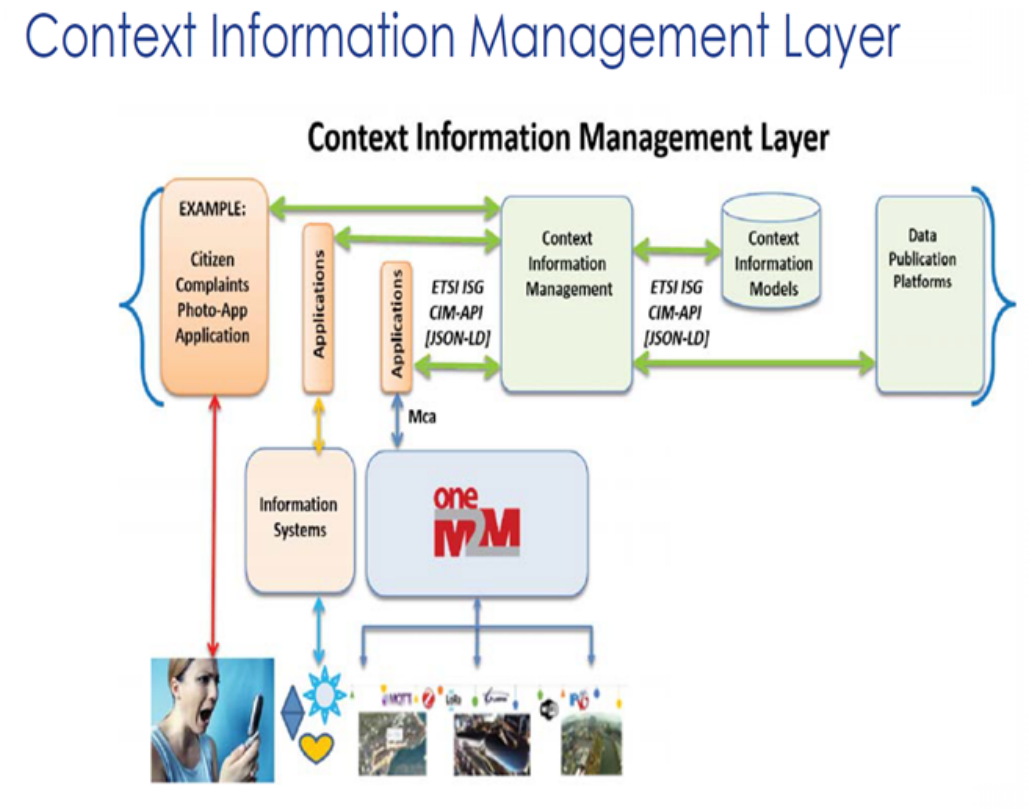
oneM2M IoT SL and CIM NGSI-LD (Context Information Management Next Generation Service Interface - Linked Data)

oneM2M and CIM NGSI-LD (Context Information Management Next Generation Service Interface - Linked Data)

The Goal of the ETSI CIM ISG on Context Information Management is to issue TSs to enable multiple Organisations to develop interoperable SW implementations of a cross-cutting Context Information Management (CIM) Layer.



The CIM Layer should enable Applications to Discover, Access, Update and Manage Context information from many different sources, as well as publish it through interoperable Data Publication Platforms.

Phase 1 - detect and describe the Standardization Gaps.

Phase 2 - Developing ISG CIM Group Specifications in Phase 2 will subsequently fill these gaps. It is expected that an extension of the RESTful binding of the OMA NGSI API involving expression using JSON-LD could aid interoperability, so this and potentially other extensions will be considered.

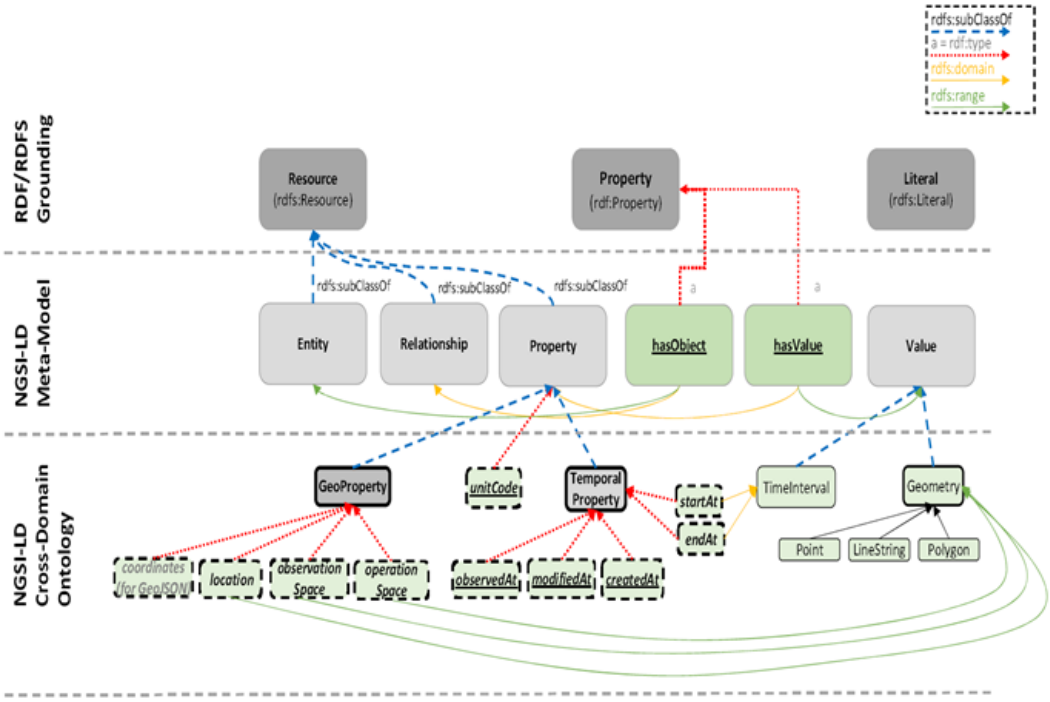


Figure 4.2.3-1: NGSI-LD Core Meta-Model plus the Cross-Domain Ontology

The CIM API allows Users to Provide, Consume and Subscribe to **Context Information c**lose to Real-time Access to Information coming from many different Sources (not only IoT Data Sources).

B.1 Mapping to oneM2M

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

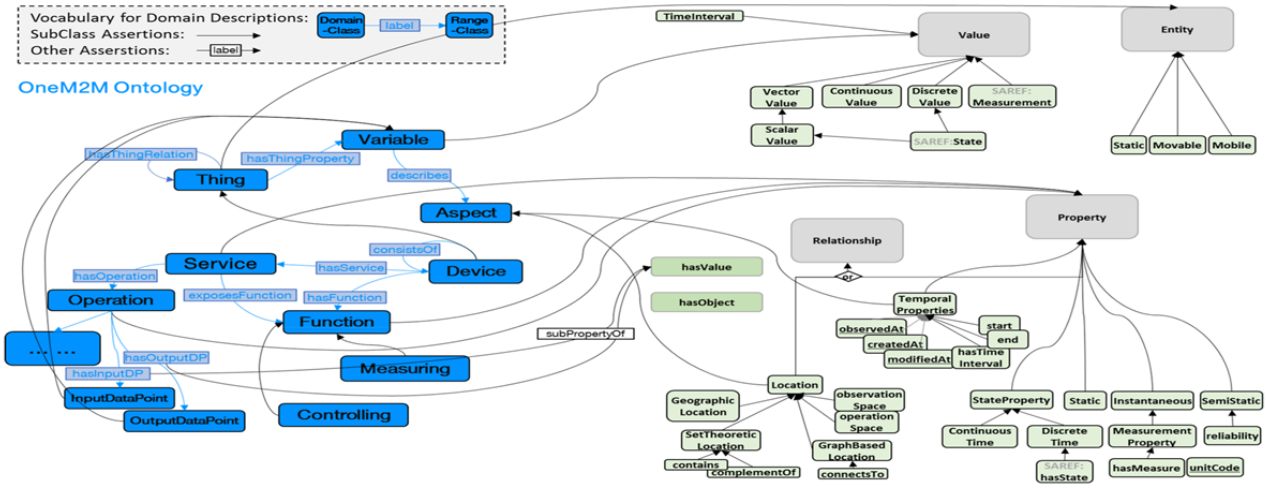


Fig. B. 1: Mapping NGSI - LD Meta - model and Cross-Domain Ontology to oneM2M Base Ontology

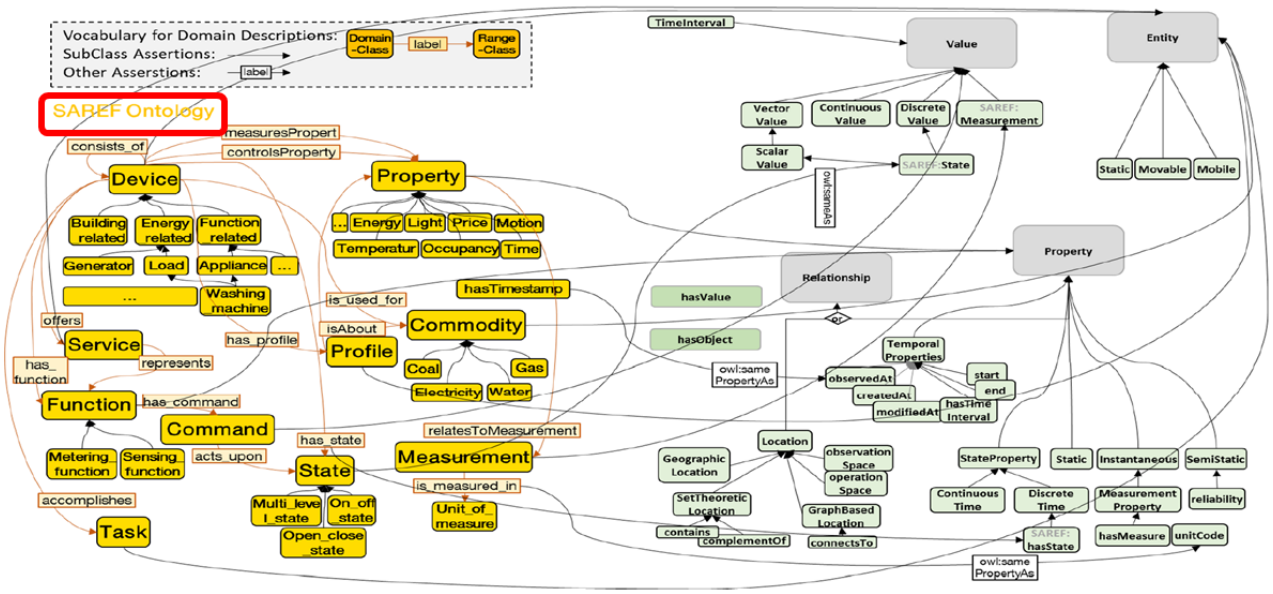


Figure B.4: Mapping NGSI-LD to SAREF

Core NGSI-LD @context

NGSI-LD serialization is based on JSON-LD , a JSON-based format to serialize Linked Data. The @context in JSON-LD is used to expand terms, provided as short hand strings, to concepts, specified as URIs, and vice versa, to compact URIs into terms. The Core NGSI-LD (JSON-LD) @context is defined as a JSON-LD @context which

contains:

- The core terms needed to uniquely represent the key concepts defined by the NGSI-LD Information Model, as mandated by clause 4.2.

- The terms needed to uniquely represent all the members that define the API-related Data Types, as mandated by clauses 5.2 and 5.3.

- A fallback *@vocab* rule to expand or compact user-defined terms to a default URI, in case there is no other possible expansion or compaction as per the current @context.

- The core NGSI-LD @context defines the term "id", which is mapped to "@id", and term "type", which is mapped to "@type". Since @id and @type are what is typically used in JSON-LD, they may also be used in NGSI-LD requests instead of "id" and "type" respectively, wherever this is applicable. In NGSI-LD responses, only "id" and "type" shall be used.

NGSI-LD compliant implementations shall support such Core @context, which shall be implicitly present when processing or generating context information. Furthermore, the Core @context is protected and shall remain immutable and invariant during expansion or compaction of terms. Therefore, and as per the JSON-LD processing rules [2], when processing NGSI-LD content, implementations shall consider the Core @context as if it were in the **last** position of the @context array. Nonetheless, for the sake of compatibility and cleanness, data providers should generate JSON-LD content that conveys the Core @context in the last position.

For the avoidance of doubt, when rendering NGSI-LD Elements, the Core @context **shall always be treated** as if it had been originally placed **in the last position**, so that, if needed, upstream JSON-LD processors can properly expand as NGSI-LD or override the resulting JSON-LD documents provided by API implementations.

The NGSI-LD Core @context is publicly available at https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.3.jsonld and shall contain all the terms as mandated by Annex B.

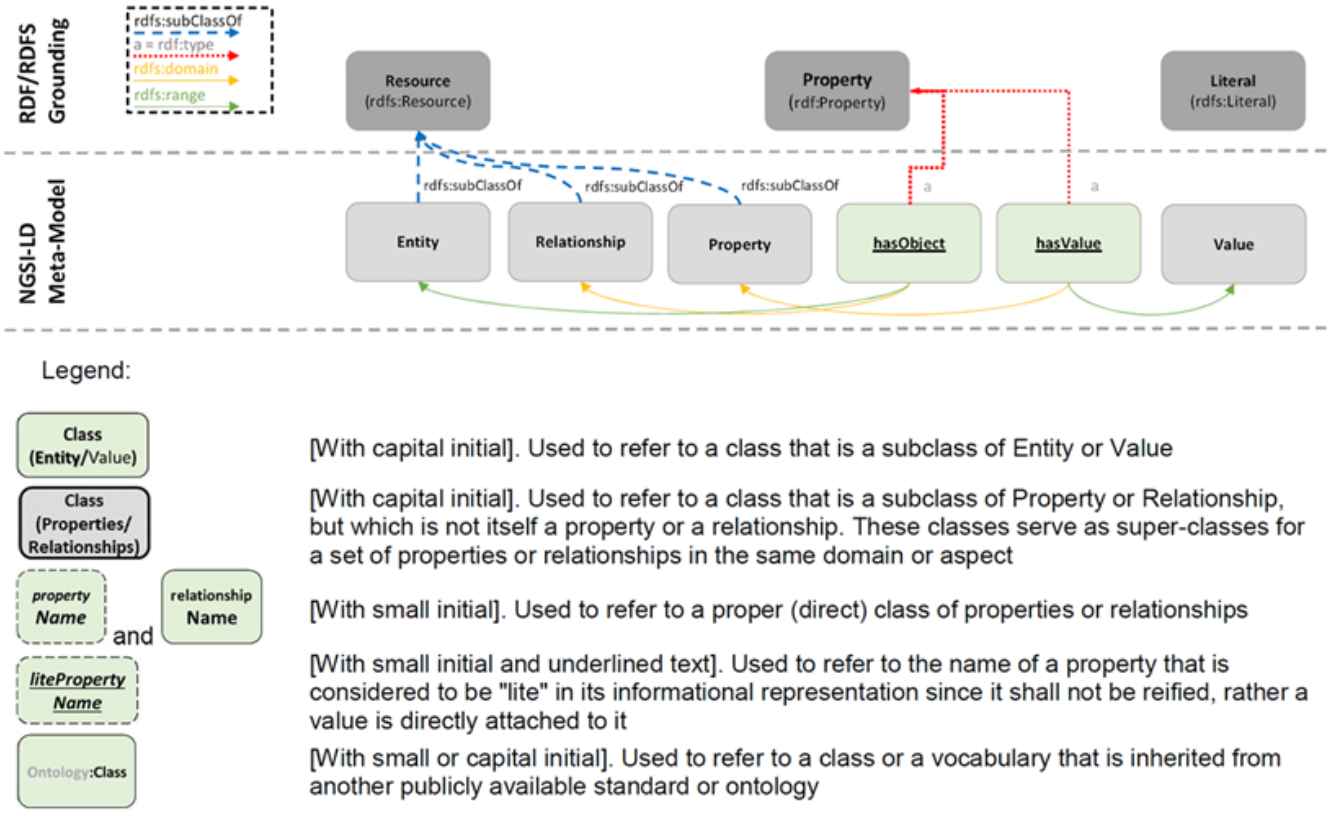


Figure 4.2.2-1: NGSI-LD Core Meta-Model

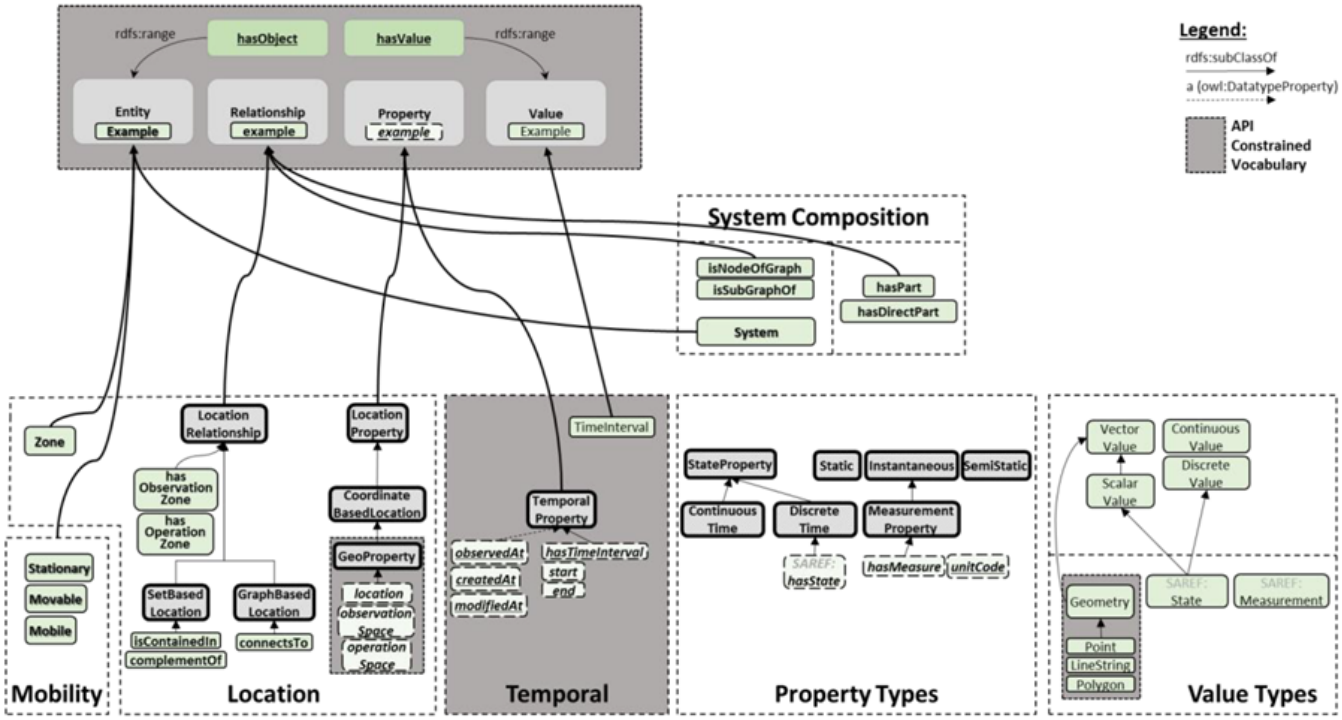


Figure 1 : NGSI-LD Cross-Domain Ontology, with referenced Meta-model[5]

Using Typing vs. using Relationships or Propertiesin ETSI CIM NGSI-LD

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.

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.

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.

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 domain8.

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.

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 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 attributes9 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.

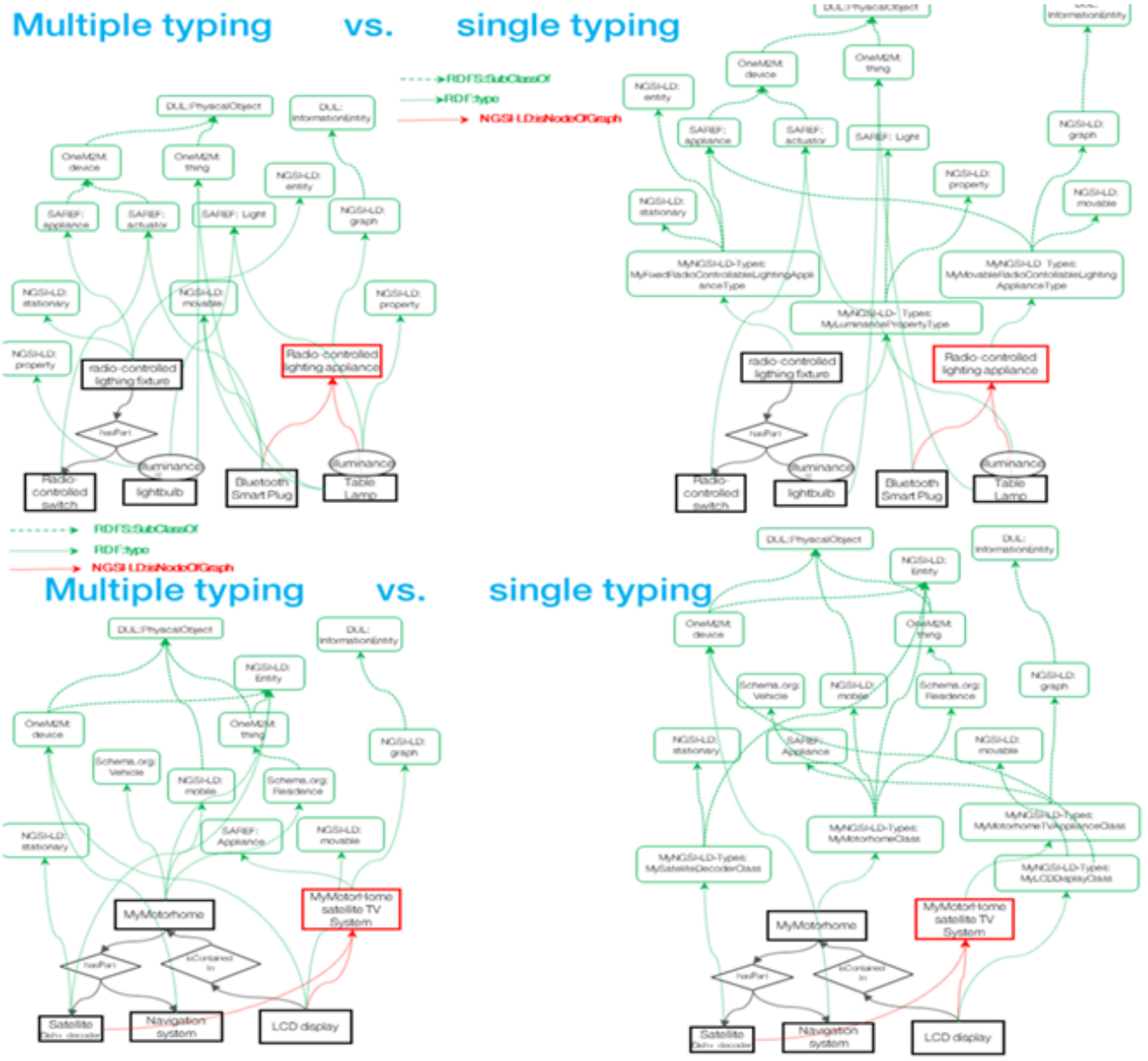


Figure A.1: Examples of different modelling choices depending on the use of multiple typing (on the left) vs. single typing (on the right)

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.