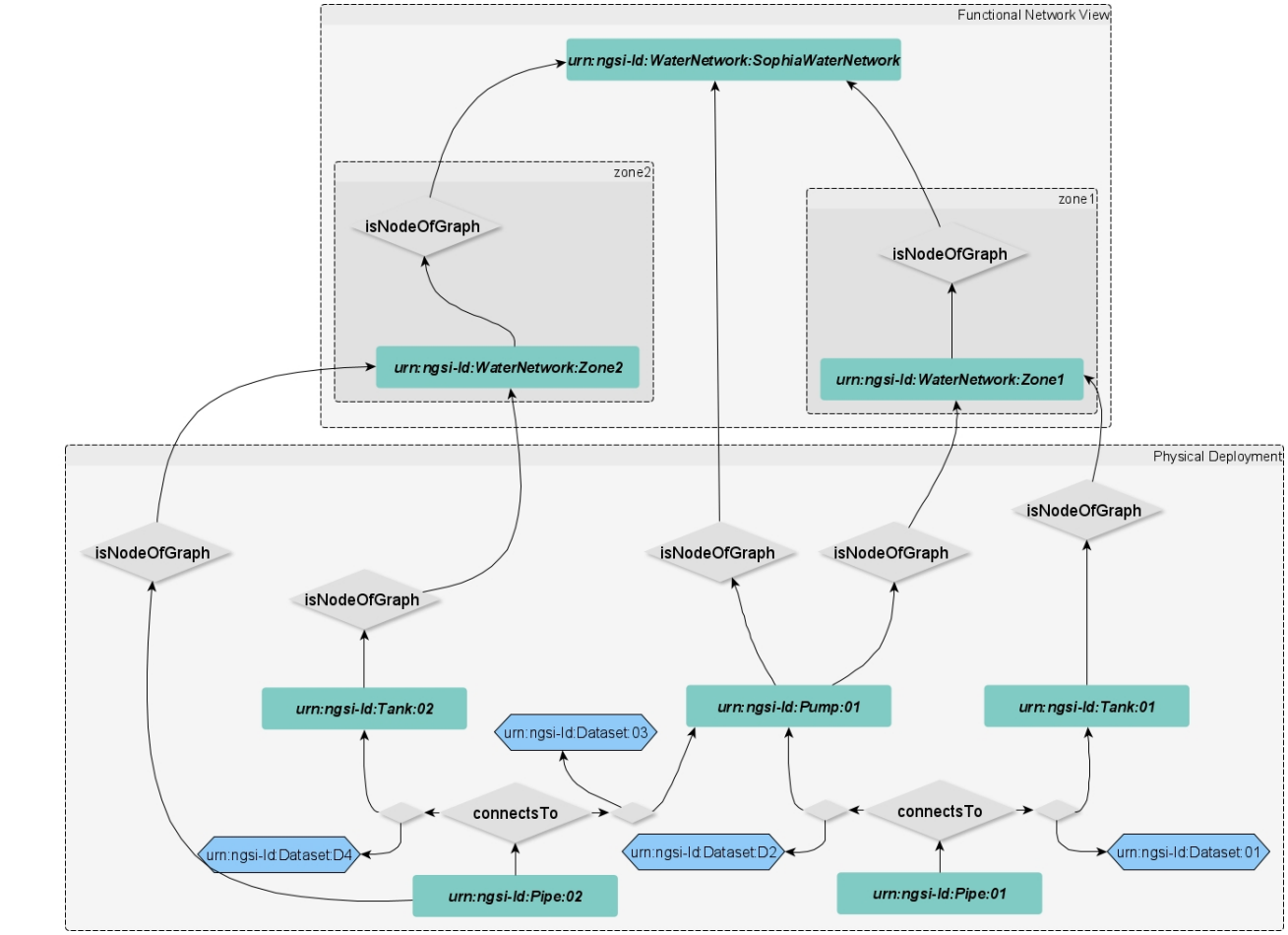


Figure B.13: Modelling the use case example using the recommended NGSI-LD ontology

ETSI
06921 Sophia Antipolis CEDEX, France
Tel +33 4 92 94 42 00
info@etsi.org
www.etsi.org