ETSI GR CIM 021 V1.1.1 (2022-08)

cross-cutting Context Information Management (CIM); usage of external data models with NGSI-LD API

Disclaimer

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

Reference
DGR/CIM-0021

Keywords
API, data models, Digital Twins, NGSI-LD

ETSI

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

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

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

Contents

Intellectual Property Rights

Essential patents

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

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

Trademarks

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

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

Foreword

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

Modal verbs terminology

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

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

Executive summary

The present document provides an analysis of a small representative of data models that maps different domains (Smart Cities, Communities, Geo-spatial Objects, etc.) and that are defined in different technologies (OWL, SKOS, RDF, JSON Schema). The goal of the analysis was to retrieve and formulate recommendations to use external data models with NGSI-LD (see ETSI GS CIM 009 [i.26]). Recommendations come as examples, suggestions and guidelines to allow users to link external data models inside NGSI-LD [i.26] Entities. Lastly, the analysis highlighted possible new features to add inside NGSI-LD [i.26]. Each proposal comes with a brief description of the use case and the reasons to support its addition inside the NGSI-LD [i.26] specification.

Introduction

There is a tremendous amount of data models being developed the needs of businesses in all verticals. The complexity of these models and the associated technologies can vary from simple tabular files (e.g. .csv) to complex OWL 2 ontologies. NGSI-LD aims at providing a cross-domain meta model based on property graphs together with an API to interact with context information.

It is important to empower NGSI-LD users how to use various ontologies: it is **not** the intent to create **new** ontologies, but to describe options to **make use of external vocabularies, taxonomies**, thesauri and ontologies within an NGSI-LD context and to describe how to articulate them with the NGSI-LD graph-based meta-model and cross-domain ontology. Some are relatively simple, providing multi-lingual definitions of terms in a structure way. This is the case of the Agrovoc [i.17] thesaurus based on SKOS. Others adds restrictions on the fields of a data model, such as defining the allowed data range of a data field. This is the case of the schema.org initiative. And others go further providing means for reasoning over the data structure which can provide means for data quality control, new properties discovery, etc. This is provided by for example, OWL 2 ontologies.

It is thus very important to make clear the potential of approaches listed above and illustrate their use within a NGSI-LD deployment to increase the usage of the cross-domain capabilities of NGSI-LD, by using working examples.

1 Scope

The present document considers a selection of relevant and representative knowledge organization systems, to provide an illustrative basis of data schemas and vocabularies relevant to be used in a NGSI-LD context. It provides recommendations on how to link external data models in NGSI-LD, considering the illustration of their use with the NGSI-LD API in complex environments, such as system-of-system models for Digital Twins. The present document also provides suggestions for changes or enhancements to the NGSI-LD specification (API and data model).

Unless expressly stated, the NGSI-LD version referenced in the present document is 1.5.1.

2 References

2.1 Normative references

Normative references are not applicable in the present document.

2.2 Informative references

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

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

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

[i.1] ETSI TS 103 264 (V3.1.1): "SmartM2M; Smart Applications; Reference Ontology and oneM2M Mapping".

NOTE: Available at: https://www.etsi.org/deliver/etsi_ts/103200_103299/103264/03.01.01_60/ts_103264v030101p.pdf.

[i.2] ETSI TS 103 410-4 (V1.1.2): "SmartM2M; Extension to SAREF; Part 4: Smart Cities Domain".

NOTE: Available at: https://www.etsi.org/deliver/etsi_ts/103400_103499/10341004/01.01.02_60/ts_10341004v010102p.pdf.

[i.3] ETSI TS 103 410-1 (V1.1.2): "SmartM2M; Extension to SAREF; Part 1: Energy Domain".

NOTE: Available at: https://www.etsi.org/deliver/etsi_ts/103400_103499/10341001/01.01.02_60/ts_10341001v010102p.pdf.

[i.4] ETSI TS 103 410-10 (V1.1.1): "SmartM2M; Extension to SAREF; Part 10: Water Domain".

NOTE: Available at: https://www.etsi.org/deliver/etsi_ts/103400_103499/10341010/01.01.01_60/ts_10341010v010101p.pdf.

[i.5] ETSI TS 103 410-8 (V1.1.1): "SmartM2M; Extension to SAREF; Part 8: eHealth/Ageing-well Domain".

NOTE: Available at: https://www.etsi.org/deliver/etsi_ts/103400_103499/10341008/01.01.01_60/ts_10341008v010101p.pdf.

[i.6] ETSI TR 103 509: "SmartM2M; SAREF extension investigation; Requirements for eHealth/Ageing-well".

NOTE: Available at: https://www.etsi.org/deliver/etsi_tr/103500_103599/103509/01.01.01_60/tr_103509v010101p.pdf.

[i.7] ETSI TS 103 410-3 (V1.1.2): "SmartM2M; Extension to SAREF; Part 3: Building Domain".

NOTE: Available at: https://www.etsi.org/deliver/etsi_ts/103400_103499/10341003/01.01.02_60/ts_10341003v010102p.pdf.

[i.8] ETSI TS 103 410-5 (V1.1.2): "SmartM2M; Extension to SAREF; Part 5: Industry and Manufacturing domains".

NOTE: Available at: https://www.etsi.org/deliver/etsi_ts/103400_103499/10341005/01.01.02_60/ts_10341005v010102p.pdf.

[i.9] ETSI TS 103 410-6 (V1.1.2): "SmartM2M; Extension to SAREF; Part 6: Smart Agriculture and Food Chain Domain".

NOTE: Available at: https://www.etsi.org/deliver/etsi_ts/103400_103499/10341006/01.01.02_60/ts_10341006v010102p.pdf.

[i.10] ETSI TS 103 410-7 (V1.1.1): "SmartM2M; Extension to SAREF; Part 7: Automotive Domain".

NOTE: Available at: https://www.etsi.org/deliver/etsi_ts/103400_103499/10341007/01.01.01_60/ts_10341007v010101p.pdf.

[i.11] ETSI TS 103 410-9 (V1.1.1): "SmartM2M; Extension to SAREF; Part 9: Wearables Domain".

NOTE: Available at: https://www.etsi.org/deliver/etsi_ts/103400_103499/10341009/01.01.01_60/ts_10341009v010101p.pdf.

[i.12] ETSI TS 103 410-11 (V1.1.1): "SmartM2M; Extension to SAREF; Part 11: Lift Domain".

NOTE: Available at: https://www.etsi.org/deliver/etsi_ts/103400_103499/10341011/01.01.01_60/ts_10341011v010101p.pdf.

[i.13] ETSI TS 103 548 (V1.1.2): "SmartM2M; SAREF consolidation with new reference ontology patterns, based on the experience from the SEAS project".

NOTE: Available at: http://www.etsi.org/deliver/etsi_ts/103500_103599/103548/01.01.02_60/ts_103548v010102p.pdf.

[i.14] ETSI TS 103 410-2 (V1.1.2): "SmartM2M; Extension to SAREF; Part 2: Environment Domain".

NOTE: Available at: https://www.etsi.org/deliver/etsi_ts/103400_103499/10341002/01.01.02_60/ts_10341002v010102p.pdf.

[i.15] Schema.org About Web Page.

NOTE: Available at: https://schema.org/docs/about.html.

[i.16] SKOS Simple Knowledge Organization System RDF Schema.

NOTE: Available at: https://www.w3.org/TR/2008/WD-skos-reference-20080829/skos.html.

[i.17] Agrovoc About Web Page.

NOTE: Available at: https://agrovoc.fao.org/browse/en/about.

[i.18] Smart Data Models About Web Page.

NOTE: Available at: https://smartdatamodels.org/index.php/about/.

[i.19] W3C® DCAT2 (V2): "Data Catalog Vocabulary (DCAT) - Version 2".

NOTE: Available at: https://www.w3.org/TR/vocab-dcat-2/.

[i.20] CIDOC-CRM (V7.2): "Volume A: Definition of the CIDOC Conceptual Reference Model".

NOTE: Available at: https://www.cidoc-crm.org/sites/default/files/cidoc_crm_version_7.2.pdf.

[i.21] OGC 11-052r4 (V1.0): "OGC GeoSPARQL - A Geographic Query Language for RDF Data".

NOTE: Available at: http://www.opengis.net/doc/IS/geosparql/1.0.

[i.22] OGC 11-052r5 (V1.1): "OGC GeoSPARQL - A Geographic Query Language for RDF Data".

NOTE: Available at: https://opengeospatial.github.io/ogc-geosparql/geosparql11/spec.html.

[i.23] INSPIRE-MIF (V0.1): "INSPIRE UML-to-GeoJSON encoding rule".

NOTE: Available at: https://github.com/INSPIRE-MIF/2017.2/blob/master/GeoJSON/geojson-encoding-rule.md.

[i.24] FoaF (V0.1): "INSPIRE UML-to-GeoJSON encoding rule".

NOTE: Available at: https://github.com/INSPIRE-MIF/2017.2/blob/master/GeoJSON/geojson-encoding-rule.md.

[i.25] WoT (V0.1): "INSPIRE UML-to-GeoJSON encoding rule".

NOTE: Available at: https://github.com/INSPIRE-MIF/2017.2/blob/master/GeoJSON/geojson-encoding-rule.md.

[i.26] ETSI GS CIM 009 (V1.5.1): "Context Information Management (CIM); NGSI-LD API".

NOTE: Available at: https://www.etsi.org/deliver/etsi_gs/CIM/001_099/009/01.05.01_60/gs_CIM009v010501p.pdf.

[i.27] Maxat Kulmanov, Fatima Zohra Smaili, Xin Gao, Robert Hoehndorf: "Semantic similarity and machine learning with ontologies, Briefings in Bioinformatics", Volume 22, Issue 4, July 2021, bbaa199.

NOTE: Available at: https://doi.org/10.1093/bib/bbaa199.

[i.28] Ali Ayadi, Ahmed Samet, François de Bertrand de Beuvron, Cecilia Zanni-Merk: "Ontology population with deep learning-based NLP: a case study on the Biomolecular Network Ontology, Procedia Computer Science", Volume 159, 2019, ISSN 1877-0509.

NOTE: Available at: https://doi.org/10.1016/j.procs.2019.09.212.

[i.29] Jalila Filali, Hajer Baazaoui Zghal, Jean Martinet: "Ontology-Based Image Classification and Annotation; International Journal of Pattern Recognition and Artificial Intelligence", 2020 34:11.

NOTE: Available at: https://www.researchgate.net/profile/Jean-Martinet/publication/337793103_Ontology-Based_Image_Classification_and_Annotation/links/60210944299bf1cc26ae92bf/Ontology-Based-Image-Classification-and-Annotation.pdf.

[i.30] J. Liu, X. Zhang, Y. Li, J. Wang and H. -J. Kim: "Deep Learning-Based Reasoning With Multi-Ontology for IoT Applications", in IEEE™ Access, vol. 7, pp. 124688-124701, 2019.

NOTE: Available at: https://doi.org/10.1109/ACCESS.2019.2937353.

[i.31] H. Paulheim. Knowledge Graph Refinement:"A Survey of Approaches and Evaluation Methods", Semantic Web Journal, (Preprint):1-20, 2016.

NOTE: Available at: http://www.semantic-web-journal.net/system/files/swj1167.pdf.

[i.32] Google®, Advanced SEO: "Understand how structured data works".

NOTE: Available at: https://developers.google.com/search/docs/advanced/structured-data/intro-structured-data?hl=en.

[i.33] W3C®, JSON-LD (v1.1): "A JSON-based Serialization for Linked Data".

NOTE: Available at: https://www.w3.org/TR/json-ld11/.

[i.34] Open Geospatial Consortium Inc. OGC 06-103r4: "OpenGIS® Implementation Standard for Geographic information - Simple feature access - Part 1: Common architecture".

NOTE: Available at: https://portal.opengeospatial.org/files/?artifact_id=25355.

[i.35] IEEE 802.15™ Working Group for Wireless Specialty Networks (WSN).

NOTE: Available at: https://www.ieee802.org/15.

[i.36] ISO/IEC CD 39075: "Information Technology - Database Languages - GQL".

[i.37] "Meet openCypher: The SQL for Graphs - Neo4j Graph Database". Neo4j Graph Database. 2015-10-21. Retrieved 2019-11-08.

NOTE: Available at: https://neo4j.com/blog/open-cypher-sql-for-graphs.

[i.38] ETSI TR 103 511: "SmartM2M; SAREF extension investigation; Requirements for AgriFood domain".

[i.39] ETSI TR 103 508: "SmartM2M; SAREF extension investigation; Requirements for Automotive".

[i.40] ETSI TR 103 510: "SmartM2M; SAREF extension investigation; Requirements for Wearables".

[i.41] ETSI TR 103 546: "SmartM2M; Requirements & Feasibility study for Smart Lifts in IoT".

[i.42] ETSI TS 103 735: "SmartM2M; Smart Lifts IoT System".

[i.43] ISO 6523-1:1998: "Information technology -- Structure for the identification of organizations and organization parts -- Part 1: Identification of organization identification schemes".

NOTE: See https://www.iso.org/standard/25773.html.

[i.44] IETF BCP 47.

NOTE: See https://tools.ietf.org/search/bcp47.

[i.45] ISO 17442:2019: "Financial services -- Legal entity identifier (LEI)".

NOTE: See https://www.iso.org/standard/75998.html.

[i.46] Website of DATEX II.

NOTE: Available at https://datex2.eu.

[i.47] ETSI GS CIM 006: "Context Information Management (CIM); Information Model (MOD0)".

NOTE: Available at: https://www.etsi.org/deliver/etsi_gs/CIM/001_099/006/01.01.01_60/gs_CIM006v010101p.pdf.

[i.48] IEC 61850: "Communication networks and systems for power utility automation".

NOTE: Available at: https://webstore.iec.ch/publication/6028.

[i.49] ISO 19107:2019: "Geographic information -- Spatial schema".

[i.50] ISO 19156:2011: "Geographic information -- Observations and measurements".

[i.51] IETF RFC 7946: "The GeoJSON Format".

[i.52] STARS4ALL European H2020 project.

3 Definition of terms, symbols and abbreviations

3.1 Terms

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

DATEX: European standard for the exchange of traffic information and traffic data [i.46]

3.2 Symbols

Void.

3.3 Abbreviations

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

AEF	Agricultural industry Electronics Foundation
API	Application Programming Interface
ASC	Ascending order
ASNE	American Society of Newspaper Editors
BCP	Best Current Practice
CIDOC	Centro Intercultural de Documentación
CIDOC-CRM	CIDOC Conceptual Reference Model
CIF	Código de Identificación Fiscal

NOTE: The Tax/Fiscal ID of the organization or person in Spain (CIF/NIF).

DCAT	Data Catalog Vocabulary (Version 1)
DCAT2	Data Catalog Vocabulary Version 2
DCAT-AP	DCAT Application profile
DE-9IM	Dimensionally Extended 9-Intersection Model
DTDL	Digital Twins Definition Language
ECGs	ElectroCardioGram(s)
EHAW	eHEALTH and Ageing-Well
FAO	Food and Agriculture Organization of the United Nations
FoaF	Friend of a Friend
GBFS	General Bikeshare Feed Specification
GeoJSON	Geographic JSON
GeoSPARQL	A Geographic Query Language for RDF Data
GIS	Geographic Information System
GLN	Global Location Number
GML	Geography Markup Language
GQL	Graph Query Language
GTFS	General Transit Feed Specification
HTTP	Hypertext Transfer Protocol
ICAR	International Committee for Animal Recording
ID	Identifier
IEEE	Institute of Electrical and Electronics Engineers
IRI	Internationalized Resource Identifier
IETF	Internet Engineering Task Force
IFC	Industry Foundation Classes
ILN	International Location Number
IoT	Internet of Things
ISIC	International Standard of Industrial Classification of All Economic Activities
ISO	International Organization for Standardization
ITS	Intelligent Transport Systems
IUDX	India Urban Data Exchange
JSON	JavaScript Object Notation

JSON-LD	JSON Linked Data
JTC	Joint Techncial Committee
NAICS	North American Industry Classification System
NGSI	Next Generation Service Interfaces
NGSI-LD	NGSI-Linked Data
NIF	Número de Identificación Fiscal

NOTE: The Tax/Fiscal ID of the organization or person in Spain (CIF/NIF).

NLP	Natural Language Processing
OASC	Open and Agile Smart Cities
OCF	Open Connectivity Foundation
OGC	Open Geospatial Consortium
OGC GeoSPARQL	OGC Geographic Query Language for RDF Data
OM	Ontology of units of Measure (OM ontology)
OWL	Web Ontology Language
OWL-DL	Web Ontology Language Description Logic
RAMI	Reference Architecture Model for Industry
RDBMS	Relational Database Management System
RDF	Resource Description Format
RDFS	Resource Description Framework Schema
RTDNA	Radio Television Digital News Association
SAREF	Smart Applications REFerence ontology
SAREF4CITY	SAREF for Smart Cities
SAREF4ENER	SAREF for Engergy
SAREF4WATR	SAREF for Water
SAREF4EHAW	SAREF for eHealth/Ageing-well
SAREF4BLDG	SAREF for Building
SAREF4INMA	SAREF for Industry and Manufacturing
SAREF4AGRI	SAREF for Agriculture and Food Chain
SAREF4AUTO	SAREF for Automotive
SAREF4WEAR	SAREF for Wearables
SAREF4LIFT	SAREF for Lift
SAREF4SYST	SAREF for System
SAREF4ENVI	SAREF for Environment
SC	Sub Committee
SEO	Search Engine Optimization
SKOS	Simple Knowledge Organization System
SPARQL	SPARQL Protocol and RDF Query Language
SQL	Structured Query Language
TC	Technical Committee
TCP/IP	Transaction Control Protocol/Internet Protocol
TD	Thing Description
TIN	Taxpayer Identification Number

NOTE: Tax/fiscal ID of organization or person in USA.

URI	Uniform Resource Identifier
URL	Universal Resource Locator
USA	United States of America
V2V	Vehicle(s) to Vehicle(s)
W3C®	World Wide Web Consrtium
W3C® SSN	W3C Semantic Sensor Network

NOTE: W3C SSN ontology includes W3C SOSA.

W3C® SOSA	W3C Sensor, Observation, Sample, and Actuator
WG	Working Group
WKT	Well-Known Text
WoT	Web of Things
WoT TD	Web of Things Thing Description
WSN	Wireless Specialty Networks

NOTE: See IEEE 802.15™ [i.35] Working Group WSN.

ZIP Zone Improvement Plan
XML Extensible Markup Language

4 Existing Relevant Semantic Models

4.1 Classification of Data Models

There are a lot of different technologies used to define and maintain data models (OWL, SKOS, RDFS, etc.). The way to map each type of data model in NGSI-LD may differ depending on the structure of the data model itself. In the present document, the following data models have been analysed:

- JSON-Schema:
  - SmartDataModels Program
- OWL:
  - SAREF family
  - GeoSPARQL Ontology
  - FoaF
- SKOS:
  - Agrovoc
- RDF:
  - DCAT
  - WoT

NOTE: Informative references to all these data models can be found in the following detailed clauses.

4.2 SAREF Family

4.2.1 SAREF Core

SAREF (ETSI TS 103 264 [i.1]) and its eleven derived extensions in different domains aims at facilitating the matching of existing assets (standards/protocols/data models/etc.) in smart applications. They are created and maintained by the ETSI TC SmartM2M. The starting point of SAREF is the concept of a device (e.g. a switch). The SAREF ontology offers lists of basic functions that can be eventually combined in order to have more complex functions in a single device. Each function has some associated commands, which can also be picked up as building blocks from a list. Depending on the functions it accomplishes, a device can be found in some corresponding states that are also listed as building blocks. When connected to a network, a device offers a service, which is the representation of a function which allows by other devices in the network to discover the function, register to it and control it remotely. A service can represent one or more functions and it is offered to any device that needs a certain set of functions. A device in the SAREF ontology is also characterized by a profile that can be used to optimize some property, such as Energy, in a home or office that is part of a building. Thus, the elements depicted by the ontology can be represented in NGSI-LD as Properties of Entities, and the connection between the different elements as Relationships. Additionally, SAREF core also describes the measurement element that relates the Device class to the SAREF Property and allows the storage of the values measured by the device.

Most relevant ones are depicted in next paragraphs including the ones for Smart Cities, Energy, Water management, and Health and ageing-well. The Smart Applications REFerence ontology (SAREF) is developed for interoperability between solutions from different providers and among various activity sectors for the Internet of Things (IoT) domain. In this clause, SAREF core and extension ontologies are described.

The core ontology defined within the SAREF core is shown in Figure 4.2.1-1.

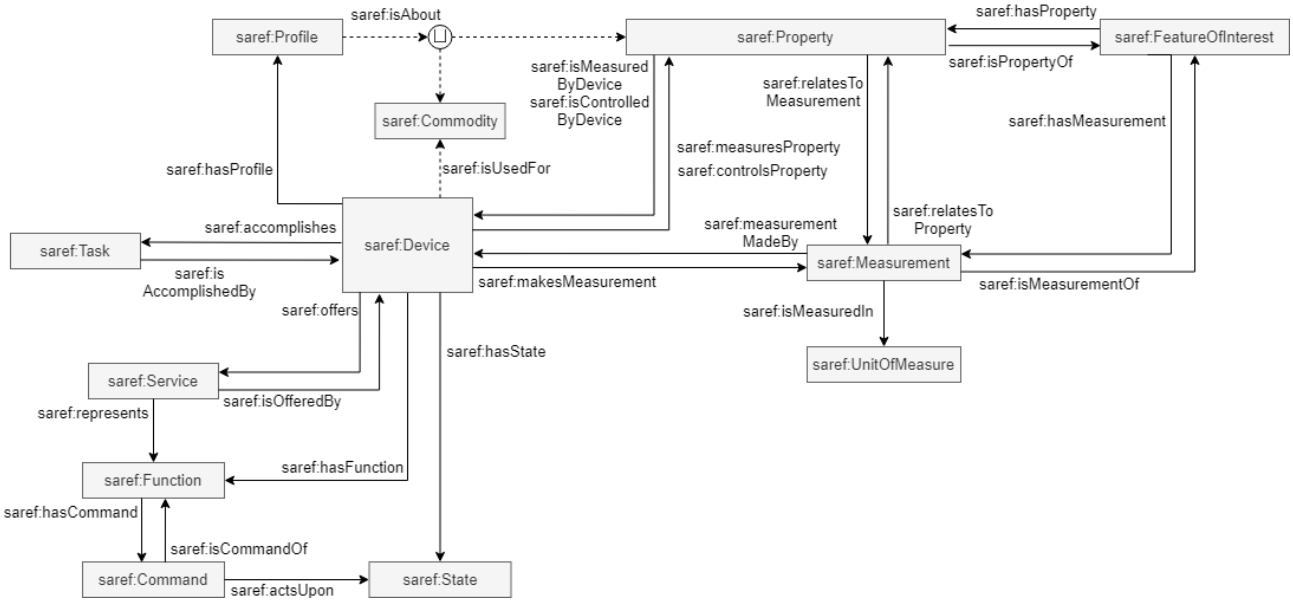

Figure 4.2.1-1: An overview of the SAREF core ontology

SAREF is designed as a domain-independent ontology for IoT applications. Thus, SAREF core ontology is designed as a set of simple core ontology patterns, including definitions for general fields. The SAREF core ontology is defined as device-centric including essential classes for IoT devices such as *saref:Device*, *saref:Function*, *saref:Task*, etc. This core ontology can be used with any real device, which is a tangible object designed to accomplish a particular task in households, common public buildings, or offices such as light switches, temperature sensors, and energy meters.

4.2.2 SAREF Extensions

4.2.2.1 Introduction

SAREF extensions are domain specific applications of the SAREF ontology identifying specific elements of the domains. Here is the list of extensions available in June 2022.

4.2.2.2 SAREF4CITY

SAREF4CITY (ETSI TS 103 410-4 [i.2]) focuses on extending SAREF in order to create a common core of general concepts for Smart City data-oriented IoT applications. SAREF4CITY has defined the classes for city management such as *s4city:AdministrativeArea*, *s4city:City*, and *s4city:Facility* that can be used for city stakeholders or various city services. SAREF4CITY is in its version 1.1.2.

4.2.2.3 SAREF4ENER

SAREF4ENER (ETSI TS 103 410-1 [i.3]) is the extension of SAREF for the energy domain that can be enabled to interconnect facilities (e.g. Italy-based and Germany-based industry associations). Classes and relationships included in SAREF4ENER can express all the powers and states of energy-related devices such as *s4ener:PowerSequence*, *s4ener:Energy*, and *s4ener:EnergyExpected*. SAREF4ENER is in its version 1.1.2.

4.2.2.4 SAREF4WATR

SAREF4WATR (ETSI TS 103 410-10 [i.4]) is an extension for the water domain which is based on a limited set of use cases and from available existing data models. SAREF4WATR has been developed in the context of the STF 566 which was established with the goal to create SAREF extensions. The water meter can be depicted as an example of extension uses. It can be described by a set of static classes and properties either reused from SAREF or from SAREF4WATR such as saref:Measurement and s4watr:WaterMeter. SAREF4WATR is in its version 1.1.1.

4.2.2.5 SAREF4EHAW

SAREF4EHAW (ETSI TS 103 410-8 [i.5]) extension has been specified and formalized by investigating EHAW (**eH**EALTH and **A**geing-**W**ell) domain related resources, as reported in ETSI TR 103 509 [i.6]. Therefore, SAREF4EHAW modular ontology has to allow the implementation of a limited set of typical EHAW related use cases, i.e.:

- Use case 1: "monitoring and support of healthy lifestyles for citizens".
- Use case 2: "Early Warning System (EWS) and Cardiovascular Accidents detection".

SAREF4EHAW mainly reuses the following existing ontologies: SAREF, SmartBAN, SAREF4ENVI, SAREF4WEAR and SAREF4health ontology which is a very first try to somehow extend SAREF ontology for the health vertical. SAREF4EHAW extension, specified and formalized as reported in ETSI TR 103 509 [i.6]. SAREF4EHAW is in its version 1.1.1.

4.2.2.6 SAREF4BLDG

SAREF4BLDG (ETSI TS 103 410-3 [i.7]), an extension of the SAREF ontology that was created based on the Industry Foundation Classes (IFC) standard for building information. It should be noted that not the whole standard has been transformed since it exceeds the scope of this extension, which is limited to devices and appliances within the building domain.

SAREF4BLDG is an OWL-DL ontology that extends SAREF with 72 classes (67 defined in SAREF4EBLDG and 5 reused from the SAREF and geo ontologies), 179 object properties (177 defined in SAREF4EBLDG and 2 reused from the SAREF and geo ontologies), and 83 data type properties (82 defined in SAREF4EBLDG and 1 reused from the SAREF ontology). SAREF4BLDG is in its version 1.1.2.

4.2.2.7 SAREF4INMA

SAREF4INMA (ETSI TS 103 410-5 [i.8]) is an extension of SAREF that was created for the industry and manufacturing domain. SAREF4INMA was created to be aligned with related initiatives in the smart industry and manufacturing domain in terms of modelling and standardization, such as the Reference Architecture Model for Industry 4.0 (RAMI), which combines several standards used by the various national initiatives in Europe that support digitalization in the manufacturing. These initiatives include but are not limited to, the platform Industrie 4.0 Germany, the Smart Industry initiative in the Netherlands, Industria 4.0 in Italy, the 'Industrie du future initiative' in France and more.

SAREF4INMA is an OWL-DL ontology that extends SAREF with 24 classes (in addition to a number of classes directly reused from the SAREF ontology and the SAREF4BLDG extension), 20 object properties (in addition to a number of object properties reused from the SAREF ontology and the SAREF4BLDG extension) and 11 data type properties.

SAREF4INMA solves the lack of interoperability between various types of production equipment to produce items in a factory and, once outside the factory, allows to uniquely track back the produced items to the corresponding production equipment, batches, material and precise time in which they were manufactured. SAREF4INMA is in its version 1.1.2.

4.2.2.8 SAREF4AGRI

SAREF4AGRI (ETSI TS 103 410-6 [i.9]) is an OWL-DL ontology that extends SAREF for the Smart Agriculture and Food Chain domain. The intention of SAREF4AGRI is to connect SAREF with existing ontologies (such as W3C SSN, W3C SOSA, GeoSPARQL, etc.) and important standardization initiatives and ontologies in the Smart Agriculture and Food Chain domain, including ICAR for livestock data (see https://www.icar.org/), AEF for agricultural equipment (see http://www.aef-online.org), Plant Ontology Consortium for plants (see http://archive.plantontology.org), AgGateway for IT support for arable farming (see http://www.aggateway.org/), as mentioned in the associated SAREF4AGRI requirements document ETSI TR 103 511 [i.38]. SAREF4AGRI is in its version 1.1.2.

4.2.2.9 SAREF4AUTO

SAREF4AUTO (ETSI TS 103 410-7 [i.10]) is an OWL-DL ontology that extends SAREF for the Automotive domain. The intention of SAREF4AUTO is to connect SAREF with existing ontologies (such as W3C SSN, W3C SOSA, GeoSPARQL, etc.) and important standardization initiatives and ontologies in the Automotive domain, as mentioned in the associated SAREF4AUTO requirements document ETSI TR 103 508 [i.39], including:

- ETSI TC ITS for V2V communications.
- The SENSORIS data model for exchanging data between vehicles and cloud services.
- The DATEX II standard for information exchange between traffic management centres, traffic information centres and service providers.
- The DATEX II Parking Publications for specifying information about parking sites and individual parking vehicles.

SAREF4AUTO is in its version 1.1.1.

4.2.2.10 SAREF4WEAR

SAREF4WEAR (ETSI TS 103 410-9 [i.11]) is an extension of SAREF for the Wearables domain. SAREF4WEAR has been specified and formalized by investigating related resources in the Wearables domain, as reported in ETSI TR 103 510 [i.40], such as:

- potential stakeholders
- standardization initiatives
- alliances/associations
- European projects
- EC directives
- existing ontologies
- data repositories

Therefore, SAREF4WEAR allows both:

- The implementation of a limited set of typical Wearable-related use cases already identified, i.e.:
  - Use case 1 "Healthcare"
  - Use case 2 "Open Air Public Events"
  - Use case 3 "Closed Environment Events"
- To fulfil the Wearable-related requirements provided in ETSI TR 103 510 [i.40], mainly the ontological ones that were mostly taken as input for the ontology specification

SAREF4WEAR is an OWL-DL ontology. For embedded semantic analytics purposes, SAREF4WEAR is designed using the modularity principle and can thus be mainly described by a set of knowledge modules. SAREF4WEAR is in its version 1.1.1.

4.2.2.11 SAREF4LIFT

SAREF4LIFT (ETSI TS 103 410-11 [i.12]) is an extension of SAREF for Smart Lift applications. It is based on a limited set of use cases and existing data models identified within available initiatives that was detailed in previous documents ETSI TR 103 546 [i.41] and ETSI TS 103 735 [i.42]. This extension specializes the SAREF4SYST ontology pattern for the Smart Lift domain.

The definition of SAREF4LIFT consisted of specifying and producing an extension of SAREF for the aforementioned domain based on the requirements resulting from previous initiatives. SAREF4LIFT is in its version 1.1.1.

4.2.2.12 SAREF4SYST

SAREF4SYST (ETSI TS 103 548 [i.13]) is a generic extension of ETSI TS 103 264 [i.1] SAREF that defines an ontology pattern which can be instantiated for different domains. SAREF4SYST defines s4syst:System, s4syst:Connection (connections between systems), and s4syst:ConnectionPoint (which highlights how systems are be connected). These core concepts can be used generically to define the topology of features of interest and can be specialized for multiple domains. The topology of features of interest is highly important in many use cases. For instance, if a room holds a lighting device, and if it is adjacent with an open window to a room whose luminosity is low, then by turning on the lighting device in the former room one may expect that the luminosity in the latter room will rise.

SAREF4SYST has two main aims:

- To extend SAREF with the capability or representing general topology of systems and how they are connected or interact.
- To exemplify how ontology patterns may help to ensure a homogeneous structure of the overall SAREF ontology and speed up the development of extensions.

SAREF4SYST consists both of a core ontology, and guidelines to create ontologies following the SAREF4SYST ontology pattern. The core ontology is a lightweight OWL-DL ontology that defines 3 classes and 9 object properties. SAREF4SYST is in its version 1.1.2.

4.2.2.13 SAREF4ENVI

SAREF4ENVI (ETSI TS 103 410-2 [i.14]) is an extension for the environmental domain. The extension was created in collaboration with domain experts in the field of light pollution currently working in the STARS4ALL European H2020 project [i.52] (see http://www.stars4all.eu/index.php/lpi/).

SAREF4ENVI has two main aims:

- To be the basis for enabling the use of SAREF in the environment domain.
- To exemplify how to enable interoperability between environmental devices in cooperation.

SAREF4ENVI is an OWL-DL ontology that extends SAREF with 32 classes (24 defined in SAREF4ENVI and 7 reused from the time, SAREF and geo ontologies), 24 object properties (22 defined in SAREF4ENVI and 2 reused from the SAREF and geo ontologies), 13 data type properties (9 defined in SAREF4ENVI and 4 reused from the SAREF ontology), and 24 individuals (9 defined in SAREF4ENVI and 12 reused from the OM ontology). SAREF4ENVI is in its version 1.1.2.

4.2.3 Common features

The SAREF extension share common elements. They all reference the SAREF core ontology using its concepts as super-classes for their domain-specific elements. Plus, extensions may reference to each other's if it is needed (e.g. SAREF4EHAW references the Wearable Device from SAREF4WEAR). All of them reference external data models (as GeoSPARQL) to reference those concepts that are not featured in any SAREF extension (e.g. representation of geospatial objects).

SAREF extensions share similar characteristic in their definitions. They are rather modular and based on shared elements to represent the devices and the measurements depending on the domain they are referencing.

They define functions as classes that implement a task to be performed. These functions have commands associated which allow the interaction between elements. They also implement the states to define the current situation of the any device/element.

Finally, both the core and the extensions have a dedicated @context file. Each file has two members:

- @graph, which contains all the triplets and definitions of every term
- @context, which contains the prefixes of every namespace (data model)

Although they are mostly used to represent the ontologies in JSON-LD documents, these files can be used to reference these ontologies in NGSI-LD as discussed in clause 5.

4.3 Schema.org

Schema.org [i.15] is an initiative founded by four very well known search engines for the creation of standardized semantic objects to allow the identification of such assets on the global web. It compiles more than 3 000 object types and specific extensions for eHealth and medical types, hotels, automotive and banks/financial institutions.

It is widely adopted in the web pages as long as these definitions are taken into account by the search engines. Consequently, their adoption is very relevant in other areas as they provide a JSON-LD export of the different elements.

The entities defined are extensively populated with attributes and each attribute is linked to their data types. It is very common that the same attribute can have different types. For example, inside the class *Organization* (shown in table 4.3-1) the address can be either a plain string, or a complex object including the sub-attributes of a Postal Address that includes more details (e.g. *addressCountry*, *addressLocality*, *AddressRegion*, *PostOfficeBoxNumber*, *postalcode* and *StreetAddress*). Main classes start with a capital letter, while attributes start with a small letter. All concepts are in camelCase. It is worth to mention that the number of different attributes (like in the example provided) is large and that these sub-attributes are deeply intertwined. For instance, the attribute *areaServed* includes *address* as a sub-attribute. Anyway, *address* can be represented as a *postalAddress* which has as a sub-attribute *areaServed*.

Table 4.3-1: Organization class defined in schema.org

actionableFeedbackPolicy	CreativeWork or URL	For a NewsMediaOrganization or other news-related Organization, a statement about public engagement activities (for news media, the newsroom's), including involving the public - digitally or otherwise - in coverage decisions, reporting and activities after publication.
address	PostalAddress or Text	Physical address of the item.
aggregateRating	AggregateRating	The overall rating, based on a collection of reviews or ratings, of the item.
alumni	Person	Alumni of an organization. Inverse property: alumniOf
areaServed	AdministrativeArea or GeoShape or Place or Text	The geographic area where a service or offered item is provided. Supersedes serviceArea.
award	Text	An award won by or for this item. Supersedes awards.
brand	Brand or Organization	The brand(s) associated with a product or service, or the brand(s) maintained by an organization or business person.
contactPoint	ContactPoint	A contact point for a person or organization. Supersedes contactPoints.
correctionsPolicy	CreativeWork or URL	For an Organization (e.g. NewsMediaOrganization), a statement describing (in news media, the newsroom's) disclosure and correction policy for errors.
department	Organization	A relationship between an organization and a department of that organization, also described as an organization (allowing different urls, logos, opening hours). For example: a store with a pharmacy, or a bakery with a cafe.
dissolutionDate	Date	The date that this organization was dissolved.
diversityPolicy	CreativeWork or URL	Statement on diversity policy by an Organization e.g. a NewsMediaOrganization. For a NewsMediaOrganization, a statement describing the newsroom's diversity policy on both staffing and sources, typically providing staffing data.
diversityStaffingReport	Article or URL	For an Organization (often but not necessarily a NewsMediaOrganization), a report on staffing diversity issues. In a news context this might be for example ASNE or RTDNA (USA) reports, or self-reported.
email	Text	Email address.
employee	Person	Someone working for this organization. Supersedes employees.
ethicsPolicy	CreativeWork or URL	Statement about ethics policy, e.g. of a NewsMediaOrganization regarding journalistic and publishing practices, or of a Restaurant, a page describing food source policies. In the case of a NewsMediaOrganization, an ethicsPolicy is typically a statement describing the personal, organizational, and corporate standards of behavior expected by the organization.

event	Event	Upcoming or past event associated with this place, organization, or action. Supersedes events.
faxNumber	Text	The fax number.
founder	Person	A person who founded this organization. Supersedes founders.
foundingDate	Date	The date that this organization was founded.
foundingLocation	Place	The place where the Organization was founded.
funder	Organization or Person	A person or organization that supports (sponsors) something through some kind of financial contribution.
funding	Grant	A Grant that directly or indirectly provide funding or sponsorship for this item. See also ownershipFundingInfo. Inverse property: fundedItem
globalLocationNumber	Text	The Global Location Number (GLN, sometimes also referred to as International Location Number or ILN) of the respective organization, person, or place. The GLN is a 13-digit number used to identify parties and physical locations.
hasCredential	EducationalOccupationalCredential	A credential awarded to the Person or Organization.
hasMerchantReturnPolicy	MerchantReturnPolicy	Specifies a MerchantReturnPolicy that may be applicable. Supersedes hasProductReturnPolicy.
hasOfferCatalog	OfferCatalog	Indicates an OfferCatalog listing for this Organization, Person, or Service.
hasPOS	Place	Points-of-Sales operated by the organization or person.
interactionStatistic	InteractionCounter	The number of interactions for the CreativeWork using the WebSite or SoftwareApplication. The most specific child type of InteractionCounter should be used. Supersedes interactionCount.
isicV4	Text	The International Standard of Industrial Classification of All Economic Activities (ISIC), Revision 4 code for a particular organization, business person, or place.
iso6523Code	Text	An organization identifier as defined in ISO 6523-1 [i.43]. Note that many existing organization identifiers such as leiCode or vatID can be expressed as an ISO 6523-1 [i.43] identifier by setting the ICD part of the ISO 6523-1 [i.43] identifier accordingly.
keywords	DefinedTerm or Text or URL	Keywords or tags used to describe some item. Multiple textual entries in a keywords list are typically delimited by commas, or by repeating the property.
knowsAbout	Text or Thing or URL	Of a Person, and less typically of an Organization, to indicate a topic that is known about - suggesting possible expertise but not implying it. It is not used to do not distinguish skill levels here, or relate it to educational content, events, objectives or JobPosting descriptions.
knowsLanguage	Language or Text	Of a Person, and less typically of an Organization, to indicate a known language. It is not used to do not distinguish skill levels or reading/writing/speaking/signing here. It ses language codes from the IETF BCP 47 [i.44] standard.
legalName	Text	The official name of the organization, e.g. the registered company name.
leiCode	Text	An organization identifier that uniquely identifies a legal entity as defined in ISO 17442 [i.45].
location	Place or PostalAddress or Text or VirtualLocation	The location of, for example, where an event is happening, where an organization is located, or where an action takes place.
logo	ImageObject or URL	An associated logo.
makesOffer	Offer	A pointer to products or services offered by the organization or person. Inverse property: offeredBy
member	Organization or Person	A member of an Organization or a ProgramMembership. Organizations can be members of organizations; ProgramMembership is typically for individuals. Supersedes musicGroupMember, members. Inverse property: memberOf
memberOf	Organization or ProgramMembership	An Organization (or ProgramMembership) to which this Person or Organization belongs. Inverse property: member
naics	Text	The North American Industry Classification System (NAICS) code for a particular organization or business person.

nonprofitStatus	NonprofitType	nonprofit Status indicates the legal status of a non-profit organization in its primary place of business.
numberOfEmployees	QuantitativeValue	The number of employees in an organization e.g. business.
ownershipFundingInfo	AboutPage or CreativeWork or Text or URL	For an Organization (often but not necessarily a NewsMediaOrganization), a description of organizational ownership structure; funding and grants. In a news/media setting, this is with particular reference to editorial independence. Note that the funder is also available and can be used to make basic funder information machine-readable.
owns	OwnershipInfo or Product	Products owned by the organization or person.
parentOrganization	Organization	The larger organization that this organization is a subOrganization of, if any. Supersedes branchOf. Inverse property: subOrganization
publishingPrinciples	CreativeWork or URL	The publishingPrinciples property indicates (typically via URL) a document describing the editorial principles of an Organization (or individual e.g. a Person writing a blog) that relate to their activities as a publisher, e.g. ethics or diversity policies. When applied to a CreativeWork (e.g. NewsArticle) the principles are those of the party primarily responsible for the creation of the CreativeWork. While such policies are most typically expressed in natural language, sometimes related information (e.g. indicating a funder) can be expressed using schema.org terminology.
review	Review	A review of the item. Supersedes reviews.
seeks	Demand	A pointer to products or services sought by the organization or person (demand).
slogan	Text	A slogan or motto associated with the item.
sponsor	Organization or Person	A person or organization that supports a thing through a pledge, promise, or financial contribution. e.g. a sponsor of a Medical Study or a corporate sponsor of an event.
subOrganization	Organization	A relationship between two organizations where the first includes the second, e.g. as a subsidiary. See also: the more specific 'department' property. Inverse property: parentOrganization
taxID	Text	The Tax/Fiscal ID of the organization or person, e.g. the TIN in the USA or the CIF/NIF in Spain.
telephone	Text	The telephone number.
unnamedSourcesPolicy	CreativeWork or URL	For an Organization (typically a NewsMediaOrganization), a statement about policy on use of unnamed sources and the decision process that is needed.
vatID	Text	The Value-added Tax ID of the organization or person.
+ Properties from Thing		

Additionally, the class *Organization* inherits the attributes of the class *Thing*, which includes a list of 12 additional attributes.

A defined context is located at *https://schema.org/docs/jsonldcontext.json* containing all the 2 844 terms. For each element its corresponding long IRI (in the format "*https://schema.org/element*") is listed.

Mapping schema.org in NGSI-LD is quite straightforward. The classes can be mapped as NGSI-LD Entities and the attributes (and sub-attributes) can mapped as NGSI-LD Attributes and/or NGSI-LD sub-Attributes.

4.4 Thesauri

4.4.1 General Overview

Thesauri are domain-based controlled vocabularies generally used to find synonyms or antonyms. They may contain definitions as well and express the difference of meaning of every term giving more insights about their usage. They are generally structured in some way: some group words in a hierarchical taxonomy of concepts, others are organized alphabetically or in some other way.

4.4.2 SKOS ontology

The Simple Knowledge Organization System ontology (SKOS, [i.16]) is a common data model for sharing and linking knowledge organization systems via the Semantic Web. SKOS ontology offers a consistent vocabulary for defining and linking different concepts. Several domain-based thesauri are proposed and maintained using SKOS.

4.4.3 Agrovoc

Agrovoc [i.17] is a relevant Linked Open Data set based on SKOS ontology about agriculture but also covers all areas of interest to FAO, such as food, nutrition, forestry, fisheries, names of animals and plants, environment, biological notions, and techniques of plant cultivation, etc. Agrovoc concept model consists of concepts, terms, and relationships. The concept is the set of terms used in any language to describe the same idea identified by dereferenceable URIs. Agrovoc supports more than 40 000 concepts, and 937 000 terms and Agrovoc thesaurus content in 41 languages is now released under the international license CC BY IGO 3.0. (see https://creativecommons.org/licenses/by/3.0/igo/).

4.5 SmartDataModels program

4.5.1 General Overview

The SmartDataModels program [i.18] is a collaborative program lead by four organizations (March 2022), FIWARE Foundation, IUDX, OASC and TMForum to provide open licensed data models based on actual experiences and open and adopted standards. Table 4.5.1-1 contains all the 12 business domains and 55 subjects (groups of data models).

Table 4.5.1-1: The 12 business domains featured in the SmartDataModels Program

Smart Cities	Smart Agrifood	Smart Water	Smart Energy
Smart Environment	Smart Robotics	Smart Sensoring	Cross sector
Smart Aeronautics	Smart Destination	Smart Health	Smart Manufacturing

These data models are open-licensed: it is not only allowed to use them in third-party applications, but it is also possible for external developers to modify any model and share their modifications with the maintainers.

Every domain is structured in its own GITHUB repository. The technical specifications are available in the "docs" directory in each data model. They are intended to be used with compliance with FIWARE NGSI version 2 and NGSI-LD. They are also available for any other platform (e.g. DTDL initiative).

The program allows for contribution and creation of additional data models, to improve the existing ones as long as new features/modifications can be used in actual use cases or in open and adopted standards (e.g. GTFS, GBFS, IEC 61850 [i.48], OCF Open Connectivity Foundation, DCAT-AP 2.0.1, etc.). The added value is that every data model provides guidance in many aspects, such as the data type of a term/concept (which is not always provided by regulations and standards). A document with the guidelines for contributions (called the "contribution manual") and dedicated tools are public to simplify the contribution process for external developers. More than 70 organizations have already contributed mapping 18 000 terms, with over 100 collaborators in the 800 data models available.

As an example, the following is a payload that may be returned by one of the services that have been created within the SmartDataModels program:

```
{
    "id": "urn:ngsi-ld:AirQualityObserved:RZ:Obsv4567",
    "type": "AirQualityObserved",
    "dateObserved": {
        "type": "Property",
        "value": {
            "@type": "DateTime",
            "@value": "2018-08-07T12:00:00Z"
        }
    },
    "NO2": {
        "type": "Property",
        "value": 22,
        "unitCode": "GP",
        "accuracy": {
            "type": "Property",
            "value": 0.95
        }
    },
    "refPointOfInterest": {
        "type": "Relationship",
        "object": "urn:ngsi-ld:PointOfInterest:RZ:MainSquare"
    },
    "@context": [
       "SAREF core",
        "saref4Env",
        "https://smart-data-models/dataModel.Environment/context.jsonld",
    ]
}
```

4.5.2 Suggested methods to link external data

Every subject (group of data models) of the SmartDataModels program includes local IRIs that can be used to reference linked data resources, but the actual goal is to combine and re-use existing ontologies and vocabularies. There are 18 ontologies/vocabularies available (including SAREF core and all its extensions, IUDX, GSMA, schema.org, etc.), but this number may increase in the future (depending on contributions).

When working with the SmartDataModels services, it is possible to define a configuration file which allows to choose which ontologies would be mapped. The order matters, hence, the first elements have a higher priority than the subsequential ones when mapping the elements. An example of a configuration file is shown below.

```
{
  "ngsi-ld": "https://raw.githubusercontent.com/smart-data-models/data-
models/master/context/ontologies_files/ngsi-ld-core-context.jsonld#1.4",
  "saref": "https://raw.githubusercontent.com/smart-data-models/data-
models/master/context/ontologies_files/saref.json#3.1.1"
  "schema.org": "https://raw.githubusercontent.com/smart-data-models/data-
models/master/context/ontologies_files/schema.org.json#1.0",
  "iudx": "https://raw.githubusercontent.com/smart-data-models/data-
models/master/context/ontologies_files/iudx.json#1.0"
}
```

As said, a configuration file sets the precedence of the mapping of the different terms (short IRIs or prefixes) of the attributes. Following the order, the service will look for the long IRIs of every term starting by the first option ("ngsi-ld" in the example). If any term has not been found, it will look for its mapping in the second option. This operation will continue until all the terms have been referenced or if there are not more options in the configuration file. If at the end of the process any term has not been referenced, then the default IRI from SmartDataModels is applied (similar to what happens inside the Context Brokers with the core context of NGSI-LD).

Using this configuration, a local context, defined with the SmartDataModels IRI, can be mapped with external vocabularies. Below it is shown an example of a local context merging the subjects Battery, building, device and Weather based on the SmartDataModels IRI.

```
{
  "@context": {
    "acPowerInput": "https://smartdatamodels.org/dataModel.Battery/acPowerInput",
    "acPowerOutput": "https://smartdatamodels.org/dataModel.Battery/acPowerOutput",
    "activePower": "https://smartdatamodels.org/dataModel.Battery/activePower",
    "address": "https://smartdatamodels.org/address",
    "alternateName": "https://smartdatamodels.org/alternateName",
    "application": "https://smartdatamodels.org/dataModel.Battery/application",
    "areaServed": "https://smartdatamodels.org/areaServed",
    "autonomyTime": "https://smartdatamodels.org/dataModel.Battery/autonomyTime",
    "averageLife": "https://smartdatamodels.org/dataModel.Battery/averageLife",
    "batteryAssessmentMethods": "https://smartdatamodels.org/dataModel.Battery/batteryAssessmentMethods",
    "batteryStatus": "https://smartdatamodels.org/dataModel.Battery/batteryStatus",
    "batteryType": "https://smartdatamodels.org/dataModel.Battery/batteryType",
    "capacityCnnn": "https://smartdatamodels.org/dataModel.Battery/capacityCnnn",
    "chargeDischargeReactivity": "https://smartdatamodels.org/dataModel.Battery/chargeDischargeReactivity",
    "chargeEfficiency": "https://smartdatamodels.org/dataModel.Battery/chargeEfficiency",
    "chargePower": "https://smartdatamodels.org/dataModel.Battery/chargePower",
    "chargingModeAllowed": "https://smartdatamodels.org/dataModel.Battery/chargingModeAllowed",
    "communication": "https://smartdatamodels.org/dataModel.Battery/communication",
    "current": "https://smartdatamodels.org/dataModel.Battery/current",
    "cycleLife": "https://smartdatamodels.org/dataModel.Battery/cycleLife",
    "dataProvider": "https://smartdatamodels.org/dataProvider",
    "dateCreated": "https://smartdatamodels.org/dateCreated",
    "dateEnergyMeteringStarted": "https://smartdatamodels.org/dataModel.Battery/dateEnergyMeteringStarted",
    "dateLastReported": "https://smartdatamodels.org/dataModel.Battery/dateLastReported",
    "dateModified": "https://smartdatamodels.org/dateModified",
    "dateObservedFrom": "https://smartdatamodels.org/dataModel.Battery/dateObservedFrom",
    "dateObservedTo": "https://smartdatamodels.org/dataModel.Battery/dateObservedTo",
    "dcPowerInput": "https://smartdatamodels.org/dataModel.Battery/dcPowerInput",
    "dcPowerOutput": "https://smartdatamodels.org/dataModel.Battery/dcPowerOutput",
    "deepOfDischarge": "https://smartdatamodels.org/dataModel.Battery/deepOfDischarge",
    "description": "http://purl.org/dc/terms/description",
    "dimension": "https://smartdatamodels.org/dataModel.Battery/dimension",
    "dischargeEfficiency": "https://smartdatamodels.org/dataModel.Battery/dischargeEfficiency",
    "dischargePower": "https://smartdatamodels.org/dataModel.Battery/dischargePower",
    "durationPeakPower": "https://smartdatamodels.org/dataModel.Battery/durationPeakPower",
    "id": "@id",
    "installationCondition": "https://smartdatamodels.org/dataModel.Battery/installationCondition",
    "installationMode": "https://smartdatamodels.org/dataModel.Battery/installationMode",
    "inverterStatus": "https://smartdatamodels.org/dataModel.Battery/inverterStatus",
    "lifeCycleNumber": "https://smartdatamodels.org/dataModel.Battery/lifeCycleNumber",
    "location": "ngsi-ld:location",
    "massEnergyDensity": "https://smartdatamodels.org/dataModel.Battery/massEnergyDensity",
    "maxOutputPower": "https://smartdatamodels.org/dataModel.Battery/maxOutputPower",
    "maximumVoltageEOC": "https://smartdatamodels.org/dataModel.Battery/maximumVoltageEOC",
    "minimumVoltageEOD": "https://smartdatamodels.org/dataModel.Battery/minimumVoltageEOD",
    "name": "https://smartdatamodels.org/name",
    "ngsi-ld": "https://uri.etsi.org/ngsi-ld/",
    "nominalAmpere": "https://smartdatamodels.org/dataModel.Battery/nominalAmpere",
    "nominalCapacity": "https://smartdatamodels.org/dataModel.Battery/nominalCapacity",
    "nominalFrequency": "https://smartdatamodels.org/dataModel.Battery/nominalFrequency",
    "nominalVoltage": "https://smartdatamodels.org/dataModel.Battery/nominalVoltage",
    "openCircuitVoltage": "https://smartdatamodels.org/dataModel.Battery/openCircuitVoltage",
    "operatingAltitude": "https://smartdatamodels.org/dataModel.Battery/operatingAltitude",
    "operatingAmpere": "https://smartdatamodels.org/dataModel.Battery/operatingAmpere",
    "operatingFrequency": "https://smartdatamodels.org/dataModel.Battery/operatingFrequency",
    "operatingTemperature": "https://smartdatamodels.org/dataModel.Battery/operatingTemperature",
    "operatingVoltage": "https://smartdatamodels.org/dataModel.Battery/operatingVoltage",
    "overloadAccepted": "https://smartdatamodels.org/dataModel.Battery/overloadAccepted",
    "overloadAcceptedTime": "https://smartdatamodels.org/dataModel.Battery/overloadAcceptedTime",
    "owner": "https://smartdatamodels.org/owner",
    "peakPower": "https://smartdatamodels.org/dataModel.Battery/peakPower",
    "possibilityOfUse": "https://smartdatamodels.org/dataModel.Battery/possibilityOfUse",
    "protectionIK": "https://smartdatamodels.org/dataModel.Battery/protectionIK",
    "protectionIP": "https://smartdatamodels.org/dataModel.Battery/protectionIP",
    "reactivePower": "https://smartdatamodels.org/dataModel.Battery/reactivePower",
    "rechargeEnergySource": "https://smartdatamodels.org/dataModel.Battery/rechargeEnergySource",
    "rechargeTime": "https://smartdatamodels.org/dataModel.Battery/rechargeTime",
    "refStorageBatteryDevice": "https://smartdatamodels.org/dataModel.Battery/refStorageBatteryDevice",
    "roundTripEfficiency": "https://smartdatamodels.org/dataModel.Battery/roundTripEfficiency",
    "seeAlso": "https://smartdatamodels.org/seeAlso",
    "selfDischargeRate": "https://smartdatamodels.org/dataModel.Battery/selfDischargeRate",
    "source": "https://smartdatamodels.org/source",
    "stateOfCharge": "https://smartdatamodels.org/dataModel.Battery/stateOfCharge",
    "stateOfHealth": "https://smartdatamodels.org/dataModel.Battery/stateOfHealth",
    "status": "ngsi-ld:status",
    "storableEnergy": "https://smartdatamodels.org/dataModel.Battery/storableEnergy",
    "toolBMS": "https://smartdatamodels.org/dataModel.Battery/toolBMS",
    "type": "@type",
    "typeEnergySource": "https://smartdatamodels.org/dataModel.Battery/typeEnergySource",
    "typeOfUse": "https://smartdatamodels.org/dataModel.Battery/typeOfUse",
    "usableEnergy": "https://smartdatamodels.org/dataModel.Battery/usableEnergy",
    "volEnergyDensity": "https://smartdatamodels.org/dataModel.Battery/volEnergyDensity",
    "weight": "https://smartdatamodels.org/dataModel.Battery/weight",
    "accelerationMeasured": "https://smartdatamodels.org/dataModel.Building/accelerationMeasured",
    "buildingTypeChildren": "https://smartdatamodels.org/dataModel.Building/buildingTypeChildren",
    "buildingTypeParent": "https://smartdatamodels.org/dataModel.Building/buildingTypeParent",
    "collapseRisk": "https://smartdatamodels.org/dataModel.Building/collapseRisk",
```

```
    "containedInPlace": "https://smartdatamodels.org/dataModel.Building/containedInPlace",
    "dateFinished": "https://smartdatamodels.org/dataModel.Building/dateFinished",
    "dateObserved": "https://smartdatamodels.org/dateObserved",
    "dateStarted": "https://smartdatamodels.org/dataModel.Building/dateStarted",
    "endDate": "https://smartdatamodels.org/dataModel.Building/endDate",
    "floorsAboveGround": "https://smartdatamodels.org/dataModel.Building/floorsAboveGround",
    "floorsBelowGround": "https://smartdatamodels.org/dataModel.Building/floorsBelowGround",
    "occupier": "https://smartdatamodels.org/dataModel.Building/occupier",
    "openingHours": "https://smartdatamodels.org/dataModel.Building/openingHours",
    "operationSequence": "https://smartdatamodels.org/dataModel.Building/operationSequence",
    "peopleCapacity": "https://smartdatamodels.org/dataModel.Building/peopleCapacity",
    "peopleOccupancy": "https://smartdatamodels.org/dataModel.Building/peopleOccupancy",
    "refBuilding": "https://smartdatamodels.org/dataModel.Building/refBuilding",
    "refMap": "https://smartdatamodels.org/dataModel.Building/refMap",
    "refOperator": "https://smartdatamodels.org/dataModel.Building/refOperator",
    "refRelatedBuildingOperation": "https://smartdatamodels.org/dataModel.Building/refRelatedBuildingOperation",
    "refRelatedDeviceOperation": "https://smartdatamodels.org/dataModel.Building/refRelatedDeviceOperation",
    "root": "https://smartdatamodels.org/dataModel.Building/root",
    "startDate": "https://smartdatamodels.org/dataModel.Building/startDate",
    "addressedAt": "https://smartdatamodels.org/dataModel.Device/addressedAt",
    "annotations": "https://smartdatamodels.org/annotations",
    "cameraName": "https://smartdatamodels.org/dataModel.Device/cameraName",
    "cameraNum": "https://smartdatamodels.org/dataModel.Device/cameraNum",
    "cameraOrientation": "https://smartdatamodels.org/dataModel.Device/cameraOrientation",
    "cameraType": "https://smartdatamodels.org/dataModel.Device/cameraType",
    "cameraUsage": "https://smartdatamodels.org/dataModel.Device/cameraUsage",
    "color": "https://smartdatamodels.org/color",
    "configuration": "https://smartdatamodels.org/dataModel.Device/configuration",
    "controlledAsset": "https://smartdatamodels.org/dataModel.Device/controlledAsset",
    "controlledProperty": "https://smartdatamodels.org/dataModel.Device/controlledProperty",
    "crossborderTransfer": "https://smartdatamodels.org/dataModel.Device/crossborderTransfer",
    "dateFirstUsed": "https://smartdatamodels.org/dataModel.Device/dateFirstUsed",
    "dateInstalled": "https://smartdatamodels.org/dataModel.Device/dateInstalled",
    "dateLastCalibration": "https://smartdatamodels.org/dataModel.Device/dateLastCalibration",
    "dateLastValueReported": "https://smartdatamodels.org/dataModel.Device/dateLastValueReported",
    "dateManufactured": "https://smartdatamodels.org/dataModel.Device/dateManufactured",
    "depth": "https://smartdatamodels.org/dataModel.Device/depth",
    "device": "https://smartdatamodels.org/dataModel.Device/device",
    "deviceClass": "https://smartdatamodels.org/dataModel.Device/deviceClass",
    "deviceState": "https://smartdatamodels.org/dataModel.Device/deviceState",
    "deviceType": "https://smartdatamodels.org/dataModel.Device/deviceType",
    "direction": "https://smartdatamodels.org/dataModel.Device/direction",
    "distance": "https://smartdatamodels.org/dataModel.Device/distance",
    "dstAware": "https://smartdatamodels.org/dataModel.Device/dstAware",
    "endDateTime": "https://smartdatamodels.org/dataModel.Device/endDateTime",
    "endedAt": "https://smartdatamodels.org/dataModel.Device/endedAt",
    "energyLimitationClass": "https://smartdatamodels.org/dataModel.Device/energyLimitationClass",
    "entityVersion": "https://smartdatamodels.org/dataModel.Device/entityVersion",
    "firmwareVersion": "https://smartdatamodels.org/dataModel.Device/firmwareVersion",
    "floor": "https://smartdatamodels.org/dataModel.Device/floor",
    "function": "https://smartdatamodels.org/dataModel.Device/function",
    "hardwareVersion": "https://smartdatamodels.org/dataModel.Device/hardwareVersion",
    "image": "https://smartdatamodels.org/image",
    "imageSnapshot": "https://smartdatamodels.org/dataModel.Device/imageSnapshot",
    "ipAddress": "https://smartdatamodels.org/dataModel.Device/ipAddress",
    "isIndoor": "https://smartdatamodels.org/dataModel.Device/isIndoor",
    "isPersonalData": "https://smartdatamodels.org/dataModel.Device/isPersonalData",
    "legitimateInterest": "https://smartdatamodels.org/dataModel.Device/legitimateInterest",
    "macAddress": "https://smartdatamodels.org/dataModel.Device/macAddress",
    "mcc": "https://smartdatamodels.org/dataModel.Device/mcc",
    "measurementType": "https://smartdatamodels.org/dataModel.Device/measurementType",
    "mediaURL": "https://smartdatamodels.org/dataModel.Device/mediaURL",
    "meterType": "https://smartdatamodels.org/dataModel.Device/meterType",
    "mnc": "https://smartdatamodels.org/dataModel.Device/mnc",
    "numValue": "https://smartdatamodels.org/dataModel.Device/numValue",
    "offPeakConsumption": "https://smartdatamodels.org/dataModel.Device/offPeakConsumption",
    "operator": "https://smartdatamodels.org/dataModel.Device/operator",
    "osVersion": "https://smartdatamodels.org/dataModel.Device/osVersion",
    "outlier": "https://smartdatamodels.org/dataModel.Device/outlier",
    "peakConsumption": "https://smartdatamodels.org/dataModel.Device/peakConsumption",
    "plannedEndAt": "https://smartdatamodels.org/dataModel.Device/plannedEndAt",
    "plannedStartAt": "https://smartdatamodels.org/dataModel.Device/plannedStartAt",
    "powerFactor": "https://smartdatamodels.org/dataModel.Device/powerFactor",
    "provider": "https://smartdatamodels.org/dataModel.Device/provider",
    "purpose": "https://smartdatamodels.org/dataModel.Device/purpose",
    "recipientList": "https://smartdatamodels.org/dataModel.Device/recipientList",
    "refDeviceModel": "https://smartdatamodels.org/dataModel.Device/refDeviceModel",
    "relativePosition": "https://smartdatamodels.org/dataModel.Device/relativePosition",
    "reportedAt": "https://smartdatamodels.org/dataModel.Device/reportedAt",
    "retentionPeriod": "https://smartdatamodels.org/dataModel.Device/retentionPeriod",
    "rssi": "https://smartdatamodels.org/dataModel.Device/rssi",
    "softwareVersion": "https://smartdatamodels.org/dataModel.Device/softwareVersion",
    "startDateTime": "https://smartdatamodels.org/dataModel.Device/startDateTime",
    "startedAt": "https://smartdatamodels.org/dataModel.Device/startedAt",
    "streamName": "https://smartdatamodels.org/dataModel.Device/streamName",
    "streamURL": "https://smartdatamodels.org/dataModel.Device/streamURL",
    "supportedProtocol": "https://smartdatamodels.org/dataModel.Device/supportedProtocol",
    "supportedUnits": "https://smartdatamodels.org/dataModel.Device/supportedUnits",
    "textValue": "https://smartdatamodels.org/dataModel.Device/textValue",
```

```
    "totalConsumption": "https://smartdatamodels.org/dataModel.Device/totalConsumption",
    "unit": "https://smartdatamodels.org/dataModel.Device/unit",
    "user": "https://smartdatamodels.org/dataModel.Device/user",
    "value": "ngsi-ld:hasValue",
    "airQualityIndex": "https://smartdatamodels.org/dataModel.Weather/airQualityIndex",
    "airQualityIndexForecast": "https://smartdatamodels.org/dataModel.Weather/airQualityIndexForecast",
    "airTemperatureForecast": "https://smartdatamodels.org/dataModel.Weather/airTemperatureForecast",
    "airTemperatureTSA": "https://smartdatamodels.org/dataModel.Weather/airTemperatureTSA",
    "alertSource": "https://smartdatamodels.org/dataModel.Weather/alertSource",
    "aqiMajorPollutant": "https://smartdatamodels.org/dataModel.Weather/aqiMajorPollutant",
    "aqiMajorPollutantForecast": "https://smartdatamodels.org/dataModel.Weather/aqiMajorPollutantForecast",
    "atmosphericPressure": "https://smartdatamodels.org/dataModel.Weather/atmosphericPressure",
    "data": "ngsi-ld:data",
    "dateIssued": "https://smartdatamodels.org/dataModel.Weather/dateIssued",
    "dateRetrieved": "https://smartdatamodels.org/dataModel.Weather/dateRetrieved",
    "dayMaximum": "https://smartdatamodels.org/dataModel.Weather/dayMaximum",
    "dayMinimum": "https://smartdatamodels.org/dataModel.Weather/dayMinimum",
    "dewPoint": "https://smartdatamodels.org/dataModel.Weather/dewPoint",
    "feelLikesTemperature": "https://smartdatamodels.org/dataModel.Weather/feelLikesTemperature",
    "feelsLikesTemperature": "https://smartdatamodels.org/dataModel.Weather/feelsLikesTemperature",
    "gustSpeed": "https://smartdatamodels.org/dataModel.Weather/gustSpeed",
    "illuminance": "https://smartdatamodels.org/dataModel.Weather/illuminance",
    "pH": "https://smartdatamodels.org/dataModel.Weather/pH",
    "precipitation": "https://smartdatamodels.org/dataModel.Weather/precipitation",
    "precipitationForecast": "https://smartdatamodels.org/dataModel.Weather/precipitationForecast",
    "pressureTendency": "https://smartdatamodels.org/dataModel.Weather/pressureTendency",
    "relativeHumidity": "https://smartdatamodels.org/dataModel.Weather/relativeHumidity",
    "relativeHumidityForecast": "https://smartdatamodels.org/dataModel.Weather/relativeHumidityForecast",
    "salinity": "https://smartdatamodels.org/dataModel.Weather/salinity",
    "severity": "https://smartdatamodels.org/dataModel.Weather/severity",
    "snowHeight": "https://smartdatamodels.org/dataModel.Weather/snowHeight",
    "solarRadiation": "https://smartdatamodels.org/dataModel.Weather/solarRadiation",
    "streamGauge": "https://smartdatamodels.org/dataModel.Weather/streamGauge",
    "subCategory": "https://smartdatamodels.org/dataModel.Weather/subCategory",
    "surfaceTemperature": "https://smartdatamodels.org/dataModel.Weather/surfaceTemperature",
    "uVIndexMax": "https://smartdatamodels.org/dataModel.Weather/uVIndexMax",
    "validFrom": "https://smartdatamodels.org/dataModel.Weather/validFrom",
    "validTo": "https://smartdatamodels.org/dataModel.Weather/validTo",
    "validity": "https://smartdatamodels.org/dataModel.Weather/validity",
    "visibility": "https://smartdatamodels.org/dataModel.Weather/visibility",
    "waveHeight": "https://smartdatamodels.org/dataModel.Weather/waveHeight",
    "waveLevel": "https://smartdatamodels.org/dataModel.Weather/waveLevel",
    "wavePeriod": "https://smartdatamodels.org/dataModel.Weather/wavePeriod",
    "weatherType": "https://smartdatamodels.org/dataModel.Weather/weatherType",
    "windDirection": "https://smartdatamodels.org/dataModel.Weather/windDirection",
    "windSpeed": "https://smartdatamodels.org/dataModel.Weather/windSpeed"
  }
}
```


The resulting context is a mapping of every short term with their external reference, according to the configuration file, preserving the original IRI for those elements not found in the corresponding vocabularies.

4.6 OGC GeoSPARQL

4.6.1 General Overview

OGC (previously known as OpenGis Consortium) is a non-profit organization which provides open standards for geospatial content and services. Although many current and past work within the OGC involves semantic and graph technologies, there is one main applicable OGC standard: GeoSPARQL. GeoSPARQL offers semantics for representing geospatial data and it specifies how geospatial data, expressed using the Resource Description Framework (RDF), can be queried using SPARQL, the query language for RDF.

References to GeoSPARQL in other standards, such as DCAT2 [i.19], CIDOC-CRM [i.20] and SAREF extensions (e.g. SAREF4CITY) suggests that, if not popular, it is a well-known data model for geospatial concepts.

The GeoSPARQL standard comprises multiple parts. The main ones are the following:

- Domain model RDF/OWL2 ontology
- Functions & Rule SKOS taxonomy
- Simple Features vocabulary
- SPARQL extension functions

OGC has published GeoSPARQL in 2012 as v0.1 [i.21] and it is currently updating it by drafting the latest v1.1 [i.22], which brings new changes as the support to GeoJSON literals and the addition of new concepts inside the ontology.

4.6.2 The ontology

The ontology defined within the GeoSPARQL standard v1.1 is shown in Figure 4.6.2-1.

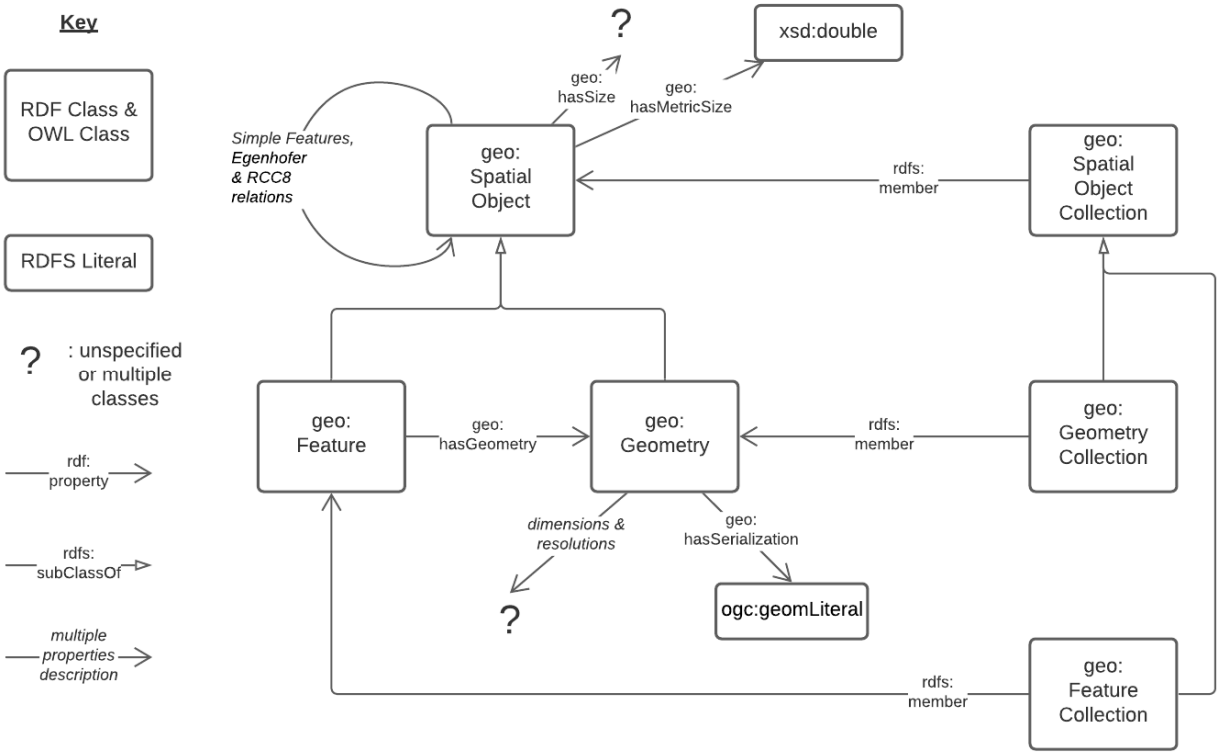

Figure 4.6.2-1: An overview of the GeoSPARQL ontology

The main concept is the *SpatialObject* which represents everything that can have a spatial representation and it is the superclass of *Geometry* and *Feature*. A *Geometry* is equivalent to GM_Object (defined in ISO 19107 [i.49]) and represents the top-level geometry type. It is the superclass of all the geometry types (such as Point, Line, Polygon, etc.). Any geometry is represented as a serialization of type *geomLiteral* (such as GeoJSON, WKT, GML, etc.). A *Feature* is equivalent to GFI_Feature (defined in ISO 19156 [i.50]) and represents a top-level feature type (such as a building, a statue, etc.). A *Feature* may be described by a *Geometry*. For instance, a *Feature* may have a bounding box (described as a Polygon) or a geospatial location (described as a Point).

SpatialObjects can have properties that add metadata or additional information (e.g. size, length, etc.). *SpatialObjects* are linked to each other with special relations named *Topological Relations*. They have been described using the Dimensionally Extended 9-Intersection Model (DE-9IM) pattern.

GeoSPARQL v1.1 defines a new class which is the *SpatialObjectCollection* which represent any collection of individual *SpatialObjects*, and it is the superclass of *FeatureCollection* and *GeometryCollection*. A *FeatureCollection* is a collection of individual Features elements. A *GeometryCollection* is a collection of individual Geometries.

When compared with other data models in the present document, the GeoSPARQL ontology is the simplest, but this is not a limitation. The limited number of classes does not reduce the expressivity of this ontology which excels in describing geospatial objects and properties. Furthermore, the GeoSPARQL ontology can be easily referenced in other data models since it is domain-independent: a *Feature* may be anything from a Building to a Statue or even a Car. For instance, as mentioned in clause 4.6.1, SAREF4CITY references GeoSPARQL by stating that one of the super-classes of a City is *Feature* from GeoSPARQL.

NGSI-LD contains references for *Geometries* (cross-domain ontology) and *Features* (core-context). They are not the ones defined in ISO 19107 [i.49] and ISO 19156 [i.50] (as the ones described in GeoSPARQL), but they have been formalized inside the GeoJSON specification (IETF RFC 7946 [i.51]). Anyway, it may be safe to say that these concepts are equivalent. Other authorities/organizations seem to agree with this fact. For instance, the Inspire Directive has created a mapping between GeoJSON and GeoSPARQL [i.23] explaining how to represent GeoSPARQL *Geometries* and *Features* using GeoJSON. Any GeoSPARQL *Geometry* can be represented with a GeoJSON Geometry since they are equivalent classes; in the same way, any "*FeatureType*" concept (as the *Feature* class in GeoSPARQL) can be represented as a GeoJSON *Feature* since they are equivalent classes.

As said, the NGSI-LD cross-domain ontology contains a reference for the GeoJSON *Geometry* class (which is the type of the value of a GeoProperty), but it does not explicitly reference the *Feature* class. Anyway, in clause 4.5.16 of the NGSI-LD specification (ETSI GS CIM 009 [i.26]) it is clearly stated that every NGSI-LD entity can be represented as a GeoJSON *Feature*.

4.6.3 SPARQL extension functions

GeoSPARQL comprises of a set of SPARQL extension functions that allow to perform geospatial queries against geometries. There are three types of query functions allowed:

- *Topological query functions*
- *Non-topological query functions*
- *Spatial Aggregate query functions*

Topological query functions may be used to assess if there exists a Topological Relation between two *Geometries*. They are predicates (functions that returns a Boolean value) that have the same meanings of the Topological Relations described in clause 4.6.2. The following example shows a query which uses the Topological Function "sfTouches" to filter the results:

Find all features that touch the union of feature my:A and feature my:D, where computations are based on my:hasGeometry.

```
PREFIX my:   <http://example.org/ApplicationSchema#>
PREFIX geo:  <http://www.opengis.net/ont/geosparql#&gt;
PREFIX geof: <http://www.opengis.net/def/function/geosparql/>
SELECT ?f
WHERE {      ?f my:hasGeometry ?fGeom .
             ?fGeom geo:asWKT ?fWKT .
             my:A my:hasGeometry ?aGeom .
             ?aGeom geo:asWKT ?aWKT .
             ?my:D my:hasGeometry ?dGeom .
             ?dGeom geo:asWKT ?dWKT .
             FILTER (geof:sfTouches(?fWKT, geof:union(?aWKT, ?dWKT))
}
```

Non-topological query functions may be used to retrieve additional information or "enriched" versions of one or more *Geometries*. For instance, one functions is the "area" function which returns a double value representing the area of the given *Geometry* in square meters; another one is the "distance" function that returns the distance between two *Geometries*. The following example shows how to use the "distance" function to order the results inside a query:

Find the 3 closest features to feature my:C, where computations are based on my:hasGeometry.

```
PREFIX uom:  <http://www.opengis.net/def/uom/OGC/1.0/&gt;
PREFIX my:   <http://example.org/ApplicationSchema#>
PREFIX geo:  <http://www.opengis.net/ont/geosparql#>
```

```
PREFIX geof:  <http://www.opengis.net/def/function/geosparql/>
SELECT ?f
WHERE {       my:C my:hasGeometry ?cGeom .
                  ?cGeom geo:asWKT ?cWKT .
                  ?f my:hasGeometry ?fGeom .
                  ?fGeom geo:asWKT ?fWKT .
                  FILTER (?fGeom != ?cGeom)
}
ORDER BY ASC (geof:distance(?cWKT, ?fWKT, uom:metre))
LIMIT 3
```

Finally, Spatial Aggregate query functions may be used to retrieve a new *Geometry* which aggregates information of the given *Geometries*. An example of a Spatial Aggregate query function is the "aggCentroid" which returns the centroid (a Point) of the set of given *Geometries*.

These SPARQL extension functions allow very powerful and expressive queries with respect to geospatial objects. NGSI-LD does support geospatial queries (GeoQueryLanguage), allowing to perform queries that are similar to the topological query functions of GeoSPARQL.

It might make no sense to provide a complex query language, as GeoSPARQL does, inside the NGSI-LD specification, but some of the non-topological and spatial aggregate functions might be good additions for the GeoQueryLanguage. For instance, GeoQueries may return enriched versions of entities containing computed information, such as the distance from the referenced geometry. Enriched versions of entities and the introduction of an "ORDER BY" statement inside the GeoQueryLanguage of NGSI-LD might allow to perform meaningful queries as "Return the three closest Cars".

4.7 Friend of a friend (FoaF)

The Friend of a Friend (FOAF [i.24]) is a project devoted to link people and information using the Web. It is built using decentralized semantic web technology and has been designed to allow the integration of data across a variety of applications, web sites and services, and software systems. FOAF provides a vocabulary to describe various characteristics and acquaintances of a person. In such a vocabulary, not only properties but also classes are defined. The FOAF vocabulary definitions are written in OWL, which makes it easy for software (reasoners) to process basic facts about the terms in the FOAF vocabulary, and consequently about the things described in FOAF documents. The URI of the namespace is "http://xmlns.com/foaf/0.1/", and is usually referenced as "foaf".

Table 4.7-1: The classes and definitions of FoaF vocabulary

Class	Definition
foaf:Agent	A class that collectively refers to 'objects that have the ability to perform certain actions', such as people, groups, and software.
foaf:Person	A class representing a person. Subclass of *foaf:Agent.*
foaf:Document	A class representing a document.
foaf:Image	A class representing an image. Subclass of *foaf:Document.*

FOAF is intended to describe not only people but also various resources related to people. The classes are provided to express a document written by a person, a photograph taken, or a group to which that person belongs. In addition to basic information such as name and e-mail address, attributes, such as hobbies, projects, and friends, are provided, and alternative information can be described.

A special feature of FOAF is that it can describe relationship between friends with the *foaf:knows* attribute. Using this relationship, it is possible to represent friends networks, groups of *foaf:Persons* interconnected to each other by *foaf:knows*. This is something that could be exploited in social networks, where it is desiderable to represent these types of networks. The following example shows the representation in XML of a *foaf:Person*.

```
<foaf:Person>
   <foaf:name>John</foaf:name>
   <foaf:mbox rdf:resource="mailto:john@xxx.xx"/>
   <foaf:knows>
      <foaf:Person>
         <foaf:name>Alice</foaf:name>
         <foaf:mbox rdf:resource="mailto:alice@xxx.xx"/>
      </foaf:Person>
   </foaf:knows>
</foaf:Person>
```

4.8 Data Catalogue Vocabulary (DCAT)

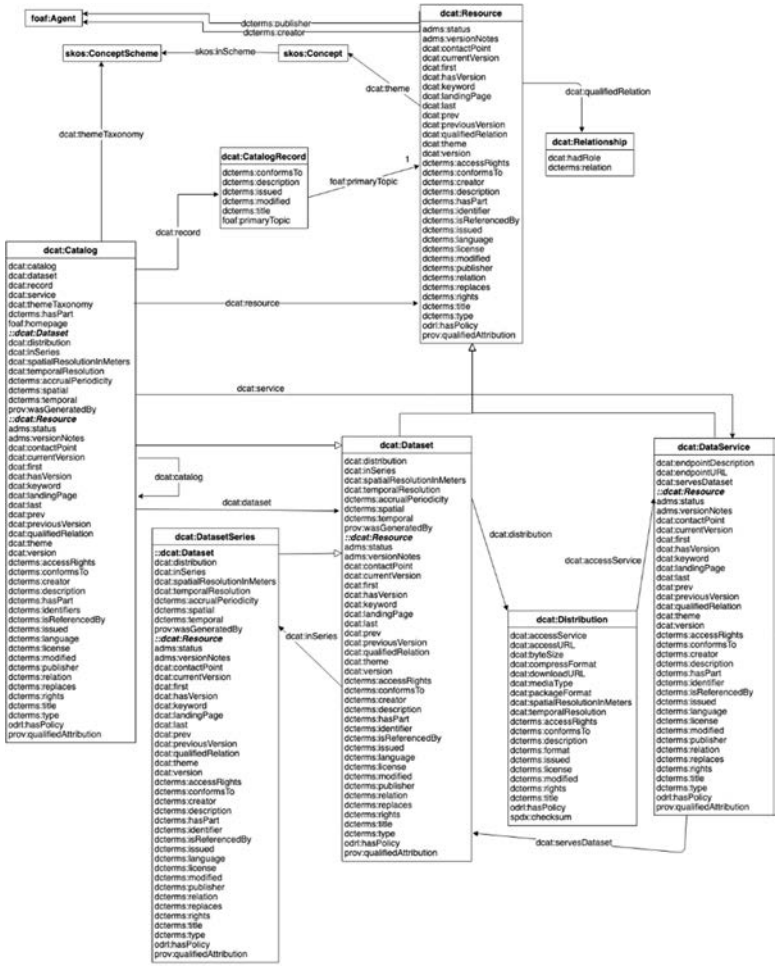

Figure 4.8-1: Overview of the DCAT model

DCAT [i.19] is an RDF vocabulary standard developed to promote interoperability between data catalogs scattered across the web. Using DCAT for the description of the data catalog, allows to take advantage of these facts:

- Easily use and fuse metadata from different data catalogs
- Data can be easily found on the web
- Various distributed data can be accessed through data catalogue management systems
- Allows to access various data with the same query and structure

The DCAT ontology depicts data resource as well as various classes and relationships between classes, including data catalogs, data services, and data distributions. The namespace for DCAT terms is "http://www.w3.org/ns/dcat#".

Table 4.8-1: The classes and definitions of DCAT

Class	Definition
dcat:Catalog	A class representing a catalog, a dataset with each element, a metadata record that describes a resource.
dcat:Resource	A class represents each element in the catalog.
dcat:Dataset	A class represents a dataset in catalog. A dataset is a collection of data created and curated by a singel agent. Data is created in many forms, including numbers, words, pixels, images, sounds, and other multi-media, and any other form that can be gathered into dataset.
dcat:Distribution	A class represents an accessible form of a dataset such as a downloadable file.
dcat:DataService	A class represents a collection of operations accessible through an interface (API) that provide access to one or more datasets or data processing functions.
dcat:DatasetSeries	A class is a dataset that represents a collection of datasets that are published separately, but share some common characteristics that group them.
dcat:CatalogRecord	A class represents a metadata record in the catalog, primarily concerning the registration information, such as who added the record and when.

4.9 WoT

The W3C Web of Things (WoT, [i.25]) is the standard that solves the problem of interworking fragmented platforms through web technology. While IoT (Internet of Things) devices communicate using various network technologies, such as IEEE 802.15™ [i.35]or TCP/IP, WoT mainly utilizes web-based communication such as HTTP and WebSocket. The WoT architecture explains the basic structure and use cases for the WoT standard and presents the requirements to create a WoT platform, a basic concept to allow the interworking between IoT platforms, and guidelines to configure devices and software.

The WoT Thing Description (TD), the core of WoT, defines the metadata of things necessary to implement the WoT. Since the WoT is not oriented toward a different IoT standard but is focused on interworking fragmented platforms, the WoT standard does not create a separate protocol for communication between objects. Instead, WoT TD defines what properties a *Thing* includes, what kind of data it generates, and its metadata so that different platforms can understand how to connect and use the device. Metadata is expressed as a JSON or as a JSON-LD document.

The WoT TD ontology is an RDF axiomatization of the TD information model, one of the building blocks of the WoT. Besides providing an alternative to the standard JSON representation format for TD documents, the TD ontology can also be used to process contextual information on Things and for alignments with other WoT-related ontologies.

5 Using External Models in NGSI-LD

5.1 Introduction and Goals

In the era of Big Data and System-of-Systems, it is not possible to map an entire system using a single data model. Clause 4 has presented reviews of a small representative of the many different types of data models. Data models differ on the types of technology used to define them (OWL, RDF, SKOS, etc.) and the domains they represent (smart cities, agriculture, health care, etc.). Despite all these differences, NGSI-LD is capable to reference the most common and known data models and it is able to represent complex concepts (as System-of-Systems) highlighting the relationships between entities, thanks to the fact that it is structured as a Property Graph.

Even though NGSI-LD is capable to reference external data models while being highly expressive, there are at least two facts that may prevent users to link external data models that are worth to point out. The **first** one is a practical issue. It may be not clear how to map concepts of an external data model in NGSI-LD. This is understandable because, as said, NGSI-LD is very expressive and there are a lot of different kinds of data models (ontologies, thesauri, SQL schemas etc.) with their own ways to represent structured data. It is, in fact, not possible to give absolute suggestions on how to map concepts since it strongly depends on the use-case, but it is possible to give few general rules that can be adapted based on the users' needs. In the present document, solutions are presented either as a suggestion to use a particular NGSI-LD element for a specific concept (e.g. SQL tables, OWL Object Property etc.), or as a general guideline to help users taking the right decision when it is not obvious which NGSI-LD element has to be used.

The **second** problem that has been noticed is potentially more concerning. Not only it may be not clear **how** to link more than one external data models in NGSI-LD (or more in general, in JSON-LD), but also it may be unclear **why** it may be useful to even link one in the first place. NGSI-LD works out of the box, because it is easy to set up and it comes with context brokers which are very powerful systems with a lot of functionalities. Even if concepts are not mapped in an external data model, every concept receives a generic URI from the context brokers, allowing users to benefit of the Data Storage, the Notification System, the Temporal API and many other functionalities ignoring the semantic part of NGSI-LD.

Even if this is legit, it is not a wise approach in the long run, and it does not take full advantage of one of the main forces of NGSI-LD which is the fact that is able to link data with its semantic meaning. There are many reasons and use cases that may benefit from the adoption of NGSI-LD combined with external data models. Some of these may be very trivial and obvious (such as to perform semantic queries or to perform input validation against a schema or a model), but there are more important and fascinating use cases that may take advantage of external data models inside NGSI-LD entities.

SUGGESTION-01	Many different use-cases can take advantage of any external data model, but it is important to note that the structure of the NGSI-LD Entities may differ depending on which data model users try to link.

One big upside of linking external data models is that it is possible to explain clearly the meaning of every single element in a data model, helping both humans and machines in their work. This is becoming more and more important because many applications and services are representations of system-of-systems: they do not just gather data from many different sources, but they have to explain how entities are connected to each other. In such applications, data belongs to many different domains, and it is not possible to find a single data model that is capable to give a definition for every single concept. For instance, in a Smart City there a lot of concepts that are defined in several different domains: there are several types of buildings, several types of sensors, several type of industries, parks with several species of plants and so on. With such a complex system, it would be very hard to understand the meaning of each concept just by reading their name. Also, it would be time consuming both to write and to read dedicated documentation with the explanation of every single element, which is needed, for instance, to the developers to understand the workflow of the application and the data structure of any element in the data model. Moreover, there can be ambiguity because many elements may have overlapping concepts. Linking external data models solves all these problems. It clarifies every single element in a model, by telling explicitly its meaning and the expected relationships/data structure, and it removes the ambiguity of overlapping concepts helping both the humans and the machines.

To help machines understanding the meaning of data is an important concept for Machine Learning models. In these years, one of the new trends is to use semantic structured data to support Machine Learning applications. An article published for Oxford Academics [i.27] states that ontologies may be used to support similarity-based analysis and Machine Learning models. The three key points behind this study are the following:

- Ontologies provide background knowledge that can be exploited in machine learning models.
- Ontology embeddings are structure-preserving maps from ontologies into vector spaces and provide an important method for utilizing ontologies in machine learning. Embeddings can preserve different structures in ontologies, including their graph structures, syntactic regularities or their model-theoretic semantics.
- Axioms in ontologies, in particular those involving negation, can be used as constraints in optimization and machine learning to reduce the search space.

In few words, a Machine Learning model may take advantage from the knowledge expressed inside an ontology to speed up the learning process. For instance, ontology-based approaches have been used to perform NLP for biomedical purposes [i.28] and to support image classification models [i.29]. Also, Machine Learning models can re-use the knowledge of many ontologies to create new knowledge. For instance, a semantic relations discovery model for a traffic safety monitoring IoT system in Smart Cities has been published in IEEE™ Xplore [i.30]. The paper [i.30] presents a way to infer new semantic rules automatically to extend reason capabilities and help their data analysis systems in order to perform tasks as connection prediction, entity prediction, relationship prediction, and attribute prediction.

Also, links to external data models are needed if the final goal is to create Knowledge Graphs. Knowledge Graphs are a relatively recent term created by Google in 2012 when they published their Google Knowledge Graph. There is not yet a clear definition of what is a Knowledge Graph, but it is reasonable to accept the definition provided by Paulheim [i.31]: "A knowledge graph (i) mainly describes real world entities and their interrelations, organized in a graph, (ii) defines possible classes and relations of entities in a schema, (iii) allows for potentially interrelating arbitrary entities with each other and (iv) covers various topical domains". In few words, a Knowledge Graph is used to represent data as a graph to be able to catch the relationships between each element and its semantic meaning. This type of representation is very common in Machine Learning Applications (worthful remark: it appears both in [i.28] and [i.30]) and in data analysis/management. Considering that GQL [i.36] and graph databases are becoming more and more popular, it is becoming more common to see implementations of Knowledge Graphs. Comparing it with other solutions, in most of the cases, NGSI-LD has the upper-hand when it comes to create a Knowledge Graph because it does not need to rely on external systems to provide semantic context for the data and because it is a Property Graph by default, capable to express the relations between entities from many different domains effortlessly.

SUGGESTION-02 When the application can interact directly with the context broker, it should always take advantage of the NGSI-LD Query system to get any additional information out of Relationships.

Lastly, NGSI-LD may be used as the data structure to represent semantic web contents for SEO purposes. Structured Data may help the search engines to reference web pages. Many search engines offer suggestions and advices on how to describe a webpage with structured data expressed in JSON-LD. When structured data is present in a website, search engines can understand the meaning of that page and decide to return it as a result of a specific query. Structured data may also be used to allow special search result features and enhancements. For example, a recipe page with valid structured data is eligible to appear in a graphical search result. Among all the possible types of structured data, the most recommended is generally JSON-LD, because it is possible to embedded JSON-LD documents in a webpage without disrupting its own structure. This means that it is also possible to adapt NGSI-LD, since it is a sub-set of JSON-LD. Currently JSON-LD is supported by the most of the search engines.

5.2 Mapping Semantic concepts in NGSI-LD

5.2.1 Review and classification of all different semantic technologies for representing data models

5.2.1.1 Overview

OWL and RDF are the basics technologies for standardized data models such as SAREF, SKOS, SOSA. Mapping of this kind of vocabulary to NGSI-LD is not a simple Task since it **strongly** depends on the use case itself and on the semantic expressivity between NGSI-LD, core (OWL, RDFS, RDF) and standardized (SAREF, SKOS, etc.) models. In the following and according to the descriptions of existing data models in clause 4, this clause provides examples and explications of mapping recommendations for these data models to map them in NGSI-LD.

5.2.1.2 Mapping SAREF ontology to NGSI-LD

Though a more comprehensive mapping of the main concepts of SAREF core ontology has been presented in ETSI GS CIM 006 [i.47], in the present clause 5.2.1.2, a small portion of the mapping of the main concepts of SAREF4ENER and SAREF4WATR extensions is presented. Although it is generally possible to follow these mappings, they may not work for every application, as it may differ depending on the specific use case.

SAREF4ENER

Figure 5.2.1.2-1 shows the mapping of the SAREF4ENER extension to the NGSI-LD meta-model.

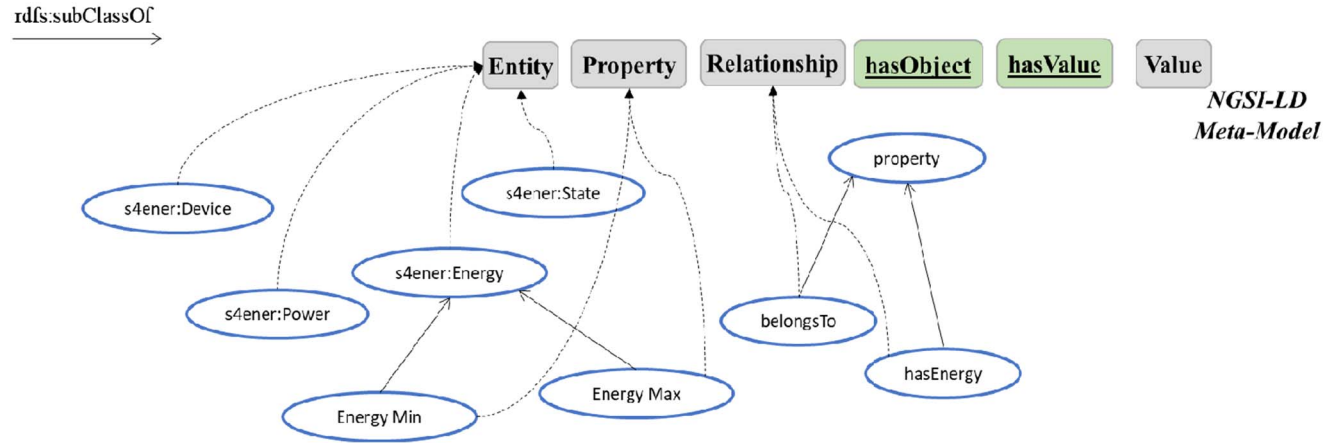

Figure 5.2.1.2-1: SAREF4ENER mapped in NGSI-LD

SAREF4WATR

Figure 5.2.1.2-2 shows the mapping of the SAREF4WATR extension to the NGSI-LD meta-model.

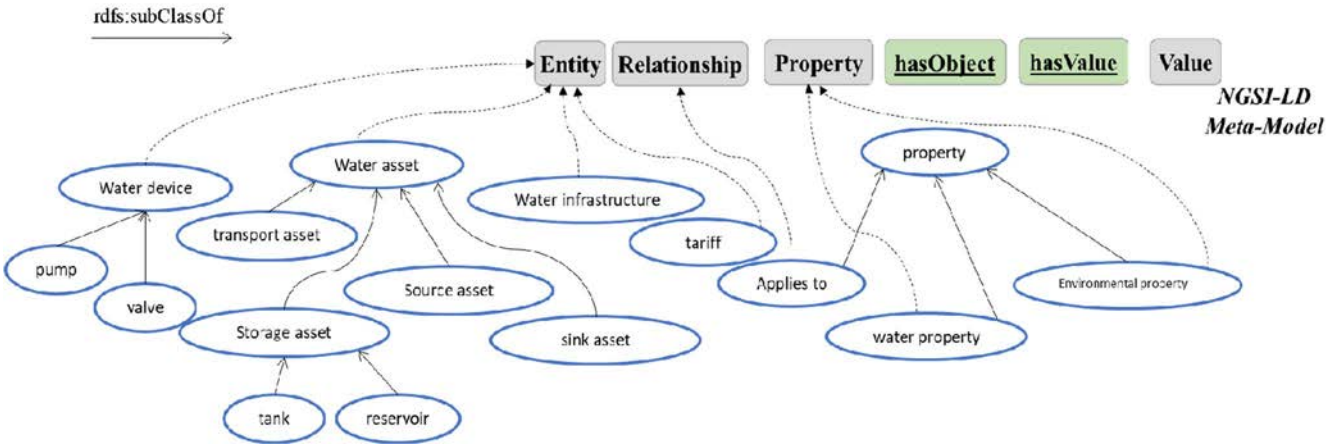

Figure 5.2.1.2-2: SAREF4WATR mapped in NGSI-LD

5.2.1.3 Mapping FoaF ontology to NGSI-LD

Figure 5.2.1.3-1 shows the mapping to NGSI-LD of the main elements defined within the FoaF ontology.

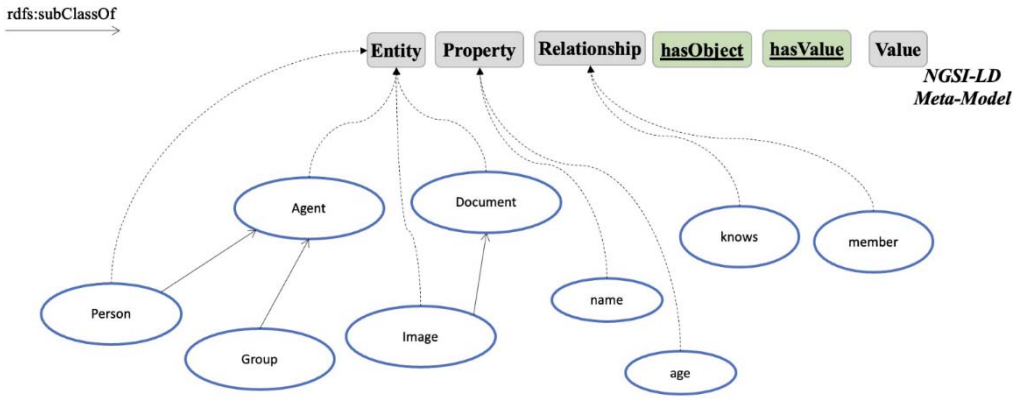

Figure 5.2.1.3-1: FoaF mapped in NGSI-LD

5.2.1.4 Mapping DCAT ontology to NGSI-LD

Figure 5.2.1.4-1 shows the mapping to NGSI-LD of the main elements of the DCAT ontology.

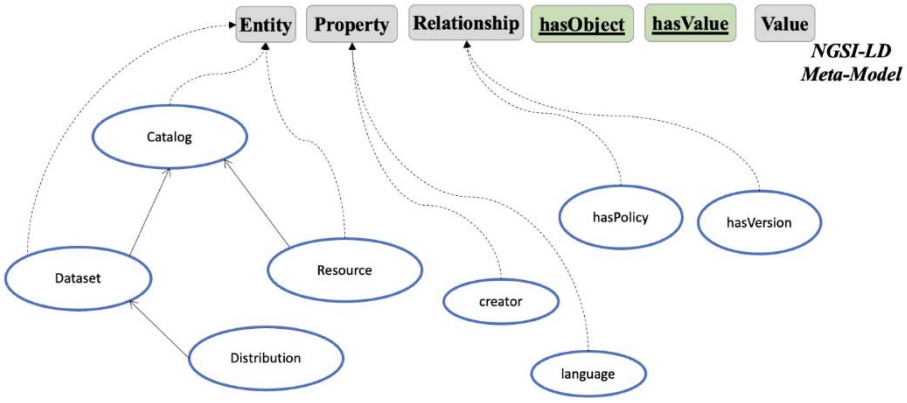

Figure 5.2.1.4-1: DCAT mapped in NGSI-LD

5.2.1.5 Suggestions to map OWL and RDFS to NGSI-LD

The examples of mapping presented in the previous clauses give an overview of how to map at least their main concepts in NGSI-LD. Anyway, the full mapping of a whole data model would not be always fair or correct because it may change depending on the use case. The following table defines a couple of mapping strategies that can be followed when mapping the main basics elements of RDFS and OWL in NGSI-LD.

Table 5.2.1.5-1: Suggestions to map OWL/RDFS in NGSI-LD

Vocabulary	Mapping and Explication
owl:Class or rdfs:Class	An OWL or RDFS class may be mapped usually to an Entity in NGSI-LD. In several cases RDFS or OWL classes may be mapped as Relationships or Properties.
rdfs:comment	A comment is used to provide a human-readable description of a resource. In NGSI-LD this may be mapped to a Property of an Entity. It could be also mapped as a Relationship or Entity of type comment.
rdfs:subClassOf	A RDFS subclass may be mapped to a Relationship that relates NGSI-LD Entities. It may be also mapped to a Property where its value is the URI of its super-class instance.
owl:DatatypeProperty	Owl data type properties are generally literal values which corresponds to a Property in NGSI-LD.
owl:ObjectProperty	Owl object properties are generally seen as relations between individuals which corresponds to Relationship in NGSI-LD.

5.2.2 Suggestions to map SKOS ontology to NGSI-LD

In NGSI-LD, SKOS ontology could help in the task of aligning the NGSI-LD data with external thesauri. The following table shows the mapping of the main elements from the SKOS ontology to NGSI-LD meta-model.

Table 5.2.2-1: Suggestions to map SKOS in NGSI-LD

Vocabulary	Mapping and Explication
skos:concept	A SKOS concept may be the type of a NGSI-LD Entity, a Relationship, or a Property.
skos:broader, skos:narrower	A SKOS broader or narrower could be mapped as Relationships relating Entities or a Property specifying the broader or the narrower of the correspondent Entity.
skos:exactMatch	A SKOS exact matching and all semantic matching concepts (close, related, broad, and narrow matches) could be seen as Properties that help in aligning the corresponding Entity to external thesauri.
skos:example, skos:definition	SKOS example and definition could be mapped as Properties as well as types of Entities.

5.3 Using URIs to point to external models

5.3.1 Link an external model with @context

The most obvious possible thing to do to link external data models in NGSI-LD would be to use long IRIs. To be more specific, long IRIs may be used to link Entity types and Attribute identifiers with concept of external data models. While being straightforward, this approach is not always practical because applications may need to work with short IRIs.

In NGSI-LD it is possible to map long IRIs with simpler and easier-to-use short IRIs. Being a subset of JSON-LD, NGSI-LD presents a dedicated field where it is possible to reference external data models called "@context". This field can be used to give a meaning (a context) for terms (most commonly short IRIs) that appear inside an Entity, as shown in the following example:

```
{
  "@context": [
   {
      "Person": "http://schema.org/Person",
      "name": "http://schema.org/name",
      "image": "http://schema.org/image",
      "email": "http://schema.org/email"
    },
    "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.5.jsonld"
  ],
  "id": "urn:ngsi-ld:person:manusporny",
  "type": "Person",
  "name": {
    "type": "Property",
    "value": "Manu Sporny"
  },
  "email": {
    "type": "Property",
    "value": "manu.sporny@example.com"
  },
  "image": {
    "type": "Property",
    "value": "http://example.com/images/manu.png"
  }
}
```

Here it is represented a NGSI-LD Entity of type "Person" having three Properties called "name", "email" and "image". As shown, the @context contains the references of all the Properties and the Entity type. For each of them it is given an URI that defines them. This URIs are used later on, when the NGSI-LD is serialized in JSON-LD, to perform an operation called "context expansion", which is the replacement of all short IRIs with the corresponding long IRIs defined inside the @context field. For instance, the result of the "context expansion" operation in the previous example would return this JSON-LD object:

```
{
  "@id": "urn:ngsi-ld:person:manusporny",
  "@type": "http://schema.org/Person",
  "http://schema.org/email": {
```

```
    "@type": "https://uri.etsi.org/ngsi-ld/Property",
    "https://uri.etsi.org/ngsi-ld/hasValue": "manu.sporny@example.com"
  },
  "http://schema.org/image": {
    "@type": "https://uri.etsi.org/ngsi-ld/Property",
    "https://uri.etsi.org/ngsi-ld/hasValue": "http://example.com/images/manu.png"
  },
  "http://schema.org/name": {
    "@type": "https://uri.etsi.org/ngsi-ld/Property",
    "https://uri.etsi.org/ngsi-ld/hasValue": "Manu Sporny"
  }
}
```

As it is possible to see above, all short IRIs has been replaced with the long IRIs expressed in the @context.

SUGGESTION-03 Even though it would be *possible* to omit the @context member since Context Brokers enforce the core context in every Entity (hence, they apply fake long IRIs to every term that has not a mapping in the @context), if the goal is to reference external data model, the **most effective** way to do it is to provide a custom @context.

The value inside the @context field can be expressed in three ways:

- **Direct Definition**, by providing a JSON with the mapping of short IRIs and long IRIs
- **Indirect Definition**, by providing an URL which points to the **JSON-LD @context file** that holds the context definition of a data model
- **Array of Definitions**, by providing an array that contains either direct and/or indirect definitions

For instance, in the previous example, the @context was defined with an array which contains the direct definitions of terms and the URL of the NGSI-LD core context (indirect definition).

While the direct and indirect definitions are straightforward, when using an array of definitions, it is strongly suggested to order the definitions from the least important (or least meaningful) to the most important (or most meaningful). In fact, during the JSON-LD serialization, if the @context is an array, the serializer would always start from the first element of the array to the last one, overwriting results if there are **overlapping concepts**. It is important to remember that every Context Broker is going to apply the NGSI-LD core context at the end of the serialization, which means that it is not possible (*not allowed*) to re-define terms featured inside the cross-domain ontology or the meta-model of NGSI-LD.

SUGGESTION-04 When deciding how to build a custom @context, users should rely on arrays of definition because they are more powerful, and they allow to overcome common problems (as for overlapping concepts).

SUGGESTION-05 Considering that the order inside an array of definitions matters, users should order all the external data models by importance in increasing order (least important to most important).

Generally speaking, most of the data models (such as SAREF, schema.org etc.) provide their definition as a JSON-LD @context file. Even though context brokers do not store the definition of the data models contained in a JSON-LD @context file, it is **strongly suggested** to reference these files as Indirect Definitions because they contain all that is needed to reference the concepts of the external data models. If for any reason it is not suitable to directly reference a JSON-LD @context file, it is **suggested** to review its @context member to see how it **references** concepts inside the data model, and which are all the external data models referenced.

Few questions may arise at this point: "How are concepts referenced in general? What is the main mechanism?" Mainly, to reference concepts of external data models they use **JSON-LD Compact IRIs**. A compact IRI is a prefix/namespace associated with the root IRI of a data model. It may be used to explicitly tell to the serializer what is the data model which defines a concept. Both the SAREF family and Schema.org use this approach: in their JSON-LD @context files the @context members contain the prefixes of all the data models used to reference their concepts.

For instance, if one would use a prefix for the "schema.org" data model with the previous example, the resulting NGSI-LD Entity would look like the following:

```
{
  "@context": [
   {
      "schema": "http://schema.org/"
    },
    "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.5.jsonld"
  ],
  "id": "urn:ngsi-ld:person:manusporny",
  "type": "schema:Person",
  "schema:name": {
    "type": "Property",
    "value": "Manu Sporny"
  },
  "schema:email": {
    "type": "Property",
    "value": "manu.sporny@example.com"
  },
  "schema:image": {
    "type": "Property",
    "value": "http://example.com/images/manu.png"
  }
}
```

In this case, "schema" is the prefix/namespace that refer to the external data model that is obtainable from "https://schema.org/". It should not be confused with the definition of a concept. The serializer is smart enough to understand that this is a prefix/namespace because the IRI associated with the key ends with a specific character: IRIs ending with "/" or "#" are considered as prefixes/namespaces.

Thanks to this, it is possible to provide compact IRIs for all the concepts (Person, name, email, image) using the prefix "**schema:**", linking all these concepts specifically to the ones defined in "schema.org".

Considering how the JSON-LD @context file provided with "schema.org" is structured, the same result can be achieved this way:

```
{
  "@context": [
    "http://schema.org/",
    "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.5.jsonld"
  ],
  "id": "urn:ngsi-ld:person:manusporny",
  "type": "schema:Person",
  "schema:name": {
    "type": "Property",
    "value": "Manu Sporny"
  },
  "schema:email": {
    "type": "Property",
    "value": "manu.sporny@example.com"
  },
  "schema:image": {
    "type": "Property",
    "value": "http://example.com/images/manu.png"
  }
}
```

This behaviour is acceptable because inside the JSON-LD @context file of "schema.org" already exists a prefix "**schema**" for "**https://schema.org/**".

SUGGESTION-06 If it is not suitable to directly reference a JSON-LD @context file, when deciding which key to choose for a prefix, it is usually better to use the same key used in the JSON-LD @context file to avoid ambiguity.

Compact IRIs are very useful, and they can be used to fix very common problems, such as the overlapping problem that may occur when linking many different data models (as discussed in clause 5.5), but, unfortunately, they cannot be used in every situation. Compact IRIs just performs **concatenation**: serializers of JSON-LD version 1.1 are not smart enough to understand by themselves which IRI to use as a prefix to reference any concepts. Moreover, it is not possible to give compact IRIs that do not end with a "/" or a "#", which is a problem if a prefix ends with other symbols. Yet another problem is that some data models (as Agrovoc does) use numeric IDs in their IRIs. For instance, the IRI for "Water" is the following: "http://aims.fao.org/aos/agrovoc/c_330633".

Finally, not all the external data models provide a JSON-LD @context file, in which is possible to find compact IRIs to use when referencing concepts, which helps in linking the external data model inside NGSI-LD Entities. If definitions are missing or it is not possible to use compact IRIs, the only solution is to provide a custom @context of Direct Definitions as the following:

```
{
  "@context": [
     {
        "agrovoc_Water": "http://aims.fao.org/aos/agrovoc/c_330633",
        "agrovoc_Iron": "http://aims.fao.org/aos/agrovoc/c_3950b"
      }
    ]
}
```

Basically, the only way is to define a map where each key is the **custom** compact IRI and the value is the expanded IRI. Custom compact IRIs can be used to reference types and Attributes inside a NGSI-LD Entity as described previously. Interestingly, this is the approach of the **SmartDataModels Program**: for every domain there exists a specific custom @context files that contain direct definitions for the most relative concepts. This approach is suggested when it is not possible to reference a concept directly with a prefix or when there are overlaps with other concepts.

SUGGESTION-07 If compact IRIs are needed to reference a concept, they should be structured to avoid ambiguity when used with external data models that rely on compact IRIs (as the SmartDataModels). Not only the chosen IRI should be meaningful, but it is preferable to make it unique within the @context definition to avoid: (1) overwriting other concept definitions with the same compact IRI and (2) being overwritten by subsequential concept definitions with the same compact IRI.

In any case, there are way more sophisticated things that custom @context files allow, but they are not relevant for the scope of the present document. More details and examples can be found in the official documentation [i.33].

5.3.2 Entity multi-typing

Multi-typing has been introduced as one of the new features of v1.5.1 of NGSI-LD API. In both object-oriented modelling and RDF/OWL models, multi-typing is one of the most powerful features when it is important to define complex concepts.

In NGSI-LD, multi-typing can be used with different goals in mind:

- To define complex concepts without creating a new type
- To provide more specific details without adding attributes/sub-attributes

When multi-typing is not supported, the only workaround to define complex concepts is to create new specific types. The problem with this approach is that, not only it is expected to create a new type every time, but it is not possible to get advantage of having a prior structure defined in an external ontology or data model. For instance, a MotorHome would be defined with single typing as a separate class. It would be needed to replicate definitions of relevant attributes from "Vehicle" and "Home", since it is both a vehicle and a home. Though multi-typing it is possible to avoid replicating definitions since they would be inherited directly from the two classes rather than having to create a new type. Also, multi-typing makes unneeded to create very narrowly specific classes for slightly different categories as a "Caravan" type. In NGSI-LD multi-typing is easily achieved by adding an array of URIs as the value of the member "type" of an entity.

```
{
  "@context": [
    {
      "vehicle": "https://example.com/vehicles-ontology/",
      "building": "https://example.com/buildings-ontology"
```

```
    },
    "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.5.jsonld"
  ],
 "id": "urn:ngsi-ld:domus:sensor:MotorHome1",
  "type": ["vehicle:Motor", "building:Home"],
  …
}
```

In the example above, it is possible to see how to define a "MotorHome" entity with multi-typing. It is interesting to note that it is totally possible to link many different ontologies when doing this, to create a more specific concept. Moreover, it is not needed to define a new Attribute to slightly change what it is represented: if for example the concept should be a "Caravan" there are many different ways to do it:

- It may be possible to add a third more specific type (e.g. "vehicle:Caravan").
- It may be possible to update one of the existing types with a more specific one (e.g. change "vehicle:Motor" to "vehicle:Caravan").

As a last remark on this point, context brokers will not check if the classes used in multi-typed Entities are compatible. So, the application has to double check if the classes used to define the type of an Entity can be together to prevent creating new types that actually makes no sense and/or contradict what it is stated in the external data models.

SUGGESTION-08 When using multi-typing, if data validation is an important feature for the application, users need to overcome every overlapping concept providing direct definitions (clause 5.3.1) or adjusting the @context accordingly (clauses 5.5.2 and 5.5.3).

Another scenario where multi-typing is highly recommended is when the goal is to characterize specific features or peculiarities of an instance (entity) that are **static/immutable**. For example, in an eHealth scenario there could be many different IoT sensors: wearables, thermometers, ECGs, etc. Same types of sensors may have different features: for instance, wearables may differ based on the brand, model, main functionalities, etc. The most simple and naive approach would be to add a set of specific Attributes to specify this type of information. This approach is not always wrong (it depends on the use case), but it does not get advantage of the multi-typing system combined with the use of external data models. All of this information can be automatically linked inside an instance by adding a specific type of a use-case-specific ontology/data model that express the main differences between two instances of the same type/class/restriction.

```
{
  "@context": [
    {
      "s4e": "https://saref.etsi.org/saref4ehaw/v1.1.1/",
      "s4w": "https://saref.etsi.org/saref4wear/v1.1.1/",
      "apple": "https://example.com/apple-ontology"
    },
    "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.5.jsonld"
  ],
  "id": "urn:ngsi-ld:domus:sensor:AppleWatch2",
  "type": ["s4e:Sensor", "s4w:Wearable", "apple :WatchSeries6"],
  …
}

{
  "@context": [
    {
      "s4e": "https://saref.etsi.org/saref4ehaw/v1.1.1/",
      "saref": "https://saref.etsi.org/core/v3.1.1/",
      "xiaomi": "https://example.com/xiaomi-ontology"
    },
    "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.5.jsonld"
  ],
  "id": "urn:ngsi-ld:domus:sensor:AABBCC112233",
  "type": ["s4e:Sensor", "saref:TemperatureSensor", "xiaomi :Mi2Temperature"],
  …
}
```

In the previous example it is possible to see how multi-typing can be used to give more specific details. The first class (Sensor) is the most generic thing that it is possible to state when defining these entities: in this use-case, every IoT device is a Sensor. The second class and the third class are more specific, giving in order:

- the type of a sensor (either a *Wearable* or a *TemperatureSensor*); and
- the exact model of a sensor (either a *WatchSeries6* or a *Mi2Temperature*).

Using the multi-typing approach, it is not needed to define specific Attributes to express this kind of information. Data models can already express "**static**" information, which does not change with time. This way, ontologies and data models can be used as external documentation to enrich entities with additional knowledge. In any case, this approach is use-case dependent and it should be used only when really needed: if too many classes are used to distinguish the main differences between entities, probably it means that the instances are not just representing a slightly different concept, but a very different one. They are either too complex or too specific and it would be better to split the entity in more than one and/or to remove the broader classes, leaving only the most relevant.

5.3.3 Attribute typing using @context

As described in clause 5.3.1, @context can be used to reference external concepts from external data models. In a NGSI-LD Entity there are two types of **Attributes** that can be linked with external data models:

- **Property** and its sub-classes (GeoProperty, LanguageProperty, etc.)
- **Relationship**

As discussed in clause 5.1, users are not forced to link external data models to Attributes because fake URIs are provided when the identifier of an Attribute is not a valid URI. Anyway, with fake URIs the knowledge of the type of an Attribute is practically lost because the identifier is not a "type", but a short description of the data stored inside the "value" member (in case of a Property) or of the link between the Entity/Attribute and another Entity (in case of Relationship). The following example shows an Entity with fake URIs.

```
{
  "@context": ["https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.5.jsonld"],
  "id": "urn:ngsi-ld:person:manusporny",
  "type": "Person",
  "name": {
    "type": "Property",
    "value": "Manu Sporny"
  },
  "knows": {
    "type": "Relationship",
    "object": "urn:ngsi-ld:person:lucasgreen"
  }
}
```

In the example above, there are two Attributes inside the entity: a Property called "name" and a Relationship called "knows". Even if it is possible to suppose the type inside the "value" member or the type of the entity with id "urn:ngsi-ld:person:lucasgreen", it is still not possible to know beforehand if there are invalid values (for instance, out of range values) or incompatible types. In top of that, if the identifier is a broader term (such as "measures") it is very hard for the human to suppose what should be stored in that field.

SUGGESTION-09 Explicit is better than implicit. Linking external data models to define Attributes removes the ambiguity of the terms used within an NGSI-LD Entity.

Linking external data models inside the @context can fix these problems. For example, supposing that one would link the "schema.org" and the "friend of a friend" data models to define both "name" and "knows", the entity would look like this one:

```
{
  "@context": [
   {
      "schema": "http://schema.org/",
      "foaf": "http://xmlns.com/foaf/0.1/"
    },
    "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.5.jsonld"
  ],
  "id": "urn:ngsi-ld:person:manusporny",
```

```
  "type": "schema:Person",
  "schema:name": {
    "type": "Property",
    "value": "Manu Sporny"
  },
  "foaf:knows": {
    "type": "Relationship",
    "object": "urn:ngsi-ld:person:lucasgreen"
  }
}
```

In this case all Attributes have been linked with external data models. The explicit definition of the Attributes' identifier as "schema:name" and "foaf:knows" allows many things to happen:

- It is possible to state whether the Property "schema:name" and the Relationship "foaf:knows" may exist inside an entity of type "schema:Person".
- It is clear that the type of the "value" is "String". So, for instance, an application should not put an Integer inside the "value" member.
- It clarifies the semantic meaning of the "value". It is not simply a "String", it is a name of a "schema:Person".
- It is clear that the type of the entity "urn:ngsi-ld:person:lucasgreen" has to be "schema:Person" (which is equivalent with "foaf:Person").

All this knowledge may be used to perform input validation before creating/updating an entity inside the context broker. Even if it is not possible to perform input validation inside a context broker with the current API, there could be specific use-cases where this would be needed. Possible suggestions on why and how to do it are given in clause 6.2.

Furthermore, when external data models are referenced in Attributes' identifiers, application of third-parties can take advantage of this fact. For instance, Search Engines may use this knowledge to understand what is contained inside a webpage (as discussed in clause 5.1). NGSI-LD Entities may be used to perform this task. For example, Google Search Engine can read JSON-LD documents inside a webpage to understand what it is about. Anyway, it is not able to understand the classical structure of NGSI-LD Attributes. So far, all the examples were about NGSI-LD Entities expressed in **Normalized Representation**. This is more verbose, and it may be not supported by third-party applications that cannot directly access the context broker (as for a search engine); for SEO purposes, the suggestion is to use NGSI-LD Entities that are represented with the **Simplified Representation**. Simplified Representation is a **lossy** representation of an Entity where, for each Attribute the value is either the content of the "value" member (Property) or the "object" member (Relationship). This representation works with Google Search Engine because Properties have a simpler structure, closer to what the Search Engine expects to see, suggesting that it is possible to use NGSI-LD Entities to describe the content of a webpage. Anyway, it is still important to underline that Relationships are ignored by search engines, as they expect actual values/types of the mapped concepts instead of IDs, so required members of a concept should be represented as Property.

Lastly, there is still one thing to note. Since NGSI-LD relies on blank nodes to describe reified members (as most of the Attributes), this may lead to the loss of the semantic type of Attributes outside context brokers, but this information is not actually lost. In fact, when external data models are referenced, it is always possible to infer the semantic type of an Attribute:

- The name of a Property may be treated as the semantic Type of what is stored inside the "value" member.
- The semantic types of the NGSI-LD Entity's ID stored inside the "object" member of a Relationship can always be inferred by performing an NGSI-LD query to retrieve the types of the given Entity.

The problem may arise when it is not possible to contact the context broker directly. More specifically, third parties may want to know about the semantic types of NSGI-LD Attributes, but they cannot ask directly the context broker. In this case, it may be not possible to actually tell the "type" of the concept.

For instance, Figure 5.3.3-1 shows how the NGSI-LD Entity of the previous example is represented in RDF.

Subject	Predicate	Object	Language	Datatype
_:b0	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	https://uri.etsi.org/ngsi-ld/Property		
_:b0	https://uri.etsi.org/ngsi-ld/hasValue	Manu Sporny		
_:b1	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	https://uri.etsi.org/ngsi-ld/Relationship		
_:b1	https://uri.etsi.org/ngsi-ld/hasObject	urn:ngsi-ld:person:lucasdavids		
urn:ngsi-ld:person:manusporny	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://schema.org/Person		
urn:ngsi-ld:person:manusporny	http://schema.org/name	_:b0		
urn:ngsi-ld:person:manusporny	http://xmlns.com/foaf/0.1/knows	_:b1		

Figure 5.3.3-1: Person entity without Datatypes for Attributes

As it is possible to see, the Datatype of both the Attributes is unknown: it is in fact lost since the "value" and "object" members does not have a "@type" field. Before going through possible solutions, it is important to say that there is not a clean solution that works for every situation. So, the solution adopted may vary depending on the use case.

For Properties, it is possible to solve this problem easily. If the value of the "value" member is **not** a native JSON, it can be expressed as a typed JSON-LD object, which is a JSON which contains an "@type" member that is an IRI (as the "type" member of an Entity). The big upside of this approach is that it is possible to assign more than one type since the "@type" member expects a string or an array of strings. The downside is that it is not possible to use it for Relationships or for Properties that have a value that is a native JSON. When the value of the "value" member is a native JSON, it is possible to provide, as the value of the "value" member, a specific JSON having this structure:

```
{
  …
  "schema:name": {
    "type": "Property",
    "value": {
       "@type": "schema:name",
       "@value": "Manu Sporny"
    }
  }
}
```

The JSON object is what is called a **compound** value: the member "@value" contains the value of the Property, meanwhile the member "@type" contains the IRI of the semantic type of what is stored inside the "@value" member. Even in this case multi-typing is possible, but there are two downsides. While this is allowed by the NGSI-LD specification, to structure Properties differently based on their semantic type. In some cases, it may be desirable to keep the "value" members of Properties inside a context broker as simple as possible. Furthermore, neither of these approaches can be used to add the type of the "object" member of a Relationship.

There is actually a way to inject the type inside specific Attributes using the concept of **scoped @context**. This approach allows to carry on the type of an Attribute outside of a context broker without disrupting the structure of a NGSI-LD entity. Scoped @context are nested definition of local @context members within the definition of a concept inside the top-level @context provided with an entity. They can be used to provide additional information such as the type of an Attribute. A way to do this is shown in **bold** in the following example:

```
{
  "@context": [
        {
            "schema": "http://schema.org/",
            "foaf": "http://xmlns.com/foaf/0.1/"
        },
        {
            "foaf:knows": {
                "@id": "foaf:knows",
                "@context": {
                    "object": {
                        "@id": "ngsi-ld:hasObject",
                        "@type": "schema:Person"
                    }
                }
```

```
            }
        },
        {
            "schema:name": {
                "@id": "schema:name",
                "@context": {
                    "value": {
                        "@id": "ngsi-ld:hasValue",
                        "@type": "schema:name"
                    }
                }
            }
        },
        "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.5.jsonld"
    ],
  "id": "urn:ngsi-ld:person:manusporny",
  "type": "schema:Person",
  "schema:name": {
    "type": "Property",
    "value": "Manu Sporny"
  },
  "foaf:knows": {
    "type": "Relationship",
    "object": "urn:ngsi-ld:person:lucasgreen"
  }
}
```

Inside the @context there are two new direct definition for "foaf:knows" and "schema:name". The definitions are placed after prefixes definition to prevent the serializer to overwrite them. They both contains a nested "@context" member that is needed to modify the types of "value" (for schema:name) and "object" (for foaf:knows). With this new @context the resulting conversion to RDF triples will result in what it is seen in Figure 5.3.3-2.

Subject	Predicate	Object	Language	Datatype
_:b0	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	https://uri.etsi.org/ngsi-ld/Property		
_:b0	https://uri.etsi.org/ngsi-ld/hasValue	Manu Sporny		http://schema.org/name
_:b1	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	https://uri.etsi.org/ngsi-ld/Relationship		
_:b1	https://uri.etsi.org/ngsi-ld/hasObject	urn:ngsi-ld:person:lucasgreen		http://schema.org/Person
urn:ngsi-ld:person:manusporny	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://schema.org/Person		
urn:ngsi-ld:person:manusporny	http://schema.org/name	_:b0		
urn:ngsi-ld:person:manusporny	http://xmlns.com/foaf/0.1/knows	_:b1		

Figure 5.3.3-2: Person Entity with defined Datatypes for Attributes

As shown in the figure, now the Datatype of each Attribute is clearly stated. There are other approaches to achieve this result, but this approach is preferable because the NGSI-LD Entity is always a valid Entity. In fact, the types will appear only when the custom @context will be applied outside the context broker. Moreover, the API specs do not allow to change the structure of an Entity inside a context broker. It is not possible to use this custom context when creating an Entity, because scoped contexts are ignored by context brokers. Still, it is possible to use it outside, from the client side.

When serialized outside of a context broker, the resulting NGSI-LD Entity would look like this one:

```
[
  {
    "http://xmlns.com/foaf/0.1/knows": [
      {
        "https://uri.etsi.org/ngsi-ld/hasObject": [
          {
            "@type": "http://schema.org/Person",
            "@value": "urn:ngsi-ld:person:lucasdavids"
          }
        ],
        "@type": [
          "https://uri.etsi.org/ngsi-ld/Relationship"
        ]
      }
    ],
    "@id": "urn:ngsi-ld:person:manusporny",
    "http://schema.org/name": [
```

```
      {
        "@type": [
          "https://uri.etsi.org/ngsi-ld/Property"
        ],
        "https://uri.etsi.org/ngsi-ld/hasValue": [
          {
            "@type": "http://schema.org/name",
            "@value": "Manu Sporny"
          }
        ]
      }
    ],
    "@type": [
      "http://schema.org/Person"
    ]
  }
]
```

It is now possible to know the semantic type of an Attribute because the "value" and "object" members are now expressed as compound values. This approach is suitable when the goal is to preserve the structure of an NGSI-LD Entity inside a context broker, while providing the semantic type of Attributes outside the context broker. There are two downsides. The first one is that scoped @context may be not supported by the serializer used by the application of third-party. The second one is that it is not possible to use multi-typing since in scoped @contexts the "@type" member expects only one string.

5.3.4 Create a custom @context file

In previous sections, there have been given a lot of recommendations on how to build a custom @context but defining custom @contexts directly on a NGSI-LD Entity **is not** the best way to do it for many reasons. In medium to big applications, custom @context definitions may be very big and the creation and building of a custom @context member are performed every time an Entity has been created. More importantly, the **biggest downside** of using custom @context defined directly inside Entities is that @context members are **removed** from Entities when stored inside a broker. This affects directly queries and Subscriptions: if it is desired to return Entities with short IRIs as the result of a query (using the LINK Header) or the content of a Notification (defining it inside the Subscription create request), the only way to achieve this is to provide the URL of an @context file that is able to convert all the long IRIs inside the Entities back to short IRIs.

Definitely, it is strongly suggested to setup a custom @context file when using many external data models, especially when there are more than one inside the same application.

SUGGESTION-10 Users need to define at least one custom JSON-LD @context file because it is the only way to apply the correct @context when interacting with the APIs (create an NGSI-LD Entity, queries, create NGSI-LD subscriptions, etc.).

The creation of an @context file is a very simple operation. Basically an @context file is a JSON-LD file that is structured as shown in the following example:

```
{
  "@context": [
    {
      "saref": "https://saref.etsi.org/core/v3.1.1/",
      "sosa": "https://www.w3.org/ns/sosa/"
    },
    …
    "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.5.jsonld"
  ]
}
```

Generally, @context files have to contain at least one of the following two members:

- @context
- @graph

Since the @graph members are not used inside the NGSI-LD, when provided for NGSI-LD Entities, it is expected that the @context file contains the @context member. Inside the @context member it is possible to do everything that has been shown in the previous clauses.

SUGGESTION-11 Since every Indirect Definition brings the serializer of the context broker to download an additional JSON-LD @context file, it is suggested to avoid using many Indirect Definition (especially nested definitions) to speed-up operations with the APIs.

When projecting the structure of a custom @context file for NGSI-LD Entities, it is important to remember that the order matters. In particular, there are basically two things to keep in mind:

- If the @context member contains an array of definitions, what comes first may be **overwritten** by next definitions (see clause 5.5.2).
- The last definition has to always be the **NGSI-LD core context**. Implementations of context brokers have to add it automatically, but it is suggested to provide it. The reason behind this is that it is not possible to modify concepts defined inside the NGSI-LD core context (such as type, Property, observedAt, etc.).

Once the file is ready, it has to be publicly downloadable using a GET HTTPS request. It is possible to refer to this file by just simply passing the URL of the file inside the @context of the NGSI-LD Entity. If for example an @context file is reachable at the following URL: "https://example.com/custom-context.jsonld", it can be linked in an NGSI-LD Entity this way:

```
{
  "@context": ["https://example.com/custom-context.jsonld"],
  "id": "urn:ngsi-ld:person:manusporny",
  "type": "schema:Person",
  "schema:name": {
    "type": "Property",
    "value": "Manu Sporny"
  },
  "foaf:knows": {
    "type": "Relationship",
    "object": "urn:ngsi-ld:person:lucasgreen"
  }
}
```

As said, custom @context files become very important when there a lot of concepts and many different data models within the same application. The size or the number of @context files that may be needed to create for the same application depend strongly from the use case. Generally, one single @context file is enough, but it may be not the case when the context broker is deployed in a federated deployment, because the knowledge is divided between many different nodes of the architecture and it may be not possible (nor needed) to hold all the references in the same @context file.

Finally, as hinted at the beginning of this clause, the custom @context is needed when the goal is to keep the short IRIs inside Entities especially when querying Entities or when setting up Subscriptions. Basically, there are three possible approaches to tackle this problem:

- Use one @context to create and retrieve every Entity
- If more @context files were used to create Entities, provide a specific @context when retrieving Entities that is the result of a merge operation of two or more of those @context files
- After retrieving the Entities, the application may apply a custom @context to format all the long IRIs back to short IRIs

All of these solutions have pros and cons, but they show how powerful can be NGSI-LD. Especially the third solution shows that it is always possible to use a specific @context that map long IRIs back in short IRIs. The long IRI (which is the universal identifier of a concept) is still kept and preserved, meanwhile the application can access the values inside Attributes by using the short IRIs. In a sense, the @context provided can be used to "adapt" the NGSI-LD Entity to appear according to what it is expected by the client application.

SUGGESTION-12 When the references are split in many @context files, it may be problematic to keep short IRIs when querying Entities since they may have been created using several JSON-LD @context files. In this case, it is either suggested to provide the URL of a custom JSON-LD @context file which merges the needed references or to let the application apply the custom @context file after retrieving the Entities.

5.3.5 Representing equivalence and hierarchy

NGSI-LD does not have a concept of semantic queries because it is model-agnostic. It is possible to reference external data models, but Context Brokers are not reasoners (nor it is intended to allow them to act as reasoners). For this reason, NGSI-LD **does not support** common features that are represented in complex data models (as OWL ontologies). To be more specific, it is **not possible** to directly represent the equivalence between concepts or the hierarchy between classes.

Anyway, there could be use cases where it would be neeeded to represent equivalence, hierarchy or other similar links between concepts, using them to query for Entities. For instance, the application would need to query for all the Entities that are equivalent with "sosa:Sensor" or all the Entities that are sub-class of "saref:Device". How to represent it in NGSI-LD?

Even though NGSI-LD does not explicitly support these features, there actually a couple of options that a modeler may adopt depending on the situation. To keep it simple, in this clause equivalence and hierarchy are directly referenced, but the principles expressed can be applied to any other type of semantic links that NGSI-LD is not able to represent.

The best fitting proposal would be to use Properties to represent these links. Using Properties, it is possible not only to represent equivalence/hierarchy between Entities, but it is also possible to represent the same links between Attributes (using sub-Properties). A Property can represent semantic links following these two steps:

- The name of the Property/sub-Property should be the identifier of the semantic link (e.g. owl:equivalentClass). *Optional*.
- The value of the Property/sub-Property should be a string or an array of strings.

The main pro of this representation is that the values of Properties can be queried asking directly the context broker, so it is very easy to obtain a list of Entities that share a semantic link. For instance, it is possible to perform these types of queries:

1) owl:equivalentClass==http://www.w3.org/ns/sosa/Sensor

It returns all the Entities that have a Property called "owl:equivalentClass" having a value that matches/contains "http://www.w3.org/ns/sosa/Sensor"

2) foaf:name.owl:equivalentDataProperty==http://schema.org/name

It returns all the Entities that have a Property called "foaf.name", having a subProperty called "owl:equivalentDataProperty" which value matches/contains "http://schema.org/name"

Anyway, there is a big downside: values are not expanded during the context expansion of JSON-LD serialization. Hence, the value has to be the long IRI (or a common short IRI known to the whole application). Furthermore, neither the value to match expressed in the query is expanded. So, it is clear that the application has to decide to use long IRIs or short IRIs consistently when using Properties. Finally, one another downside is that this approach may lead to inconsistency problems because of the redundancy of information: every Entity has to be updated if a semantic link is added/removed.

Generally, Properties would be the way to go to represent these links, but there may be also other ways to represent semantic links. It would be possible to follow these routes:

- Multi-typing
- Relationships

Multi-typing may be used when it is not important to represent semantic links between Attributes, while using short IRIs (context expansion always works with types). The real problem when using multi-typing this way is that the application has to know that this feature is used this way: inside a context broker it will not be possible to distinguish whether multi-typing is used to define complicated classes/restrictions or if it is used to represent equivalence/hierarchy.

Relationships may be used if the goal is to represent meta information/meta Entities. These comes with two downsides: it forces to create one Entity for each concept that has to be represented and datasetIds should be used to represent 1-to-many Relationships (if there is more than 1 semantic link to represent).

5.4 Representing schemas in NGSI-LD

5.4.1 SQL

Structured Query Language (SQL) is a domain-specific language used in programming and designed for managing data held in a Relational Database Management System (RDBMS). It is particularly useful in handling structured data, i.e. data incorporating relations among entities and variables.

SQL structures can be transformed into NGSI- LD entities effortlessly. The indexes (Relationship between tables) are implemented by the relationships between different entities stored in NGSI-LD. However, in the data types conversion of SQL to NGSI-LD, it will cause some information loss as long as the different numeric types are not considered in NGSI-LD (NGSI-LD inherits data types of JSON-LD). Thus, all numeric types but integers will be mapped to the type *number*, independently of the original precision.

For instance, taking in account the following SQL definitions of tables:

```
CREATE TABLE Person (
    PersonID int,
    LastName varchar (255),
    FirstName varchar (255),
    Address varchar (255),
    City varchar (255),
    Height float,
    OrganizationID int
);
```

```
CREATE TABLE Organization (
     Name varchar (255)
);
```

Both the definitions can be mapped into the following equivalent NGSI-LD Entities:

```
{
  "id": "uri:ngsi-ld:Person:0003",
  "type": "Person",
  "LastName": {
     "type": "Property",
     "value": "mate en salsa"
  },
  "FirstName": {
     "type": "Property",
     "value": "Jacinto"
  },
  "Address": {
     "type": "Property",
     "value": "Castellana 285"
  },
  "City": {
     "type": "Property",
     "value": "Astorga"
  },
  "Height": {
     "type": "Property",
     "value": "1.77"
  },
  "OrganizationID": {
     "type": "Relationship",
     "object": "uri:ngsi-ld:Organization:0001"
  },
  "@context":
     [
       "https://uri.etsi.org/ngsi-ld/v1/ngsi-
ld-core-   context-v1.5.jsonld"
     ]
}
```

```
{
  "id": "uri:ngsi-ld:Organization:0001",
  "type": "Organization",
  "Name": "Mantecadas Astoga",
  "@context":
     [
       "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-
core-   context-v1.5.jsonld"
     ]
}
```

Finally, as a guideline, table 5.4.1-1 shows how the data types may be converted when mapping from SQL to NGSI-LD.

Table 5.4.1-1: Data type conversion

SQL data types	JSON-LD types (those accepted by JSON Schema)
char, varchar	string
Bit, tinyInt, smallint, mediumint, bigint	number
float, decimal, double	Integer
Bool, boolean	boolean
Date, timestamp, time, datetime	String (format date and date-time)
Json (see note)	Object
Array (see note)	Array
NOTE: Some SQL databases accept these types of values.	

The other way around is also feasible. JSON schema representation of data models can be exported to SQL in the most part: as long as these JSON schemas represent the structure of NGSI-LD payloads, therefore the payloads can be exported to SQL with the next preventions.

Some conditional structures and options have to be expanded for being allocated into SQL outcomes. SQL data types exceed the ones in JSON schema. It means that all JSON schema data types can be easily represented in SQL; the basic data types, number, Boolean, and string are easily translated into SQL equivalents.

On the other hand, hierarchical attributes in NGSI-LD have to be deployed into related tables of SQL, and to create a common index to implement the relationship. Conditional structures (*oneOf*, *anyOf* or *allOf*) have to be expanded also into the set of attributes and allocated, initially in the same output table. Arrays data types can be implemented by many of the current databases, although they are not part of the SQL standard.

Finally, other elements like *pattern* to validate the names of the attributes can hardly be directly translated into SQL. This can be present in the definition of an entity in JSON schema. Having said that a valid JSON payload compliant with a JSON schema using *pattern* clause contains one or several attributes with names and types that can be easily mapped into the equivalent insert instruction into SQL. Table 5.4.1-2 shows an example of this process. Worth to be noted, the names of the attributes should be long IRIs; in this example they are not extended with the context for simplicity reasons.

Table 5.4.1-2: Conversion from NGSI-LD to SQL INSERT statement

JSON schema	{
	"$id": "https://smart-data-
	models.github.io/XXXsubjectXXX/XXXdataModelXXX/schema.json",
	"title": "",
	"type": "object",
	"properties": {
	"id": {
	"type": "string",
	"description": "Property. Unique identifier of the entity"
	},
	"type": {
	"type": "string",
	"description": "Property. NGSI Entity type. It has to be Person",
	"enum": [
	"Person"
	]
	},
	"LastName": {
	"type": "string"
	},
	"FirstName": {
	"type": "string"
	},
	"patternProperties": {
	"Address[1-3]]": {
	"type": "string"
	}
	},
	"City": {

	<pre>"type": "string"
	},
	"Height": {
	"type": "number"
	},
	"OrganizationID": {
	"type": "string"
	}
	},
	"required": [
	"id",
	"type"
	]
	}</pre>

Payload valid against the JSON schema	<pre>{
	"id": "uri: ngsi-ld: Person: 0003",
	"type": "Person",
	"LastName": {
	"type": "Property",
	"value": "mate en salsa"
	},
	"FirstName": {
	"type": "Property",
	"value": "Jacinto"
	},
	"Address1": {
	"type": "Property",
	"value": "Castellana 285"
	},
	"Address2": {
	"type": "Property",
	"value": "Franklinstrasse 13"
	},
	"Address3": {
	"type": "Property",
	"value": "Astorga"
	},
	"City": {
	"type": "Property",
	"value": "Astorga"
	},
	"Height": {
	"type": "Property",
	"value": "1.77"
	},
	"OrganizationID": {
	"type": "Relationship",
	"object": "uri:ngsi-ld:Organization:0001"
	},
	"@contex": [
	"https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.5.jsonld"
	]
	}</pre>

SQL INSERT Statement	INSERT INTO Person (id, LastName, FirstName, Address1, Address2, Address3, City, Height, OrganizationId) VALUES ("uri:ngsi-ld:Person:0003", "mate en salsa", "Jacinto", "Castellana 285", "Franklinstrasse 13", "Times square 1","Astorga", 1.77,"uri:ngsi-ld:Organization:0001");

5.4.2 GQL

Graph Query Language (GQL), defined in ISO/IEC CD 39075 [i.36], is a standard that has the goal to incorporate common foundations from SQL and Property Graph languages and provide an universal Property Graph Query Language that users can depend on to access graph databases, which have become very popular in the last few years.

Being a Property Graph, a direct comparation with NGSI-LD is feasible. They share a lot of features, but there are a couple of differences that are worth to point out.

First of all, NGSI-LD is capable to reference ontologies directly, while GQL does not have a way to do it by itself. For instance, one of the closest implementation of something that can resemble GQL, openCypher [i.37], relies on external libraries to be able to reference external data models. NGSI-LD is capable to reference virtually any type of ontologies.

NGSI-LD is more powerful in representing graph of graphs constructs (e.g. system of systems) than GQL because GQL does not support multilevel composition. In GQL, every node and edge of a subgraph obtains a specific label (or qualifier) to state that they belong to that specific subgraph. This approach has a very big downside that it is the high difficulty to represent relationships between "siblings" subgraphs. This becomes more important when representing digital twins for example, where it is desiderable to represent all kinds of relationships between twins through their components. In NGSI-LD this is achieved using relationships/edges via the special "isNodeOfGraph" and "isSubGraphOf" relationships.

GQL relies heavily on labels. Labels are qualifiers that can be used to create restrictions of nodes and edges. In a way, labels can be seen as the name of tables in SQL. NGSI-LD does not have the concept of labels and the member "type" is not exactly the same as the "label" from GQL because nodes and edges in GQL can also have a "type".

Lastly, GQL supports both directed and undirected edges, meanwhile NGSI-LD supports only directed edges/relationships.

Because of the fact that GQL is a Property Graph, it is always possible to convert data from GQL to NGSI-LD. The mapping is pretty simple:

- Nodes are Entities
- Edges are Relationships:
  - Directed edges can be obtained by simply adding a single Relationship
  - Undirected edges can be obtained by adding two Relationships
- Labels and types of nodes can be mapped as "type" of an Entity
- One label of an edge can be used as the identifier of a Relationship
- Property of nodes can be mapped as Property
- Property of edges can be mapped as Property of a Relationship

The other way around is pretty much the reverse of what just illustrated. The only big incompatibility between NGSI-LD and GQL is the lack of "labels" in NGSI-LD: NGSI-LD types performs both the role of GQL types and GQL labels and it may be not trivial to decide whether an NGSI-LD type should be treated as a GQL label or a GQL type. Plus, edges in GQL can have more than one label, meanwhile in NGSI-LD a Relationship can only be "labelled" with one only label (the identifier). Introducing the concept of "labels" inside NGSI-LD would iron out this problem.

5.5 Federate diverse/heterogeneous semantic models in one system

5.5.1 Challenges when merging semantic graphs with overlapping concepts

Generally, a complete use case could cover many domains of application. For examples, in Smart City use cases, it is possible to find domains such as IoT, Buildings, Vehicles, Organizations, etc. In Smart Aquaculture, it may be possible to find domains such as Fisheries, Water Quality Parameters, Weather.

It is clear that currently very few projects can describe their semantic elements coming from one single source. Also, finding a consistent models or ontologies covering several domains at the same time is a hard task. Thus, merging several semantic sources is the most likely scenario. This leads to potential clashes between the defined terms. Especially for those less specialized (e.g. category, type, etc.).

There are three different types of clashes:

- Concept with the same structure in two different models (less likely)
- Concept with different structures in two different models (more likely)
- Concept from an external model overlapping one another concept from the NGSI-LD core context

While the first one is not a problem at all (unless it is desiderable to use a specific model), the other two clashes are more challenging and deserve to be discussed in dedicated clauses. Basically, there are two ways to solve these types of clashes:

- Ordering the @context array effectively
- Using custom short IRIs or compact IRIs (prefixes)

5.5.2 @context order logic

As hinted in clause 5.3.1, when the @context member is defined as an **array of definitions** the order of the elements is not trivial since NGSI-LD is a sub-set of JSON-LD and it inherits the same order logic: the last elements overwrite the first ones.

For this reason, it is important to order the array **effectively.** There is only one strict rule to keep in mind when doing this task: the last element (the most important) has to always be the NGSI-LD core context. Needs to be noted that, to avoid errors, the current implementations of the context brokers apply the NGSI-LD core context by default and therefore even in the case of not including anything in the @context member it will be applied.

As said, the array should be in **ascending** order. The first elements have to be considered "less important" than the ones that come next. So, basically there are two ways to decide the internal order of this array:

- By importance/frequency
- By specialization

The order logic cannot solve the third challenge expressed in clause 5.5.1, but it can solve the second in almost every situation. There is only one case that is not possible to overcome with this approach. Given 2 different concepts that overlaps in two different models that are going to be used in an NGSI-LD Entity, it is not possible to reference one concept from a model and the other from the second model because one of the two concepts is going to be overwritten depending on the order. In this case (and for the third challenge), the only solution is to use a different short IRI or compact IRI as discussed in clause 5.5.3.

As an example, it is proposed a smart cities use-case that covers, city infrastructure, weather, air quality (IoT) and uses some domain-specific attributes. Following the previous recommendations, its context file may follow the following structure:

```
{
  "@context": {
     "link to the infrastructure context",
     "link to the weather context",
     "link to the IoT context",
     "link to the use-case specific domain context",
     "link to the NGSI-LD core context"
  }
}
```

The most important data model is listed just before the core context to prevent the overwriting of terms during the JSON-LD serialization, while the least specialized data model is placed at the top of the list.

5.5.3 Use compact IRIs and prefixes

There are situations where it may be needed to reference concepts that are normally not possible to reference because they either clash with the core context (e.g. scope) or they cannot be referenced by just changing the order of the elements in the @context member. Long IRIs tackle this issue easily, however it is also true that this notation is not end-user-friendly because shorter attributes names are preferred. Thus, several solutions are possible, none of them perfectly tackling the issue.

Solution 1: rename all short names. For example, what was initially defined as "category" for both building and sensor, now it can be defined by "buildingCategory" and "sensorCategory", while keeping their long IRI (e.g. "buildingCategory": https://smart-data-models/dataModel.Building/category and "sensorCategory": https://smart-data-models/dataModel.Device/category).

This solution maintains an easy readability while keeping the original IRI. However, it does not preserve the original short names. It forces to create specific context for this example and, usually, it is not compatible with other situations.

Solution 2: Rename one of the short names. What initially was defined as "category" for both building and sensor, now it can be defined by category and "sensorCategory", while keeping their long IRI (e.g. "category": https://smart-data-models/dataModel.Building/category and "sensorCategory": https://smart-data-models/dataModel.Device/category.

This solution also maintains an easy readability while keeping the original IRI. However, it does not preserve one of the original short names and it can lead to ambiguity when reading the payloads without being for those who are not aware of the change in one of the short names.

Solution 3: Preceding short names with the acronym (compact IRI) of the namespace the term belongs to that it is expanded into the long IRI. For instance, what initially was defined as "Person" can be defined as "foaf:Person" and "schema:Person", while keeping their long IRI as "http://xmlns.com/foaf/0.1/knows" or "https://schema.org/Person".

This approach tackles the problem by including a prefix to most, if not all, of the properties it provides a complete solution, but it makes slightly more difficult to map with non-linked-data systems as long as the short names have always a prefix.

The first two solutions rely on the creation of a custom @context definition (or file) with the mappings of each short custom IRI with the original long IRI. This file can be created with external services (as the ones defined by the SmartDataModels Program).

Instead, solution 3 does not always results in the creation of a custom @context definition because (as discussed in clause 5.3.1) it is possible to rely on the @context files defined by the maintainers of the data models.

5.6 System of Systems and graph representation

System of systems (e.g. Digital Twins) can be represented in NGSI-LD using two different types of composition:

- A top-down system composition, using the special Relationships "hasPart" and "hasDirectPart"
- A bottom-up system composition, using the special type "Graph" and Relationships "isNodeofGraph" and "isSubGraphOf"

It clearly depends on the use-case, but it is not uncommon to use both these approaches to represent systems of systems. While it may be clear how to map a complex system in NGSI-LD (either using "hasPart" or sub-graphs), it may be unclear what are the benefits to link external data models as well.

The main reason behind the decision to use external data models with system of systems representations is to take advantage of the knowledge and the structure provided by the external data models referenced. Usually, systems of systems are asked to reference concepts from many different domains so external data models can help to identify if there are Entities with missing/invalid Properties and they can also help to assess whether an Entity can be linked with a different one or not.

Plus, when adopting the bottom-up system composition method, it is possible to take advantage of the multi-typing system to add a domain-specific reference. Inside the specification of NGSI-LD [i.26], the "Graph" type and the Relationships "isNodeOfGraph" and "isSubGraphOf" are not mandatory when creating a graph, but they are a convention, the suggested way to do it. So, Entities that have the "Graph" type are like all the other Entities: they can have more than one type and Attributes. Hence it is possible to use the multi-typing system to reference a concept of an external data model as described in clause 5.3.2 and it is also possible to add metadata information about the graph by adding Attributes (either Properties and/or Relationships). For instance, in an e-health scenario a clinic may be mapped as a graph of graphs where each room is a graph. For all the "Graph" Entities representing the rooms, it would make sense to add a specific type such as the "BuildingSpace" of SAREF4BLDG (SAREF for Building) and/or to add few attributes such as a Property that contains the number of the room or the date when the room has been renewed. As discussed in clause 5.3.2, this allows to perform sortof semantic queries when trying to retrieve Graphs.

Generally, there are not many differences when linking external data models when working with graphs, anyway, when modelling a complicated system in NGSI-LD as a graph or a graph of graphs, usually the reason to follow this path is to take advantage of the graph structure to perform queries: it is possible to query all the entities of a graph with a single query, which makes it very convenient. Furthermore, if actuation (which is the capability to change the behaviour of a real object by changing the related Entity inside the context broker) is needed, then, depending on which actuation workflow method it is adopted, it may be needed to rely on NGSI-LD Subscriptions/Notifications. If the client wants to receive short IRIs inside the results coming from a query or a notification, the main problem is the one discussed in clause 5.3.4 regarding the choice of the correct @context file, because it is more likely to incur in a situation where Entities that map concepts from different domains are linked together in the same graph (hence retrieved by the same query/Subscription).

5.7 Suggestions and Conclusions

To summarize all the recommendations given in clause 5, it follows a list with all of them:

SUGGESTION-01	Many different use-cases can take advantage of any external data model, but it is important to note that the structure of the NGSI-LD Entities may differ depending on which data model users try to link.
SUGGESTION-02	When the application can interact directly with the context broker, it should always take advantage of the NGSI-LD Query system to get any additional information out of Relationships.
SUGGESTION-03	Even though it would be *possible* to omit the @context member since Context Brokers enforce the core context in every Entity (hence, they apply fake long IRIs to every term that has not a mapping in the @context), if the goal is to reference external data model, the **most effective** way to do it is to provide a custom @context.
SUGGESTION-04	When deciding how to build a custom @context, users should rely on arrays of definition because they are more powerful, and they allow to overcome common problems (as for overlapping concepts).
SUGGESTION-05	Considering that the order inside an array of definitions matters, users should order all the external data models by importance in increasing order (least important to most important).
SUGGESTION-06	If it is not suitable to directly reference a JSON-LD @context file, when deciding which key to choose for a prefix, it is usually better to use the same key used in the JSON-LD @context file to avoid ambiguity.
SUGGESTION-07	If compact IRIs are needed to reference a concept, they should be structured to avoid ambiguity when used with external data models that rely on compact IRIs (as the SmartDataModels). Not only the chosen IRI should be meaningful, but it is preferable to make it unique within the @context definition to avoid: (1) overwriting other concept definitions with the same compact IRI and (2) being overwritten by subsequential concept definitions with the same compact IRI.

SUGGESTION-08	When using multi-typing, if data validation is an important feature for the application, users need to overcome every overlapping concept providing direct definitions (clause 5.3.1) or adjusting the @context accordingly (clauses 5.5.2 and 5.5.3).
SUGGESTION-09	Explicit is better than implicit. Linking external data models to define Attributes removes the ambiguity of the terms used within an NGSI-LD Entity.
SUGGESTION-10	Users need to define at least one custom JSON-LD @context file because it is the only way to apply the correct @context when interacting with the APIs (create an NGSI-LD Entity, queries, create NGSI-LD subscriptions, etc.).
SUGGESTION-11	Since every Indirect Definition brings the serializer of the context broker to download an additional JSON-LD @context file, it is suggested to avoid using many Indirect Definition (especially nested definitions) to speed-up operations with the APIs.
SUGGESTION-12	When the references are split in many @context files, it may be problematic to keep short IRIs when querying Entities since they may have been created using several JSON-LD @context files. In this case, it is either suggested to provide the URL of a custom JSON-LD @context file which merges the needed references or to let the application apply the custom @context file after retrieving the Entities.

As showed in this clause, external data models can and should be used in NGSI-LD. In general, this is relatively easy to achieve because NGSI-LD is a subset of JSON-LD, which is capable to reference any type of external data models. Considering its Property Graph definition, NGSI-LD proves to be very powerful in representing structured data and real systems. It is true that NGSI-LD users may skip on linking external data models, but as discussed it does not pay off: there are different use-cases where the usage of external data models is the "ace in the hole", something that may help in producing better applications for very complicated domains (smart-cities, e-health, etc.) capable both to show the links between instances and to clarify the semantic meaning and even the expected structure of every concept. There is room for improvements still. For instance, Context Brokers may be empowered with some (even basic) reasoners-like capabilities or they may be allowed to store schemas to use to validate the input, or to give more information about an Entity. Further discussions on what may be possible to address are introduced in clause 6.

6 Possible additions to NGSI-LD specification

6.1 What can be achieved in centralized vs. federated deployments?

NGSI-LD deployments comes in two ways: centralized and federated. Depending on the use-case it may be enough to rely on a centralized deployment, but there are cases where this is either not feasible or realistic. For instance, when operating with Smart Cities there may be a lot of different data sources and it may be possible that the same Entity contains information from 2 or more data sources. Even though usually the adoption of a federated deployments implies to design the whole architecture in a certain way, it is not needed to perform any particular operation when using external data models because of how Entities are stored. Inside Context Brokers, any Entity is stored using long IRIs. This happens because Context Brokers do not store the custom @context file/members when provided. If they did it, it would have been a problem with federated deployments because the information about the @context may be split in different nodes. In summary, *it is not needed* to change anything inside the NGSI-LD specification.

6.2 Representation of a schema

NGSI-LD is intended to work schema-less. Anyway, there are use cases where the presence of a schema may be desired. NGSI-LD is used to represent "closed-world" systems that may be already partially captured by a schema to either describe the system or check/enforce consistency and integrity constraints. Schemas would allow to manage complex systems (as system of systems and digital twins) avoiding classical problems that schema-less models have (inconsistency, redundancy of information, etc.). For these reasons, it may be convenient to allow context brokers to perform input validation against a schema. If the context broker could perform input validation, it would be possible to allow or deny the creation/update of any Entities/Attributes if the new changes do not meet the requirements. To be able to do it, context brokers should be enabled to represent the schema in some way.

One possible solution could be to represent the schema using graph homomorphisms.

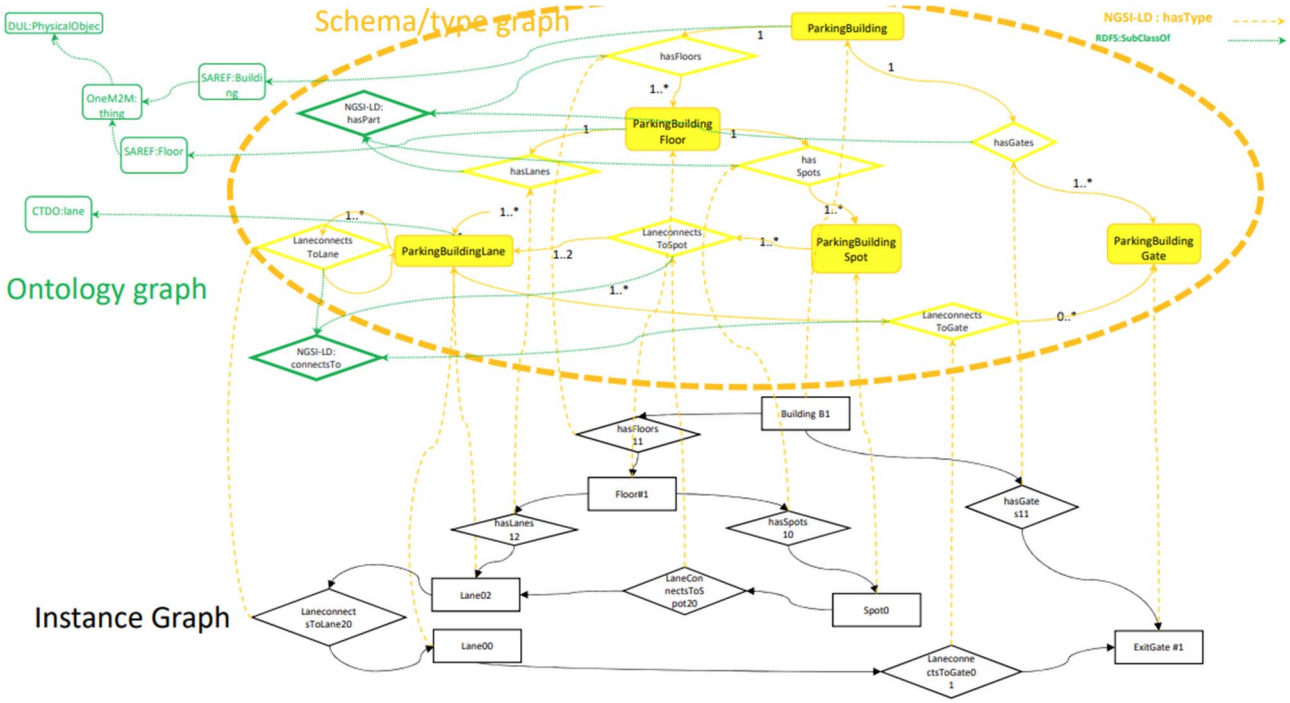

Figure 6.2-1: Schema representation using graphs homomorphisms

Figure 6.2-1 shows three different graphs: the RDF/Ontology graph (green elements), the schema graph (yellow elements) and the instance graph (black elements). The RDF/Ontology graph is outside of the context broker, so it is only referenced inside the Entities and Attributes of the schema graph. The schema graph is the representation of every element that exists in the domain of the application. Entities and Attributes represented in this graph may reference external data models or the NGSI-LD cross-domain ontology and may be called "Prototypes". Finally, the instance graph is the representation of instances. Elements of this graph may reference external data models or the NGSI-LD cross-domain ontology as well, but they also have a special Relationship called "NGSI-LD:hasType" that point to an element of the schema graph. Through this Relationship is it possible to state that an Entity/Relationship of the instance graph has to match the Entity/Relationship of the schema graph. Basically, the elements inside the schema graph can be used to perform input validation against the elements of the instance graph. The meta-relationship "NGSI-LD:hasType" should be defined in the meta model as the following:

NGSI-LD:hasType

- hasType rdf:type owl:ObjectProperty
- rdfs:domain [owl:unionOf (NGSI-LD:Entity, NGSI-LD:Relationship)]
- rdfs:range [rdf:type owl:class: owl:unionOf (NGSI-LD:Entity, NGSI-LD:Relationship)]

The schema graph can be provided by the users using the same API calls to create/update Entities and Attributes. Context brokers have to be able to understand if an Entity belongs to the schema graph. There could be two ways to do it without changing the API:

- Entities of the schema graph have to have the special type as "NGSI-LD:Schema-Entity"; or
- Context Brokers may create a special Graph Entity called "Schema". Every Entity of the schema graph has to have the special Relationship "isNodeOfGraph" pointing to the "Schema" Entity;Having a schema, would allow to perform input validation within the context broker (to a certain degree). It could work with different modalities: for instance, it could have a strict mode that would also deny to add Attributes inside an Entity that do not figure in the Prototype, or a "slack" mode that would allow it.

Finally, it would be also possible to perform specific queries that relies on the schema as "Return all the Entities of type X that have (or have not) the same structure (or the same number of Attributes) with the Prototype Y". These queries can already be performed with the current API **statically**, but with a schema they could change **dynamically** by simply changing the Prototypes in the schema graph.

6.3 Range for values of literals (integers, dates, enums)

In clause 6.2, the validation was intended as a matching between the types of the domain (an Entity or an Attribute) and the range (the type of the value or the object member) and/or to check if an Entity is missing any Attributes, but there are many more types of input validation.

An important one, considering digital twins/IoT systems, is the possibility to express a range for the accepted values of literals and it is not uncommon to see that the value of a concept (e.g. owl:DataProperty) is restricted in some way: to give an example, a valve may have a range of temperature expressed in numbers that are considered "safe" values. Exceeding this range may be very dangerous.

Data models provides a way to express these restrictions. For example, in owl it is possible to use DatatypeRestrictions to define constraints on literal values. It is possible to give the maximum integer allowed or to express the maximum length of a string.

Context brokers may take advantage of these constraints to perform input validation and return more detailed information about errors. In this sense, the types of literals ranges may be:

- Forbidden values
- Safe-to-work values

The former can be used to disallow invalid operations (e.g. "the ZIP code has to have 5 digits"), meanwhile the latter can be used to Fire up NGSI-LD Notifications (to subscribed users) to notify once these constraints are being violated (e.g. a dangerous value for the temperature of a valve).

The context broker may take advantage of the schema representation described in clause 6.2 to understand if there are constraints for literal values (e.g. DatatypeRestrictions). Anyway, one another approach could be to introduce, inside the cross-domain ontology, special sub-Properties that can express these constraints. The first approach is more preferable because it is more efficient since data models are capable to express these constraints, meanwhile the second approach would allow to permit it even without having a schema or an external data model.

6.4 Query Language improvements

6.4.1 Relationships and navigation in the NGSI-LD Property graph

One feature that would greatly improve the Query Language would be to be able to navigate through Relationships to return a graph of Entities. For instance, in NGSI-LD it is not possible to make a query as "return all the Persons that know a Person that knows X (being X a specific Person)". The result of this query would be a graph of Entities (or a list of graphs of Entities). This may be achieved by allowing Context Brokers to navigate Relationships when querying and introducing inside the Query Language a specific set of rules to represent the graph queries.

As one last observation, something may be changed with Relationships as well to improve the Query Language. In NGSI-LD, Relationships are considered first-class citizens, yet it is not possible to query for Relationships only, or to reference a specific instance of a Relationship. This last point could change if every Relationship would have an ID as edges do in other Property Graphs implementations (as usually is the case with databases supporting openCypher queries). Having IDs for Relationships, it would be possible to perform the following queries:

- Query for a specific instance of a Relationship (check existence)
- Query for the Entity pointed by a specific instance of a Relationship
- Navigate the Property Graph when querying, being able to illustrate the path (composed of Entities and Relationships)

The IDs may be auto generated.

6.4.2 Query sub-Attributes values without giving the identifiers of the parent Attributes

Using the dot notation, it is possible to query for sub-Attributes by listing all the parent Attributes and then adding the condition to check depending on the query (equal, not equal, etc). In NGSI-LD it is not possible to query sub-Attributes without providing the parent Attributes. There are use-cases where this would be useful. For instance, in clause 5.3.5 it was discussed how to represent semantic links to perform semantic queries. In NGSI-LD it is not possible to perform these queries:

- "Return all the Entities that have an Attribute that is equivalent to '*saref:Temperature*'"
- "Return all the Entities that have an Attribute that has been measured by urn:ngsi-ld:sensor:123" (relying on the "measuredBy" relationship)

Both these queries cannot be performed because it is **supposed** that the application knows and provides the name of every parent Attributes that contain the target sub-Attribute, which may be not possible to know before-hand nor important (because the main point would be to return the Entity, not to match a specific pattern of Attributes).

Considering optimization and/or performance, this new modality may be optional by adding a specific parameter when needed.

6.4.3 Add the missing Topological Functions

In clause 4.10 of ETSI GS CIM 009 [i.26] the NGSI-LD Geoquery Language is defined. As stated in that clause, the georel are defined as in OGC 06-103r4 [i.34]. Although most of them have been introduced in the Geoquery Language, two of them are missing: **touches** and **crosses**. Adding them would allow to perform specific geo-spatial queries that are not possible to be represented by the other georels as:

- "Return the Park that *touches* (which means, only one side of the boundaries overlap) a Street"

6.4.4 Return enriched Entities with computed data

As discussed in clause 4.6.3, in GeoSPARQL it is possible to return computed values as the distance between two geospatial objects when performing queries. Context Brokers are already tasked to compute the distance or the area of geospatial objects (since they can perform GeoQueries), so they would be able to return Entities enriched with computed data, in a way that reminds what happens with the Aggregated Representation.

If explicitly requested, GeoQueries may return the distance between two Entities (if the georel is "near") or the area of intersection (if the georel is "intersects") as a sub-Property of the target GeoProperty. This Property may have a structure similar to the following:

```
"location": {
   "type": "GeoProperty",
   "value": { … },
   "distance": {
     "type": "Property",
     "value": "100",
     "unitCode": "meters",
     "referenceGeometry" {
        "type": "GeoProperty",
        "value": { … }
     }
  }
}
```

Users should be allowed to take advantage of this when using Subscriptions as well. In this case, Entities inside the NGSI-LD Notification would be enriched with computed values.

Other types of computed data that may be returned may be:

- The area of a Polygon
- Total area of intersection between two Polygons
- Point of intersection between a Polygon and a Line

6.4.5 Introduce the orderBy clause

In NGSI-LD is actually not possible to directly use the orderBy clause when querying, with few exceptions as Temporal Queries (lastN parameter). This is a big limitation of Queries, because it is up to the application to reorder the results.

Being able to use an orderBy clause would allow to perform interesting queries as:

- "Find Buildings near (<1 000 m) to the reference geometry ordered by distance in descending order"
- "Find 10 Cars ordered by speed in ascending order" [with limit=10]

While the first query may still be performed if the application reorders the entities retrieved, the second query is absolutely impossible to perform because of the limit constraint: in fact, any Context Broker would return 10 Cars in any order.

6.5 Add the special Property "label"

In clause 5.4.2, it was reported that NGSI-LD does not fully support GQL because of the lack of labels. The missing of labels impacts directly the capability to convert data from GQL to NGSI-LD and (especially) from NGSI-LD to GQL. This could be a problem if implementations of NGSI-LD would be used on top of GQL to take advantage of the rich query mechanism that GQL provides.

This incompatibility would be solved by adding the concept of "label" inside Entities and Relationships. A "label" should be a non-reified special member (for Entities) or sub-member (for Relationships). Its value should be either a string or an array of strings. It is not a type definition, so it is not needed to expand the value.

Labels can be used to query as it happens for types, with the only difference that they are not influenced by the context expansion (so it is never needed to provide a custom @context as sometimes happens with types).

6.6 Forbid scoped and local @context definitions

In JSON-LD it is possible to define very complex @context members that are capable to change the structure of JSON-LD documents when applied. There are actually two particular features that allow to redefine JSON-LD documents during the serialization with the @context members:

- Scoped @context
- Local @context

Scoped @context definitions are nested definitions of @context inside the main @context member at the top level of the JSON-LD document. They allow to change the identifier or to completely redefine the structure of terms in the JSON-LD document.

Local @context definitions are @context definitions that figure internally another term of the JSON-LD document. They have a higher priority than the main @context member that figures in the top level, so they potentially can be used to redefine terms of the NGSI-LD core context (e.g. renaming "value" as "scope").

At the current time, the API does not explicitly forbid the usage of these features. It is true that it is not possible to have local @context members inside Attributes, but implementations may still overlook this problem since there is not an explicit warning regarding it.

In this regard, considering the dangerous behaviour that could be permitted by allowing them, it is suggested to explicitly forbid the usage of these features inside the body of an NGSI-LD Entity (local @context) and inside the @context member at the top level of an NGSI-LD Entity (scoped @context). Implementations of Context Brokers have to discard any NGSI-LD payload (Entities, Attributes, Subscriptions, etc.) that contains scoped @context or local @context definitions.

Annex A:
Change History

Date	Version	Information about changes
2022/02/28	0.0.1	Early draft of the document
2022/05/31	0.0.2	Stable draft of the document
2022/07/04	0.1.0	Final draft of the document
2022/07/21	1.1.1	Clean-up before publishing

History

Document history		
V1.1.1	August 2022	Publication