Project Acronym:	**VICINITY**
Project Full Title:	**Open virtual neighbourhood network to connect intelligent buildings and smart objects**
Grant Agreement:	**688467**
Project Duration:	**48 months (01/01/2016 - 31/12/2019)**

Deliverable D2.2

Detailed Specification of the Semantic Model

Work Package:	WP2 – Standardization Analysis and VICINITY platform conformity
Task(s):	T2.2 – Definition of VICINITY semantic model as extension of existing IoT ontologies
Lead Beneficiary:	UPM
Due Date:	31 August 2017 (M20)
Submission Date:	31 August 2017 (M20)
Deliverable Status:	First version
Deliverable Type:	R
Dissemination Level:	PU
File Name:	**VICINITY_D2.2_VICINITYSemanticModel_v1.0.pdf**

European Commission | Horizon 2020 European Union funding for Research & Innovation

IoT European Platforms Initiative

VICINITY Consortium

No	Beneficiary		Country
1.	TU Kaiserslautern (Coordinator)	UNIKL	Germany
2.	ATOS SPAIN SA	ATOS	Spain
3.	Centre for Research and Technology Hellas	CERTH	Greece
4.	Aalborg University	AAU	Denmark
5.	GORENJE GOSPODINJSKI APARATI D.D.	GRN	Slovenia
6.	Hellenic Telecommunications Organization S.A.	OTE	Greece
7.	bAvenir s.r.o.	BVR	Slovakia
8.	Climate Associates Ltd	CAL	United Kingdom
9.	InterSoft A.S.	IS	Slovakia
10.	Universidad Politécnica de Madrid	UPM	Spain
11.	Gnomon Informatics S.A.	GNOMON	Greece
12.	Tiny Mesh AS	TINYM	Norway
13.	HAFENSTROM AS	HITS	Norway
14.	Enercoutim – Associação Empresarial de Energia Solar de Alcoutim	ENERC	Portugal
15.	Municipality of Pylaia-Hortiatis	MPH	Greece

Authors List

Leading Author (Editor)			
Surname	**First Name**	**Beneficiary**	**Contact email**
García-Castro	Raúl	UPM	rgarcia@fi.upm.es

Co-authors (in alphabetic order)				
No	**Surname**	**First Name**	**Beneficiary**	**Contact email**
1.	Fernández-Izquierdo	Alba	UPM	albafernandez@fi.upm.es
2.	Heinz	Christopher	UNIKL	heinz@cs.uni-kl.de
3.	Kostelnik	Peter	IS	peter.kostelnik@intersoft.sk
4.	Poveda-Villalón	María	UPM	mpoveda@fi.upm.es
5.	Serena	Fernando	UPM	fserena@fi.upm.es

Reviewers List

List of Reviewers (in alphabetic order)				
No	**Surname**	**First Name**	**Beneficiary**	**Contact email**
1.	Oravec	Viktor	BVR	viktor.oravec@bavenir.eu
2.	Uwiringiyimana	Marie Madeleine	UNIKL	uwiringi@cs.uni-kl.de
3.	Mach	Marian	IS	Marian.Mach@tuke.sk

Revision Control

Version	Date	Status	Modifications made by
0.1	22. 02.2017 (M13)	Initial Draft	María Poveda-Villalón (UPM)
0.2	03. 07.2017 (M19)	First Draft formatted with contributions received	María Poveda-Villalón (UPM)
0.3	19. 07.2017 (M19)	Deliverable version for final review by partners	María Poveda-Villalón (UPM), Raúl García-Castro (UPM), Alba Fernández-Izquierdo (UPM), Fernando Serena (UPM)
0.4	20. 07.2017 (M19)	Added chapter 2	Christopher Heinz (UNIKL)
0.5	25. 07.2017 (M19)	Deliverable version uploaded for Quality Check	María Poveda-Villalón (UPM), Raúl García-Castro (UPM)
0.6	21. 08.2017 (M20)	General update according to QAR	María Poveda-Villalón (UPM), Alba Fernández-Izquierdo (UPM)
0.7	22. 08.2017 (M20)	Update section 7 according to QAR	Peter Kostelnik (IS)
0.8	23. 08.2017 (M20)	Update section 2 according to QAR	Christopher Heinz (UNIKL)
0.9	24. 08.2017 (M20)	Quality Check (changes integration)	María Poveda-Villalón (UPM)
0.10	25. 08.2017 (M20)	Final Draft reviewed	María Poveda-Villalón (UPM), Viktor Oravec (BVR), Marie Madeleine Uwiringiyimana (UNIKL), Marian Mach (IS)
1.0	29. 08.2017 (M20)	Submission to the EC	Carna Radojicic (UNIKL)

Table of Contents

Table of Tables

Table of Figures

Table of Listings

List of Definitions and Abbreviations

Abbreviation	Definition
API	Application Programming Interface
DUL	DOLCE (Descriptive Ontology for Linguistic and Cognitive Engineering) Ultra Light
ETSI	European Telecommunications Standards Institute
IG	Interest Group
IoT	Internet of Things
IRE	Identifier, Resource and Entity
OGC	Open Geospatial Consortium
ORSD	Ontology Requirements Specification Document
OWL	Web Ontology Language
RDF	Resource Description Framework
SAREF	Smart Appliances REFerence
SOSA	Sensor, Observation, Sample, and Actuator
SPARQL	SPARQL Protocol and RDF Query Language (recursive acronym)
SSN	Semantic Sensor Network
WG	Working Group
WoT	Web of Things
W3C	World Wide Web Consortium

1. Executive summary

The present document is the deliverable "D2.2. - Detailed Specification of the Semantic Model" of the VICINITY [18] project, funded by the European Commission's Directorate-General for Research and Innovation (DG RTD), under its Horizon 2020 Research and Innovation Programme (H2020).

This deliverable gives an overview and documentation of the VICINITY ontology network as of August 2017. The document covers, among others, the following main topics:

- **Methodological guidelines** and recommendable **infrastructure** to be used during the ontology development process.
- The VICINITY ontology **network** is presented in this deliverable including the description of the modules identified as a need, and therefore developed, so far:
    - **Core ontology**: this module is developed in order to represent the information needed to exchange IoT descriptor data between peers through the VICINITY platform.
    - **WoT ontology**: this module aims at representing how and where things can be discovered or accessed in the Web of Things environment.
    - **Wot Mappings ontology**: this module has been developed within VICINITY in order to support the automatic data lifting process as described in "D3.3. Open Interoperability Gateway API, first version"-. Briefly, mappings explicitly define implicit semantics of structured data exchanged by peers in VICINITY. Moreover, the Mappings ontology supports the description of mappings for data that may come from external *web things*, and ultimately, Web resources made available through heterogeneous API endpoints.
- **Examples** of use of the above-mentioned modules.

In addition, the document provides an overview of the main existing standard ontologies that cover similar aspects of the VICINITY ontology and includes some conclusions and future lines of work.

2. Introduction

The VICINITY interoperability approach relies on ontologies (i.e., semantic data models) that will be exploited throughout the VICINITY infrastructure. In computer science, ontologies are defined as "formal, explicit specifications of a shared conceptualization" [16]. The VICINITY ontologies will be formal in the sense of following Description Logics and being implemented in the W3C Web Ontology Language standard OWL.[1] The conceptualization to be shared among the VICINITY components and plugged systems will cover different domains of interest ranging from horizontal domains like time and space to specific definitions needed within the VICINITY ecosystem. For this reason, the VICINITY approach is based on a modular ontology network in which existing standard ontologies will be reused whenever possible. This document will describe the process followed to build such ontologies and the resulting models in detail.

The goal of this deliverable is to detail the VICINITY ontologies developed for the release 0.1 of the VICINITY platform. This decision has been made by the VICINITY consortium during the M13 plenary meeting. That is, the ontology to be delivered in M20 should cover the needs for the platform release 0.1. This does not mean that the ontology is version 0.1 (see "D3.3. Open Interoperability Gateway API, first version") as it will be later described. The scope of this deliverable is not limited to the description of the resulting ontologies; the processes followed and the infrastructure deployed during the ontology development are also explain. In addition, some standard ontologies are reviewed and examples of how to use the developed VICINITY ontology are provided.

The rest of this deliverable is structured as follows:

- **Section 3** provides an overview of main standard ontologies related to the scope of the VICINITY ontology network.
- **Section 4** is devoted to the methodological guidelines followed during the ontology development.
- **Section 5** describes the infrastructure deployed during the ontology development process.
- **Section 6** is dedicated to the description of the VICINITY ontology network and each of the modules developed, namely the WoT ontology (Section 6.1), the Core ontology (Section 6.2), and the WoT Mappings ontology (Section 6.3).
- **Section 7** includes a number of examples about how to use and instantiate the different modules of the VICINITY network.
- **Section 8** provides some conclusions and future lines of work.

For the sake of understandability of this document, readers not familiar with the concept of ontologies in computer science and the web might find the basis for this topic in the "Ontology Development 101: A Guide to Creating Your First Ontology".[2] In addition, the VICINITY deliverables

[1] https://www.w3.org/TR/owl-ref/
[2] http://www.ksl.stanford.edu/people/dlm/papers/ontology101/ontology101-noy-mcguinness.html

"D1.5. VICINITY technical requirements specification" and "D3.3. Open Interoperability Gateway API, first version" can provide background knowledge to readers about the project and the relationship between deliverables and outputs.

3. Relevant standard ontologies for VICINITY

VICINITY aims to support IoT interoperability by developing a generic ontology, based on existing standards (from W3C, ETSI, oneM2M, etc.), that can be used to interchange IoT data across the different components from the VICINITY platform and the different IoT infrastructures used in the project.

In line with what has been evaluated in "D2.1. "Analysis of Standardisation Context and Recommendations for Standards Involvement" – so far, this chapter will feature an overview of the standard ontologies that are most relevant to VICINITY; i.e., this document will only present ontologies that have been developed in cooperation with standardization bodies.

Well aware that the following will not capture all the available ontologies that have been developed for the IoT so far, and that may be of interest to VICINITY when new ontology requirements arrive, it is advisable to point interested readers to the LOV4IoT ontology catalogue[3] for a more detailed overview on ongoing activities in this field.

3.1. W3C Semantic Sensor Network Ontology

The Semantic Sensor Network ontology (also known as "SSN") was developed by the W3C Semantic Sensor Network Incubator Group[4] (SSN-XG). The purpose of this OWL ontology is to describe sensors, their capabilities and properties, the act of sensing itself and the resulting observations of the physical world. The SSN-XG was working from March 2009 to June 2011.

It was agreed upon to build an ontology compatible, but not constrained by OGC standards, such as SensorML and Observations & Measurements (O&M). Furthermore, concepts are defined in a modular architecture and as broad as possible, so that specific interpretations could later be defined where necessary. Possible applications range from satellite imagery or large scale scientific monitoring to the Web of Things[5] (WoT). Figure 1 shows the resulting SSN ontology [3]. The SSN ontology is conceptually organized into ten modules, only covering concepts and relations relevant to sensors, leaving concepts related to other, or multiple, domains to be included from other ontologies when the ontology is used. Doing so makes the ontology modular and reusable. Overall, it consists of 41 concepts and 39 object properties. Since DOLCE-UltraLite (DUL [6]) as a lightweight ontology was chosen as the upper ontology, SSN inherits 11 concepts and 14 object properties from DUL.

[3] http://sensormeasurement.appspot.com/?p=ontologies
[4] https://www.w3.org/2005/Incubator/ssn/XGR-ssn/
[5] https://www.w3.org/WoT/
[6] http://ontologydesignpatterns.org/wiki/Ontology:DOLCE+DnS_Ultralite. For the ontology code see: http://www.ontologydesignpatterns.org/ont/dul/DUL.owl

The SSN modules are represented by boxes in Figure 1. Classes or concepts are represented by rounded boxes. Hierarchical relations are represented with plain arrows in which the origin of the arrow represent the more specific concept and the ending the more general one. Finally, dashed arrows represent axiom properties defined between classes.

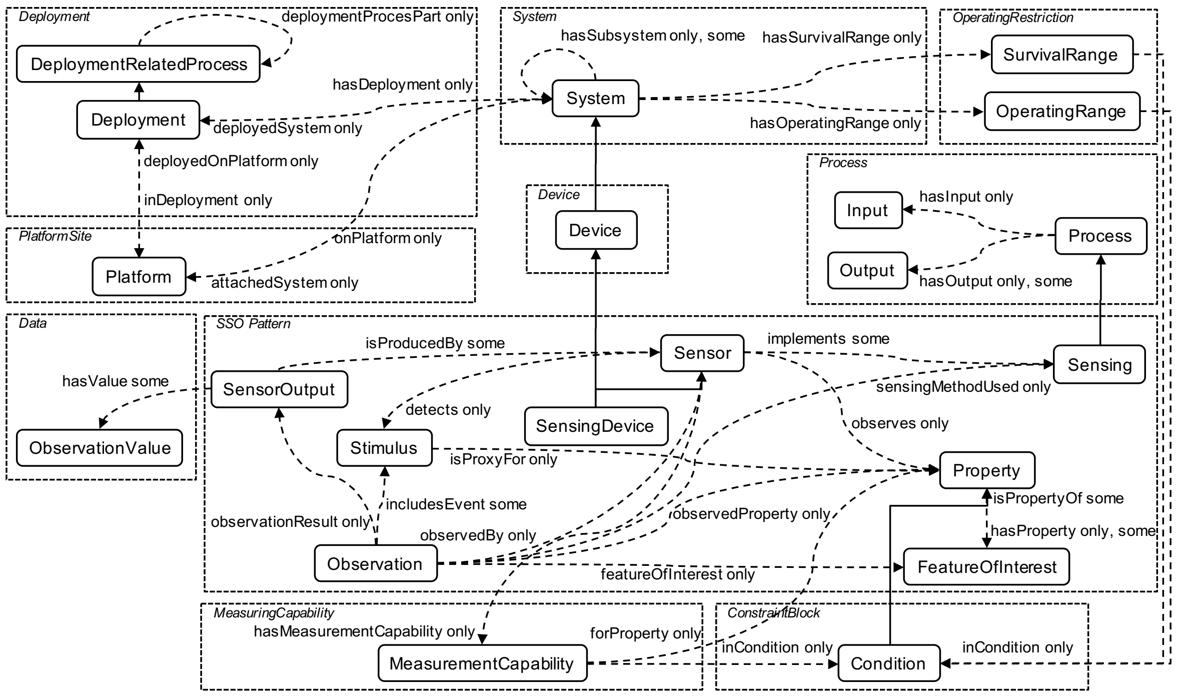


Figure 1. Semantic Sensor Network (SSN) Ontology (Figure extracted from [3]).

Not only is SSN able to describe sensors and their capabilities, such as accuracy, the methods used for sensing, operating and survival ranges. It also offers means to describe the field deployment of a sensor, including, e.g., deployment lifetime or sensing purpose. Figure 2 shows these concepts in more detail.

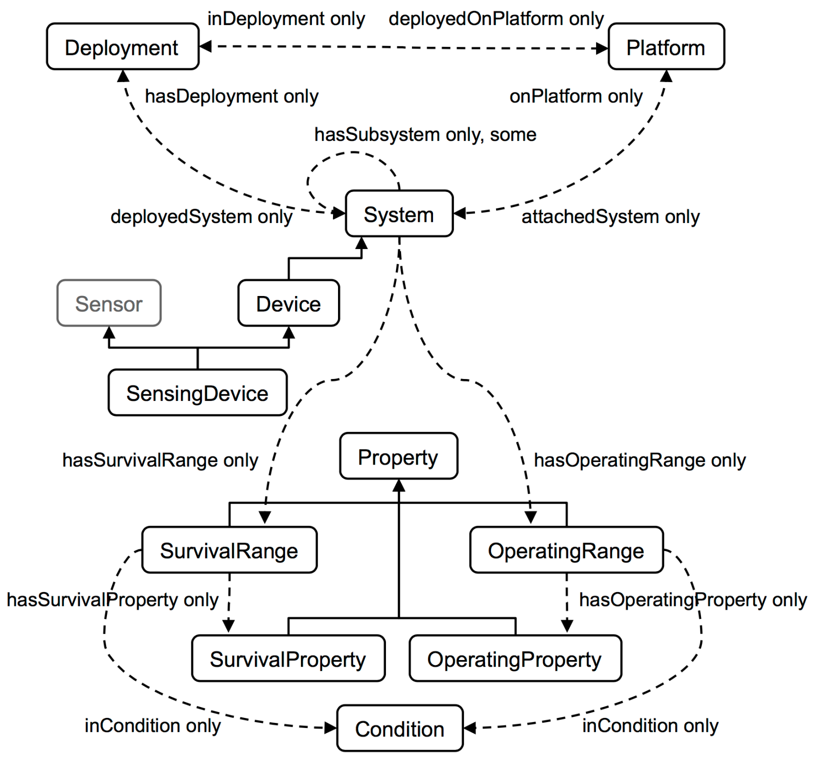


Figure 2. Deployment, Platforms, System and Properties in SSN (Figure extracted from [3]).

Modelling of concepts such as units of measurement, locations, hierarchies of sensor types, and feature and property hierarchies is out of scope of the SSN ontology. The intention was to create core sensor description ontology, which can be easily extended with specific domain concepts.

The current version of the VICINITY ontology uses the SSN ontology to represent properties (`ssn:Property`) and features of interest (`ssn:FeatureOfInterest`). The SSN ontology is also used to represent device characteristics by means of specific types of properties such as capabilities (`ssn:Capabitliy`) and measurement properties (`ssn:MeasurementProperty`, `ssn:Frequency`, etc.).

In December 2014, a new Working Group started in the W3C, the Spatial Data on the Web WG[7], which had as one of its goals to further develop the SSN ontology and convert it into a formal recommendation (i.e., standard). During the development, the Working Group has addressed different updates on the ontology: changes in the scope and audience of the ontology (switching from ontology engineers to a broader scope of web developers, resource-constrained IoT devices,

[7] https://www.w3.org/2015/spatial/

etc.), updates in the ontology to solve different shortcomings identified since its first release, use of new technical developments to supports lightweight semantics.

One of the major changes compared to the previous version of the ontology has been the modularization of the ontology, which now is composed of different separated modules, and the alignment of such modules with other existing ontologies, as can be seen in Figure 3. During this modularization activity, the core parts of the SSN ontology have been separated into an independent module, the Sensor, Observation, Sample, and Actuator (SOSA) ontology, that allows representing the main entities, relations, and activities involved in sensing, sampling, and actuation using lightweight semantics.

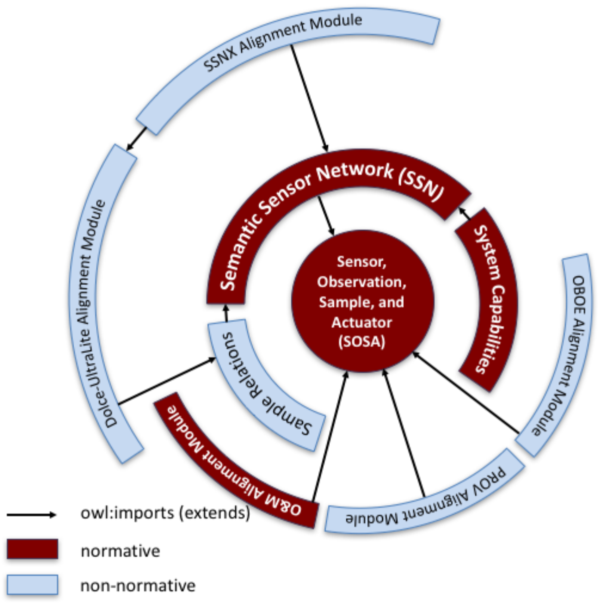


Figure 3. The SOSA and SSN ontologies and their vertical and horizontal modules (Figure extracted from [13]).

The new version of the SSN ontology is currently not being used in the VICINITY ontology, mainly due to the need of reusing a stable version of the ontology in the VICINITY development. By the beginning of July 2017, the SSN ontology has been published as a Candidate Recommendation at the W3C[8] [13] and it is expected to become a Recommendation by the end of September 2017. Therefore, during the second half of 2017 one of the tasks to be performed over the VICINITY ontology is to update it to the second version of SSN to be conformant with the latest standard. No problem in this task is envisioned since the VICINITY partner *Universidad Politécnica de Madrid* (UPM) has been actively working in the development of the (old and) new SSN ontology and the VICINITY ontology is already ready for the migration.

[8] https://www.w3.org/TR/2017/CR-vocab-ssn-20170711/

3.2. ETSI Smart Appliances REFerence ontology

In November 2015, the first version of the Smart Appliances REFerence ontology (SAREF) standard for smart appliances was published by ETSI TC SmartM2M [5] after an initial standardization initiative launched by the European Commission DG CONNECT in collaboration with ETSI TC SmartM2M. This standard subsequently evolved into a new version published in March 2017 [6] and currently includes also three Technical Specifications that extend the SAREF ontology to three different domains, namely energy (SAREF4ENER [7]), environment (SAREF4ENVI [8]), and buildings (SAREF4BLDG [9]). The current and envisioned modules of the SAREF ontology can be seen in Figure 4.

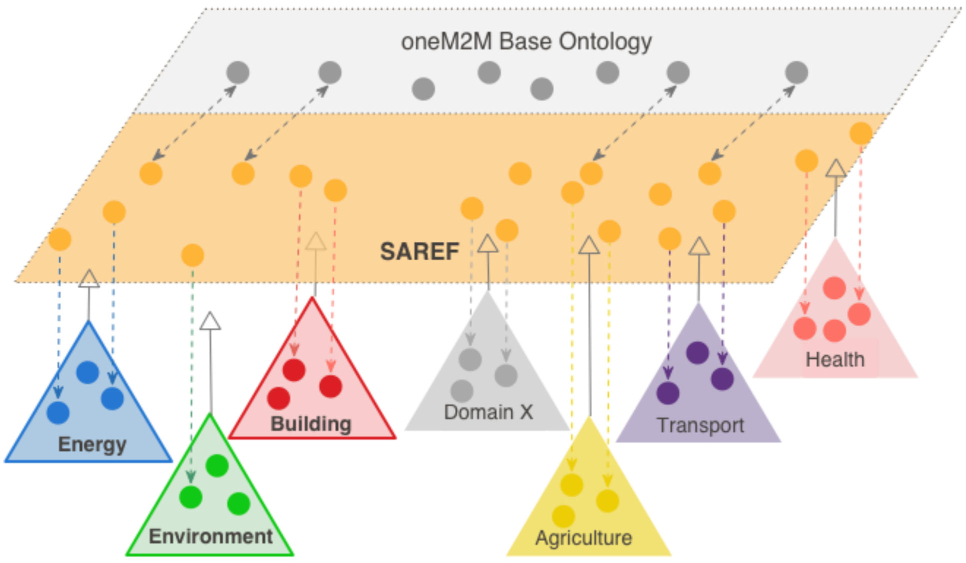


Figure 4. The current and envisioned modules of the SAREF ontology (Figure extracted from [4]).

The starting point of SAREF is the concept of Device; a Device is “a tangible object designed to accomplish a particular function in households, common public buildings or offices” [6]. Devices are for example a light switch, a temperature sensor or a washing machine. Figure 5 shows an overview of main classes and properties defined in the SAREF ontology. In such figure, concepts are represented by boxes and relationships between them are indicated by means of labelled directed arrows. Hierarchical relationships are represented by arrows with a white triangle ending.

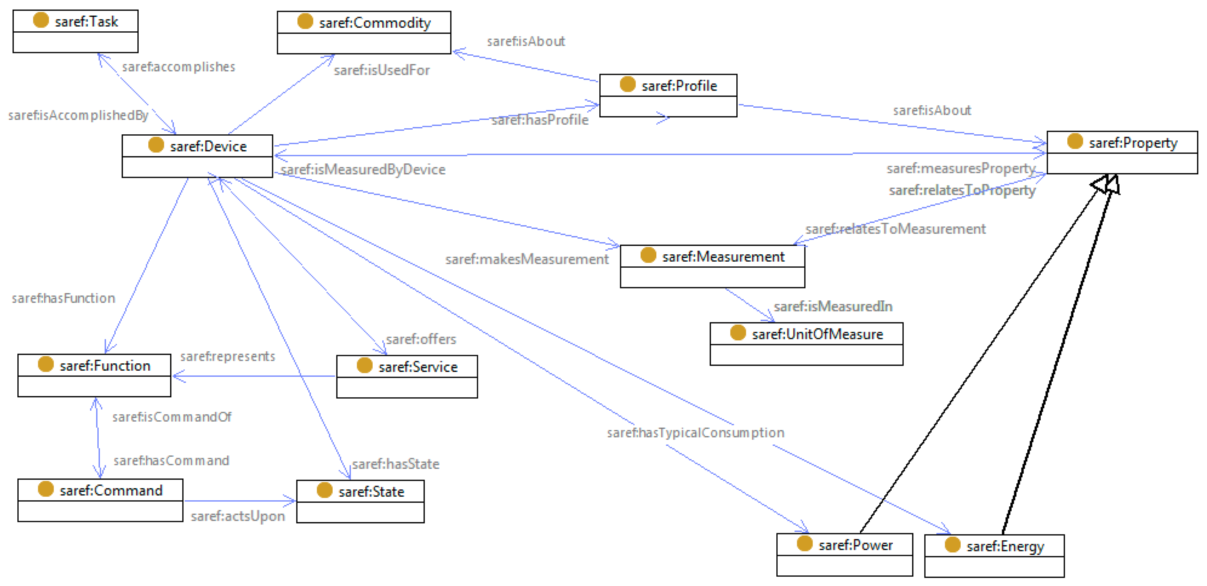


Figure 5. SAREF general overview (Figure extracted from [6]).

The SAREF ontology offers a list of basic functions that can be eventually combined in order to have more complex functions in a single device. For example, a switch offers an actuating function of type "switching on/off". Each function has some associated commands, which can also be picked up as building blocks from a list. For example, the "switching on/off" is associated with the commands "switch on", "switch off" and "toggle". Depending on the function(s) it accomplishes, a device can be found in some corresponding states that are also listed as building blocks.

A Device offers a Service, which is a representation of a Function to a network that makes the function discoverable, registerable and remotely controllable by other devices in the network. A Service can represent one or more functions. A Service is offered by a device that wants (a certain set of) its function(s) to be discoverable, registerable, remotely controllable by other devices in the network. A Service must specify the device that is offering the service, the function(s) to be represented, and the (input and output) parameters necessary to operate the service. A Device is also characterized by an (Energy/Power) Profile that can be used to optimize the energy efficiency in a home or office that are part of a building.

SAREF4BLDG is an OWL ontology that extends SAREF in the building domain in order to represent the devices defined in the IFC (Industry Foundation Classes) standard.[9]

Figure 6 presents an overview of the classes (only the top levels of the hierarchy) and the properties included in the SAREF4BLDG extension. As it can be observed the classes `s4bldg:Building`, `s4bldg:BuindlingSpace` and `s4bldg:PhysicalObjects` have been declared as

[9] http://www.buildingsmart-tech.org/ifc/IFC4/Add1/html/

subclasses of the class `geo:SpatialThing` in order to reuse the conceptualization for locations already proposed by the geo ontology. The modelling of building objects and building spaces are adapted from the SAREF ontology. In this sense, a property between physical objects are the building space in which they are contained is defined, namely `s4bldg:isContainedIn`. Classes or concepts are represented by boxes. Hierarchical relations are represented with plain arrows in which the origin of the arrow represent the more specific concept and the ending the more general one. Finally, dashed arrows represent properties expected to be stated between the classes linked by such properties.

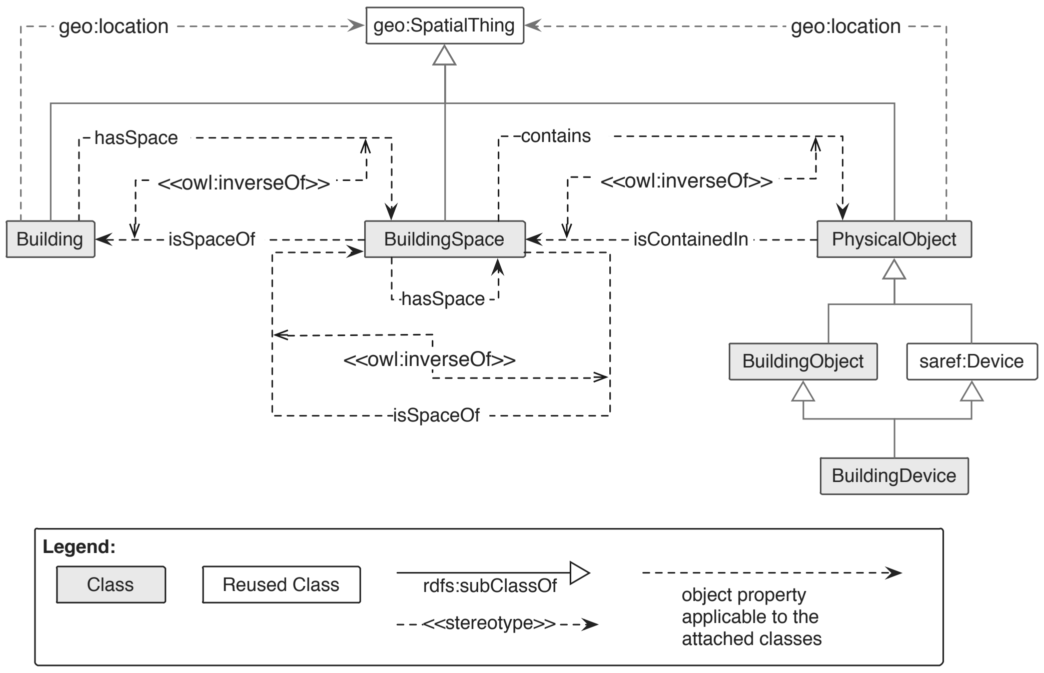


Figure 6. Excerpt of the SAREF4BLDG ontology overview (Figure based on [9])

The current version of the VICINITY ontology uses the SAREF4BLDG ontology to represent building spaces (`s4bldg:BuildingSpace` and `s4bldg:isContainedIn`). It is worth mentioning that the VICINITY partner *Universidad Politécnica de Madrid* have been actively working in the development of the second version of the SAREF ontology and of its extensions.

3.3. oneM2M Base Ontology

Ontologies are used in oneM2M to provide syntactic and semantic interoperability of oneM2M systems with external systems [10]. These external systems are expected to be described by domain specific ontologies, not subject to oneM2M. The only ontology that is specified by oneM2M is the oneM2M Base Ontology formalized in OWL. The oneM2M Base Ontology is the minimal ontology that is required such that other external ontologies can be mapped into oneM2M, e.g., by subclassing, equivalence, etc. These external ontologies can then bring domain knowledge into the oneM2M implementation. They are used to describe real-world “Things” like, e.g., buildings, rooms or just a single lightbulb. The core of oneM2M ontology is illustrated in Figure 7. In such figure, concepts are represented by ellipses and relationships between them are indicated by means of labelled directed arrows. Hierarchical relationships are represented by arrows labelled with “is-a”.

The Base Ontology has been designed with the intent to provide a minimal number of concepts, relations and restrictions that are necessary for semantic discovery of entities in the oneM2M System. To make such entities discoverable in the oneM2M System they need to be semantically described as classes (concepts) in a - technology/vendor/other-standard specific - ontology and these classes (concepts) need to be related to some classes of the Base Ontology as sub-classes.

Additionally, the Base Ontology enables non-oneM2M technologies to build derived ontologies that describe the data model of the non-oneM2M technology for the purpose of interworking with the oneM2M System.

The Base Ontology only contains Classes and Properties but not instances because the Base Ontology and derived ontologies are used in oneM2M to only provide a semantic description of the entities they contain. Instantiation (i.e., data of individual entities represented in the oneM2M System - e.g., devices, things, etc.) is done via oneM2M resources.

The current version of the VICINITY ontology network does not fully cover some domains (e.g., services or functions); however, it will be analysed how to reuse the oneM2M Base ontology in future iterations regarding these functionalities.

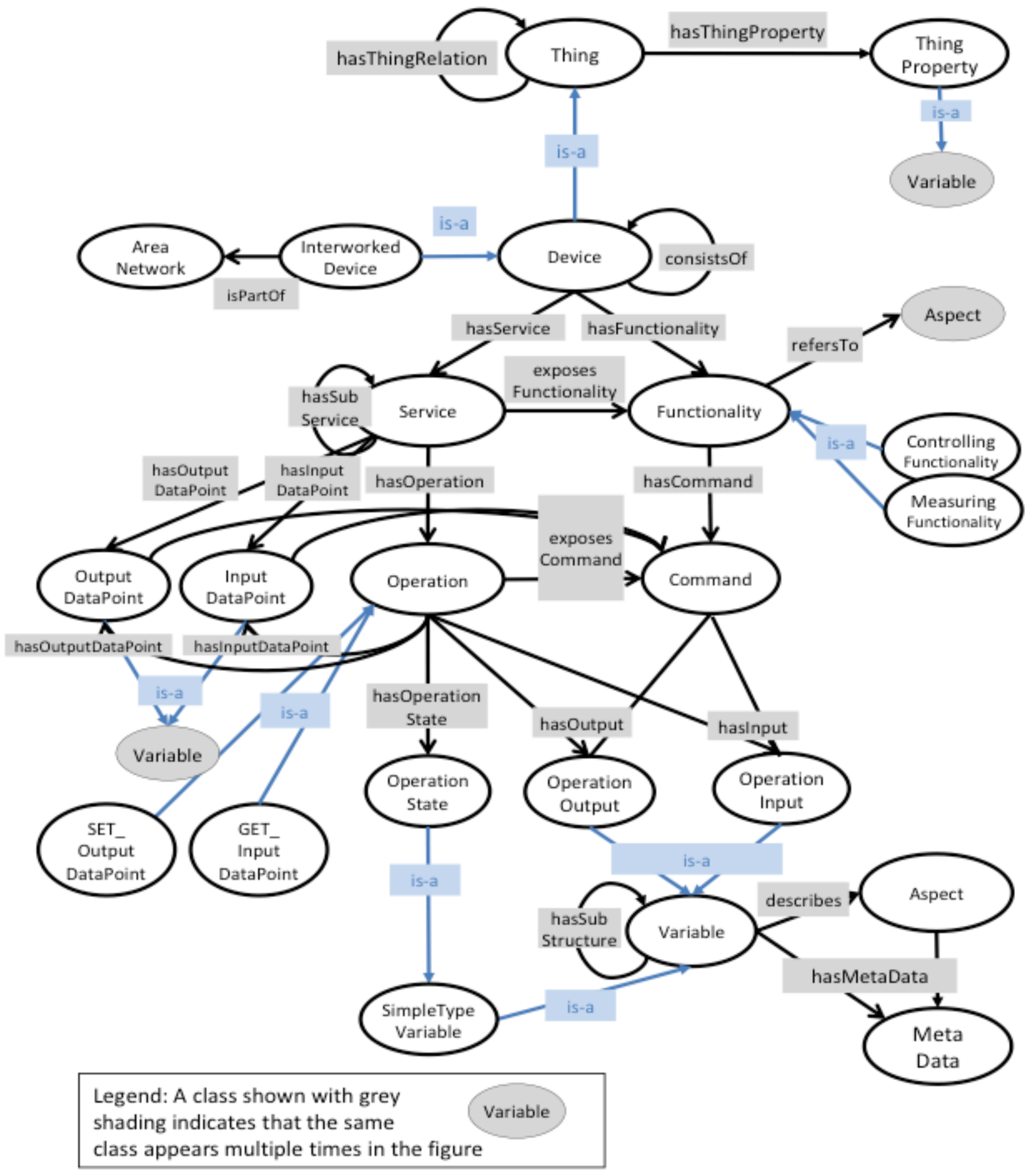


Figure 7. OneM2M Base Ontology (Figure extracted from [10]).

As indicated in Figure 4, mappings between SAREF and oneM2M ontologies have been already defined, more precisely they are documented in [6]. The mappings between classes are represented in Table 1 while the mappings defined between properties are shown in Table 2.

SAREF	Mapping	oneM2M
saref:Device	owl:equivalentClass	oneM2M:Device
saref:Service	owl:equivalentClass	oneM2M:Service
saref:Function	owl:equivalentClass	oneM2M:Function
saref:SensingFunction	owl:equivalentClass	oneM2M:MeasuringFunction
saref:ActuatingFunction	owl:equivalentClass	oneM2M:ControllingFunction
saref:Command	owl:equivalentClass	oneM2M:Command

Table 1. Mappings between SAREF and oneM2M classes.

SAREF	Mapping	oneM2M
saref:offers	owl:equivalentProperty	oneM2M:hasService
saref:hasFunction	owl:equivalentProperty	oneM2M:hasFunction
saref:represents	owl:equivalentProperty	oneM2M:exposesFunction
saref:hasCommand	owl:equivalentProperty	oneM2M:hasCommand
saref:consistsOf	owl:equivalentProperty	oneM2M:consistsOf

Table 2. Mappings between SAREF and oneM2M properties.

3.4. W3C Web of Things

The Web of Things (WoT) aims to bring real-world objects into the World Wide Web, since these objects (i.e., Things) that are intended for everyday life and have little or no computing capability are exposed and can be accessed via the Web [14]. In order to further the development of intelligent, widely used applications through a common knowledge representation of Things to which the developers of the application can refer, the WoT provides a formal representation of the physical knowledge in this domain. To facilitate the ubiquitous experience, it is necessary to link the discovery and the description of environmental issues with the knowledge in this application area. Actual formats of the descriptions can be application-dependent, although such descriptions of Things already exist and can be accessed via the Internet.

An Interest Group of the W3C has been working actively on this topic since 2014 [1] and since January 2017, started its activity as a Working Group. In addition, W3C has developed the terms “Thing Description” and “Interaction” as the main points of the WoT resources.

In the WoT architecture, Things – physical or virtual – are represented by so-called “servients” as shown in Figure 8.

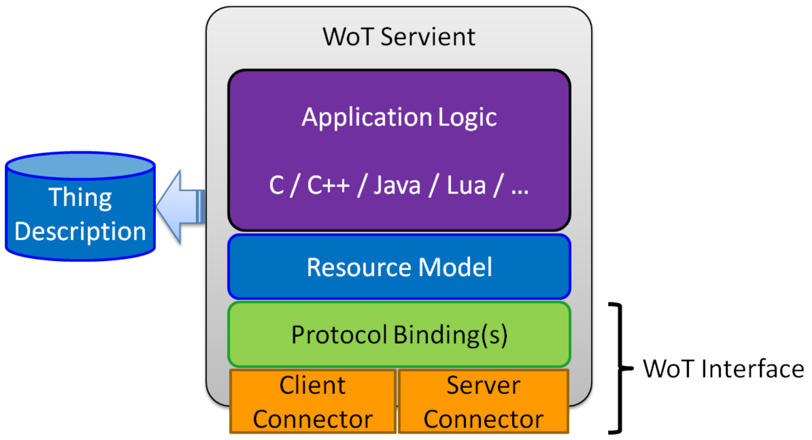


Figure 8. Things are implemented by a servient and communicate through their WoT Interface.[10]

Servients offer a WoT Interface for communication; i.e., a web API following the recommendations of the WoT IG. Information on Servients and their Interfaces are saved as a “Thing description” about them. This Thing Description must be acquired to use and interact with the Thing, since it describes the semantics of a Thing as well as its WoT Interface.

To describe “Things”, the use of the IRE (Identifier, Resource, Entity) pattern was proposed. The IRE pattern is based on the idea that web resources can be used as addressable representation for real-world entities. Websites have a resource ID (URI) associated with them. Real world Entities do not have an URI. Proxy relationships are materialized into IRE which means that a resource is either an informal or formal proxy [1].

By supporting the W3C stakeholders to extend the Web to the physical world, the IRE might prove to be important in designing the Web of Things. After a discussion, the group will concentrate on finding a definition for the semantic models of Things and their abilities.

The UPM, more precisely the researchers involved in the VICINITY project, are part of the WoT IG and WoT WG. In this context, the WoT ontology is being formalized in the OWL ontology language within the VICINITY project taking as input the W3C requirements (see Section 6.1 for more details). In addition, such WoT ontology has been taken as base for the W3C WoT vocabulary. For more details about the contribution of UPM in this standardization activity please see Annex A and B of the VICINITY deliverable “D2.1. “Analysis of Standardisation Context and Recommendations for Standards Involvement”.

[10] http://w3c.github.io/wot/current-practices/wot-practices.html

3.5. Towards value-added services

In general, ontologies and standardization approaches in the IoT can be seen from three different perspectives:

1. **A Device-oriented approach:** Here, things are given names and they are classified according to their functionality. A device is either a light switch, a door-sensor, an actuator, etc. This approach is useful to give an abstract structuring and classification of devices, independent of the application or domain.
2. **A Scenario-based approach:** Here, use-cases and scenarios are defined. To handle these use-cases, commands, such as, e.g., "switch light on/off" are defined. This offers a clear path from requirements to commands and is hence useful in terms of standardization.
3. **A Service-oriented approach**: Here, the focus is on services and their realization in order to reach a specific objective. "Things" are classified based on how and what they can contribute in order to reach said objective. A refrigerator for example offers a "thermal/cooling service" on one hand, but can also shift its energy demand in order to contribute to an energy peak-load management.

All of the above views or approaches are of course not isolated or contradictory. Instead they can be combined in order to provide more meaningful semantic assets.

In earlier work, UNIKL was focusing on service-oriented aspects of ontologies towards smart energy management, during the SmartCoDe Project[11]. As an example, the DogOnt ontology [1] [12] as shown in Figure 9 was enhanced with a service-oriented view. In the given figures, classes or concepts are represented as ellipses. Relationships between them are indicated by labelled directed arrows. Hierarchical relationships are represented by dashed arrows. The colors are used to semantically group classes. Items in green ellipses represent Things in the building domain (physical things), while Item in orange indicate functionalities and ways to interact with things (virtual things).

[11] https://www.fp7-smartcode.eu/
[12] https://github.com/iot-ontologies/dogont

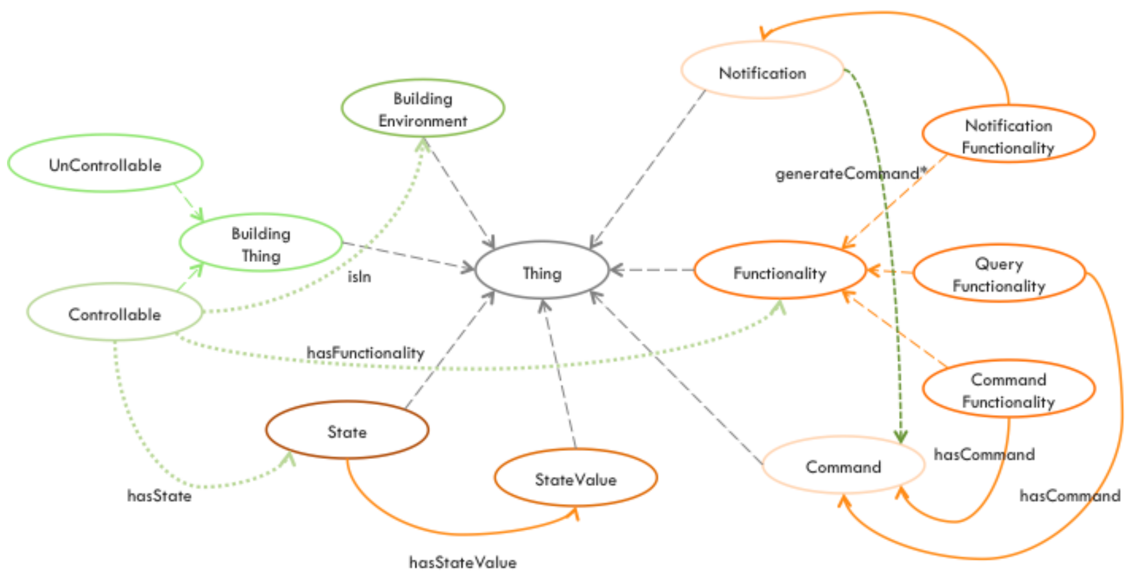


Figure 9. Top-level part of the DogOnt ontology (Figure extracted from [11]).

In the given context, services could be seen as a special kind of functionality a Thing offers. A Smart Service in this context is a functionality directed to reach a certain goal. In contrast to a pure functionality, a Smart Service also includes the autonomous actions and decisions of a "smart appliance".

As an example for a Smart Service, during the SmartCoDe Project, Energy Management was one particular service of interest. For this special case, Energy Management is a further specialization of the Smart Service Class. The resulting enhancement is shown in Figure 10 [11]. In addition to Figure 9, this shows new concepts concerning Services of Things, indicated by red ellipses.

Once Smart Services are present in such an ontology, they can also be connected to and assigned to Things. Using the resulting ontology one can easily search for devices that support a particular service and hence quickly realize new value-added services.

In the VICINITY project, many such value-added services are about to be developed. Being able to quickly find matching devices, based on their contribution towards a particular goal will prove of great value. The current version of the VICINITY ontology already provides the link between devices and the services they offer; future versions of the ontology will build upon such a view as well.

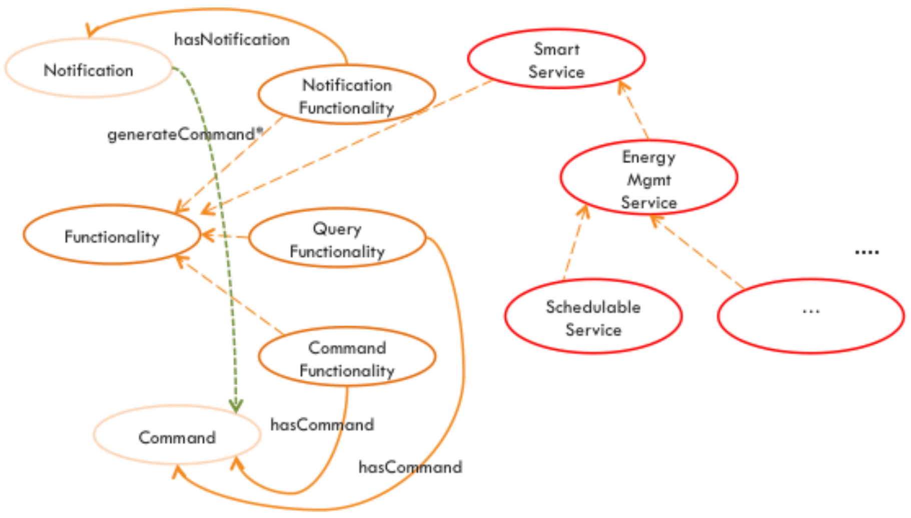


Figure 10. Services merged with DogOnt (Figure extracted from [11]).

3.6. Reused ontology elements in VICINITY network

This section summarized the reused elements from some of the above mentioned ontologies in the current version of the VICINITY ontology network. Table 3 includes the list of classes and properties reused from the SSN and S4BLDG ontologies.

Ontology	Classes reused	Properties reused
SSN	Device SensingDevice FeatureOfInterest Property Condition MeasurementCapability MeasurementProperty OperatingProperty MaintenanceSchedule OperatingPowerRange OperatingRange SurvivalProperty BatteryLifetime SystemLifetime SurvivalRange	forProperty isPropertyOf inCondition hasValue hasProperty hasMeasurementCapability hasMeasurementProoperty hasOperatingProperty hasOperatingRange hasSurvivalProperty hasSurvivalRange qualityOfObservation
S4BLDG	BuildingSpace	isContainedInBuilding

Table 3. Reused elements from SSN and S4BLDG ontologies.

4. Ontology development methodology

This section presents the ontology requirements specification, implementation, publication and maintenance processes defined for the VICINITY ontologies. The development methodology described in this section is based on NeOn methodology [17]. The aim of this chapter is to define the processes to be carried out during the ontology development. Figure 11 shows an overview of the processes that have to be performed and of the products resultant of them.

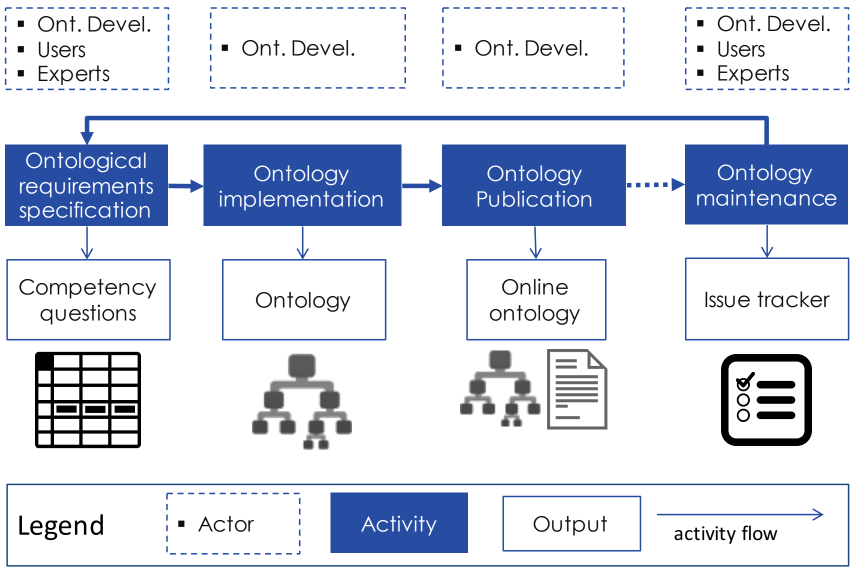


Figure 11. Ontology development process.

4.1. Ontological requirements specification

The aim of the requirements specification process is to state why the ontology is being built and to identify and define the requirements the ontology should fulfil. In this step, involvement and commitment by experts in the specific domain at hand is required to generate the appropriate industry perspective and knowledge.

The activities proposed for the ontology requirement specification process are shown in Figure 12.

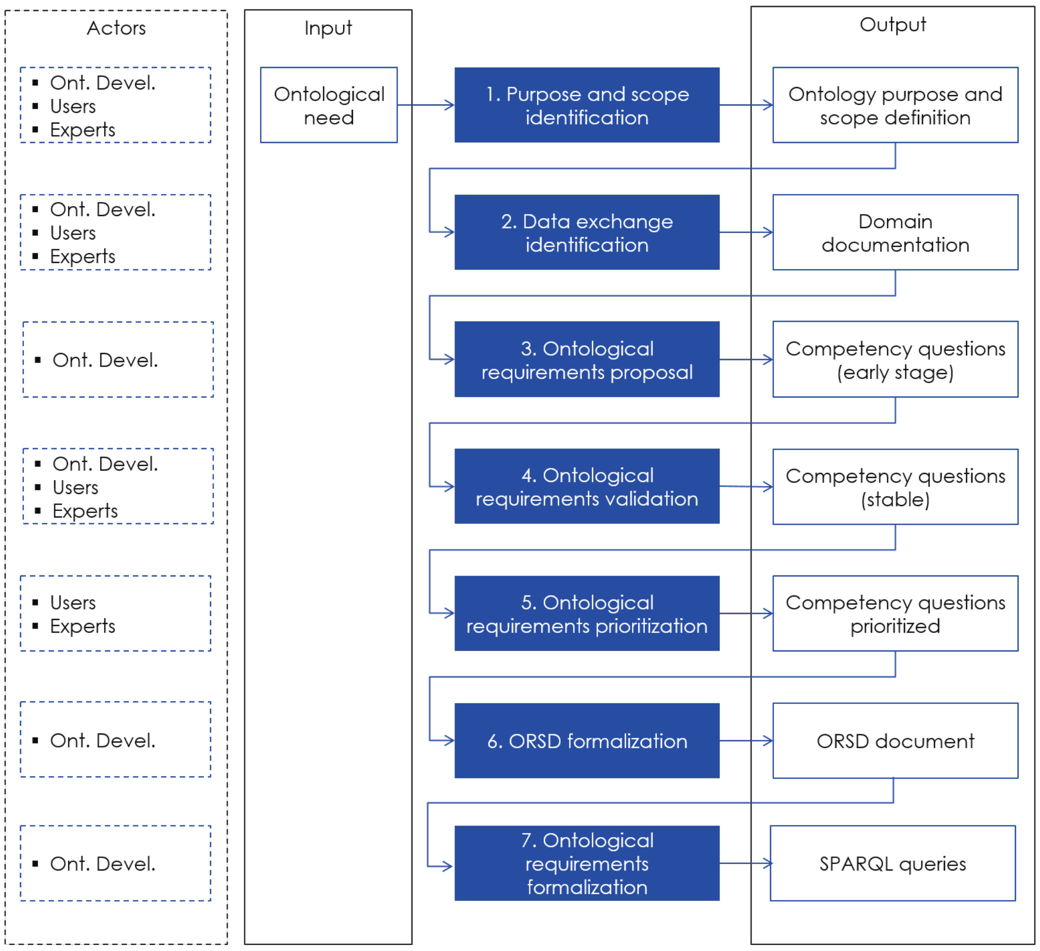

Actors
Input
Output
▪ Ont. Devel.
▪ Users
▪ Experts
Ontological need
1. Purpose and scope identification
Ontology purpose and scope definition
▪ Ont. Devel.
▪ Users
▪ Experts
2. Data exchange identification
Domain documentation
▪ Ont. Devel.
3. Ontological requirements proposal
Competency questions (early stage)
▪ Ont. Devel.
▪ Users
▪ Experts
4. Ontological requirements validation
Competency questions (stable)
▪ Users
▪ Experts
5. Ontological requirements prioritization
Competency questions prioritized
▪ Ont. Devel.
6. ORSD formalization
ORSD document
▪ Ont. Devel.
7. Ontological requirements formalization
SPARQL queries

Figure 12. Workflow proposed for ontology requirement specification.

4.1.1. Purpose and scope identification

The goal of this activity is to define the purpose and scope of the given ontology or ontology module. The ontology development team works in collaboration with users and domains experts to define the purpose and scope of each ontology or module to be developed.

The communication between the domain experts, users and ontology development team could be carried out by means of an online meeting.

4.1.2. Data exchange identification

The goal of this activity is to provide the ontology development team with the necessary documentation about the domain to be modelled. In this case, the documentation to be shared might correspond to:

- Manuals,
- APIs specifications,
- Datasets,
- Standards,
- Formats.

The domain experts are the responsible of providing this documentation.

4.1.3. Ontological requirements proposal

Taking as input the documentation and data provided by domain experts and users, the ontology development team generates a first proposal of ontological requirements written in the form of Competency Questions [12].

The format used for this proposal follows a tabular approach in which the following fields are included:

- Requirement identifier, which should be unique for each requirement
- Competency question, which includes:
  - Question
  - Answer
- Provenance information, which includes:
  - Origin of the requirement
- Partners (users or domain experts) related to the definition of the requirement
- Comments about the requirement
- Relation with other requirements
- Priority of the requirements, which can be high, medium or low
- Status of the requirement, which can be proposed, accepted, rejected or superseded by
- Sprint in which the requirements must be implemented

4.1.4. Ontological requirements completion and validation

During this activity domain experts and users in collaboration with the ontology development team validate whether the ontology requirements defined in the previous step are correct and complete.

The following criteria can be used in this validation task as stated in [12]:

- A set of requirements is correct if each requirement refers to some features of the ontology to be developed.
- A set of requirements can be considered complete if users and domain experts review the requirements and confirm that they are not aware of additional requirements.
- A set of requirements can be considered internally consistent if no conflicts exist between them.

- A set of requirements is verifiable if there is a finite process with a reasonable cost that tests whether the final ontology satisfies each requirement.
- Each requirement must be understandable to end-users and domain experts.
- An ontology requirement is unambiguous if it has only one meaning; that is, if it does not admit any doubt or misunderstanding.
- A set of requirements is concise if each and every requirement is relevant, and no duplicated or irrelevant requirements exist.
- A set of requirements is realistic if each and every requirement meaning makes sense in the domain.
- A set of requirements is modifiable if its structure and style allow changing issues in an easy, complete and consistent way.

4.1.5. Ontological requirements prioritization

This activity will be performed if there is the need for prioritizing functional requirements. The main implications of this prioritization are the possibility of planning and scheduling the development of the ontology in sprints. This prioritization would be present in the backlog driving therefore the ontology development process.

To carry out this prioritization the ontology development team works with the domain experts to identify which requirements need to be fulfilled first. The communication between the domain experts and the ontology development team could be carried out by means of an on-line or in-person interview.

4.1.6. ORSD formalization

Once the ontology development team has all the information about the requirements, they create the Ontology Requirements Specification Document (ORSD). This specification document stores all the requirements identified and the information associated to them.

4.1.7. Ontological requirements formalization

During this activity, the ontological requirements written in natural language are formalized into test cases. These tests cases, which are stored in RDF files, should include:

- Identifier of the requirement associated,
- Description of the test case, which includes a link to the ORSD,
- SPARQL query extracted from the competency question,
- Expected result of the query.

The goal of this activity is to create machine-readable test cases. These test cases have SPARQL queries which can be executed over the ontology to verify if the ontology satisfies the ontological requirements identified.

4.2. Ontology implementation

The aim of the ontology implementation process is to build the ontology using a formal language, based on the ontological requirements identified by the domain experts. After defining the first set of requirements, though modification and addition of requirements is allowed during the development, the ontology implementation phase is carried out through a number of sprints. The ontology developers schedule and plan the ontology development according to the prioritization of the requirements in the ontology requirements specification process. The ontology development team builds the ontology iteratively, implementing only a certain number of requirements in each iteration. The output of each iteration is a new version of the ontology.

Figure 13 shows the activities to carry out during the ontology implementation for each iteration.

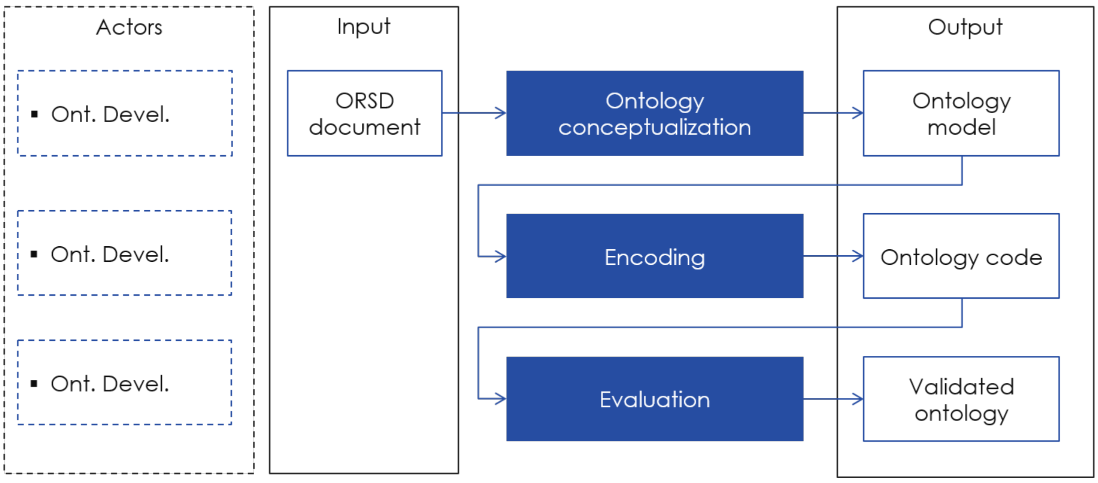


Figure 13. Workflow proposed for ontology implementation.

4.2.1. Ontology conceptualization

The aim of this activity is to build an ontology model from the ontological requirements identified in the requirements specification process. During the ontology conceptualization, the domain knowledge obtained from the ORSD document is organized and structured into a model by the ontology developers.

4.2.2. Encoding

During this activity, the ontology development team generates computable models in the OWL language from the ontology model.

The ontology code resultant from this activity includes metadata, such as creator, title, publisher, license and version of the ontology.

It is worth noting that during the development of the VICINITY ontology the following *Ontology Versioning* convention has been adopted:

The versioning identification will be as similar as possible to the conventions used in software development. In this case, each release will follow the pattern *v.major.minor.fix*, where each field follows the rules:

- **major**: The field is updated when the ontology covers the complete domain it intends to model. That is, it is a complete product and covers the final goal of the development.
- **minor**: The field is updated when:
    - all the requirements of a subdomain are covered.
    - documentation is added to the ontology.
- **fix**: The field is updated when:
    - typos or bugs are corrected in the ontology.
    - classes, relationships, axioms, individuals or annotations are added, deleted or modified.

In each iteration the minor and fix fields might be changed from zero to several times.

Each ontology developed under the VICINITY ontology network would follow its particular version status. For example, the mapping ontology might be in version v1.0.0 while the Core ontology might be in version v0.2.3. That is, the network evolution is not managed as a whole, as it is not a particular product but the virtual composition of many ontologies where each ontology evolves independently.

4.2.3. Evaluation

Before publishing a release version of the ontology, the ontology developers evaluate the ontology in different aspects:

- The ontology developers guarantee that the ontology does not have syntactic, modelling or semantic errors.
- The ontology developers guarantee that the ontology fulfil the requirements scheduled for the ontology using the test cases generated in the requirements specification process.

4.3. Ontology publication

The aim of the ontology publication process is to provide an online ontology accessible both as a human-readable documentation and a machine-readable file from its URI. The ontology needs to be evaluated before its publication to guarantee that is ready to be used.

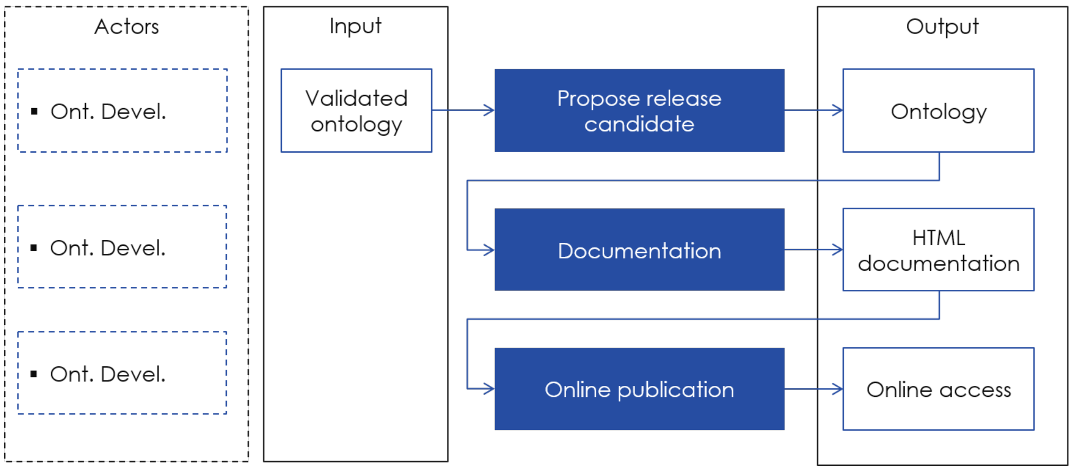


Figure 14. Workflow proposed for ontology publication.

4.3.1. Propose release candidate

Once the ontology developers have implemented and validated the ontology, they propose a release version of the ontology to be published on the Web:

- In case the ontology fulfils all the requirements of a given subdomain, the ontology development team generates a release version of the ontology, e.g., the VICINITY ontology for release 0.1 is tagged as “release version v0.1.X”.
- In case the ontology does not implement all the ontological requirements identified, only those scheduled for the iteration, the ontology development team generates a pre-release version of the ontology.

Both release and pre-release versions of the ontologies are evaluated and ready to be used.

4.3.2. Documentation

Taking as input the ontology generated in the previous activities, the ontology development team in collaboration with the domain experts generates the ontology documentation of the release candidate. This documentation includes:

- An HTML description of the ontology which describes the classes, properties and data properties of the ontology, and the license URI and title being used. The domain experts have to collaborate with the ontology development team to describe the classes and the properties. This description also includes metadata, such as creator, publisher, date of creation, last modification or version number.
- Diagrams which store the graphical representation of the ontology, including taxonomy and class diagrams.

4.3.3. Online publication

Once the documentation of the ontology has been generated, the ontology is published on the Web. This online ontology is accessible via its namespace URI as a machine-readable file and a human-readable documentation using content negotiation.

4.4. Ontology Maintenance

The goal of this activity is to update and add new requirements to the ontology that are not identified in the ORSD or to identify and corrects errors or to schedule a new iteration for ontology development. During the ontology development process, the domain experts can propose new requirements or improvements over the ontology. If these requirements or improvements are approved by the ontology development team, they are added to the ontology.

4.5. Experience with the described methodology

After developing the VICINITY ontologies following the described methodology we conclude that it helps us to manage the ontology development efficiently in all the development processes.

Regarding the requirements specification process, having the ORSD structured help us to define the requirements and to identify its information associated, e.g. provenance or additional comments which allow the developers to improve the understanding of the requirements. However, while we were developing the ontologies we refined the structure of the ORSD. This refinement included the deletion of the field “Project Task”, because we did not make use of it during the development of the ontologies, and the addition of the field “Superseded by”, because some of the requirements evolve during the development and instead of changing them we have decided to indicate that these requirements are deprecated.

It is worth noting that, although the majority of these processes are stable and already used in other projects, as they are extracted from NeOn Project[13], the step “Ontological requirements formalization” is still an experimental step we added to verify if the ontology fulfils all the requirements defined in the ORSD. We are trying to analyze if all the ontological requirements can be represented as SPARQL queries.

[13] http://www.neon-project.org/nw/Welcome_to_the_NeOn_Project

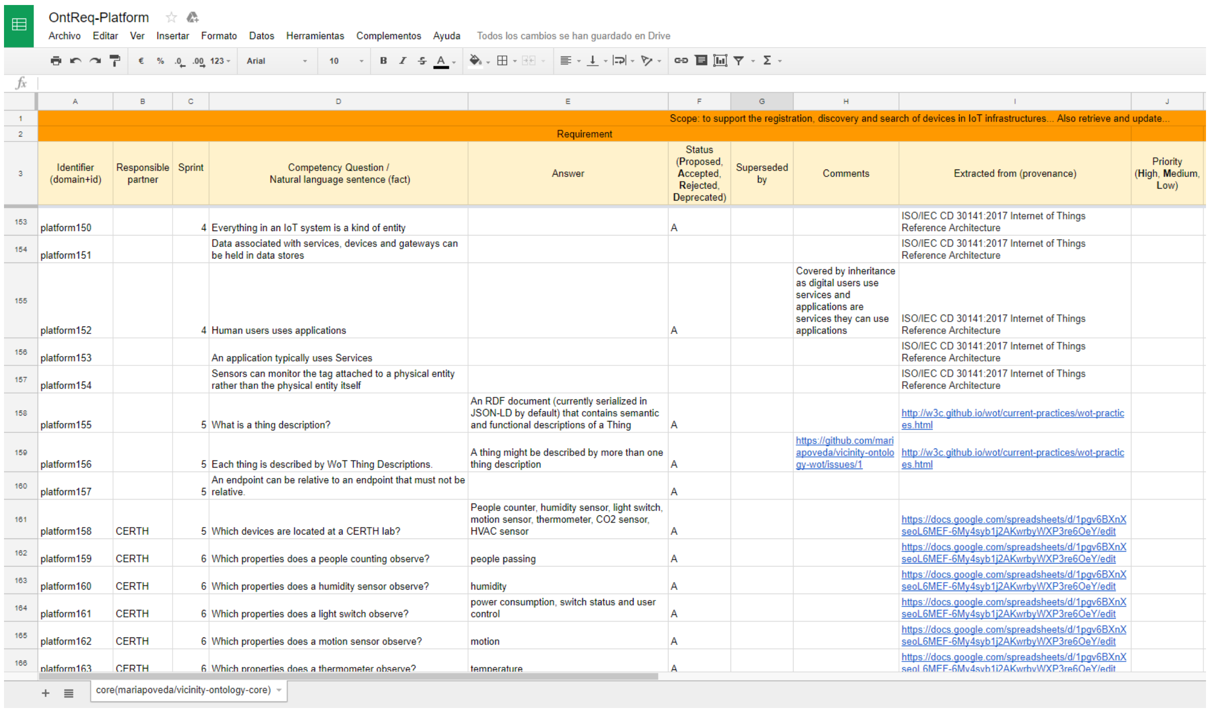
5. Ontology development infrastructure

In this section the ontology development infrastructure used to support all the activities in the ontology development process is described.

5.1. Infrastructure for ontological requirements specification

To support the ontology requirements specification phase, the ontology developers use Google Spreadsheets to gather and store the requirements. These spreadsheets are associated to the VICINITY ontologies, published online and openly accessible.

Each ontology has a different spreadsheet associated. The name of the sheet indicates the ontology to which the requirements belong to and the GitHub repository in which the ontology is stored, e.g., the tab named "WoT(mariapoveda/wot-ontology)" stores the requirements of the Web of Things ontology, with the associated repository https://github.com/mariapoveda/wot-ontology. Figure 15 shows an excerpt from the requirements stored for the VICINITY ontology model.


OntReq-Platform
Archivo Editar Ver Insertar Formato Datos Herramientas Complementos Ayuda
Todos los cambios se han guardado en Drive
Arial
10
core(mariapoveda/vicinity-ontology-core)

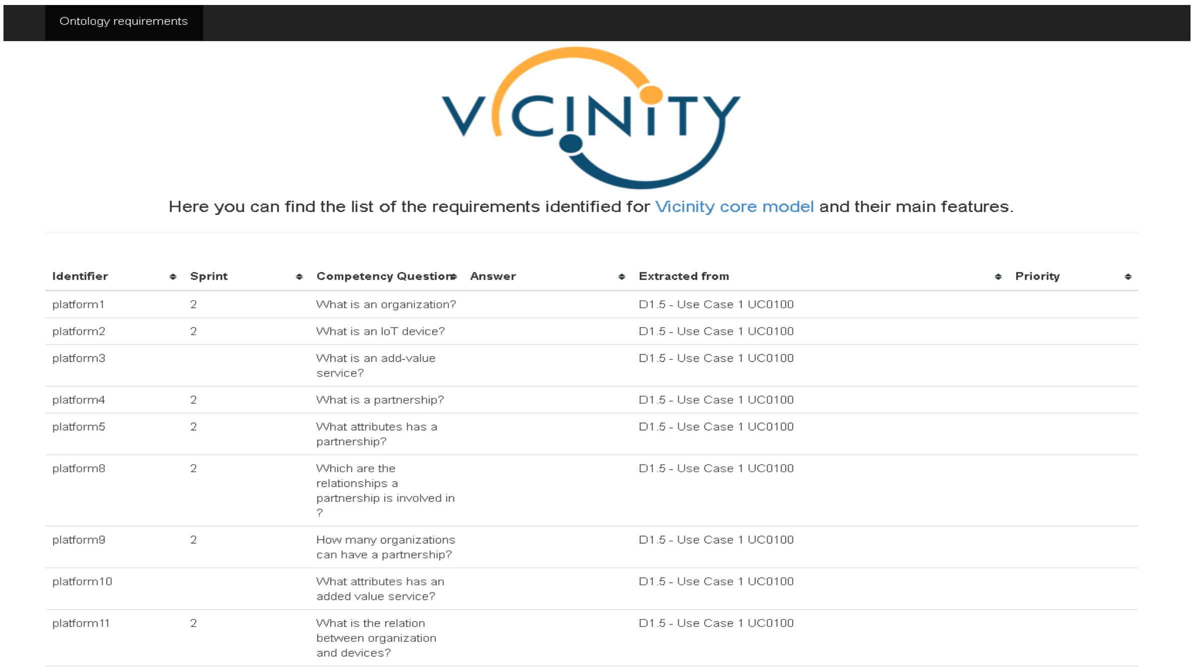
Scope: to support the registration, discovery and search of devices in IoT infrastructures... Also retrieve and update...

Requirement

	A	B	C	D	E	F	G	H	I	J
3	Identifier (domain+id)	Responsible partner	Sprint	Competency Question / Natural language sentence (fact)	Answer	Status (Proposed, Accepted, Rejected, Deprecated)	Superseded by	Comments	Extracted from (provenance)	Priority (High, Medium, Low)
153	platform150		4	Everything in an IoT system is a kind of entity		A			ISO/IEC CD 30141:2017 Internet of Things Reference Architecture	
154	platform151			Data associated with services, devices and gateways can be held in data stores					ISO/IEC CD 30141:2017 Internet of Things Reference Architecture	
155	platform152		4	Human users uses applications		A		Covered by inheritance as digital users use services and applications are services they can use applications	ISO/IEC CD 30141:2017 Internet of Things Reference Architecture	
156	platform153			An application typically uses Services					ISO/IEC CD 30141:2017 Internet of Things Reference Architecture	
157	platform154			Sensors can monitor the tag attached to a physical entity rather than the physical entity itself					ISO/IEC CD 30141:2017 Internet of Things Reference Architecture	
158	platform155		5	What is a thing description?	An RDF document (currently serialized in JSON-LD by default) that contains semantic and functional descriptions of a Thing	A			http://w3c.github.io/wot/current-practices/wot-practices.html	
159	platform156		5	Each thing is described by WoT Thing Descriptions.	A thing might be described by more than one thing description	A		https://github.com/mariapoveda/vicinity-ontology-wot/issues/1	http://w3c.github.io/wot/current-practices/wot-practices.html	
160	platform157		5	An endpoint can be relative to an endpoint that must not be relative.		A				
161	platform158	CERTH	5	Which devices are located at a CERTH lab?	People counter, humidity sensor, light switch, motion sensor, thermometer, CO2 sensor, HVAC sensor	A			https://docs.google.com/spreadsheets/d/1pgv6BXnXseoL6MEF-6My4syb1j2AKwrbyWXP3re6OeY/edit	
162	platform159	CERTH	6	Which properties does a people counting observe?	people passing	A			https://docs.google.com/spreadsheets/d/1pgv6BXnXseoL6MEF-6My4syb1j2AKwrbyWXP3re6OeY/edit	
163	platform160	CERTH	6	Which properties does a humidity sensor observe?	humidity	A			https://docs.google.com/spreadsheets/d/1pgv6BXnXseoL6MEF-6My4syb1j2AKwrbyWXP3re6OeY/edit	
164	platform161	CERTH	6	Which properties does a light switch observe?	power consumption, switch status and user control	A			https://docs.google.com/spreadsheets/d/1pgv6BXnXseoL6MEF-6My4syb1j2AKwrbyWXP3re6OeY/edit	
165	platform162	CERTH	6	Which properties does a motion sensor observe?	motion	A			https://docs.google.com/spreadsheets/d/1pgv6BXnXseoL6MEF-6My4syb1j2AKwrbyWXP3re6OeY/edit	
166	platform163	CERTH	6	Which properties does a thermometer observe?	temperature	A			https://docs.google.com/spreadsheets/d/1pgv6BXnXseoL6MEF-6My4syb1j2AKwrbyWXP3re6OeY/edit	

Figure 15. VICINITY Core ontology requirements in a Google Spreadsheet.

These Google Spreadsheets are converted to an HTML file with the most relevant information for the users to facilitate visualization. Figure 16 shows an excerpt from the HTML document generated from the requirements of the VICINITY ontology model.

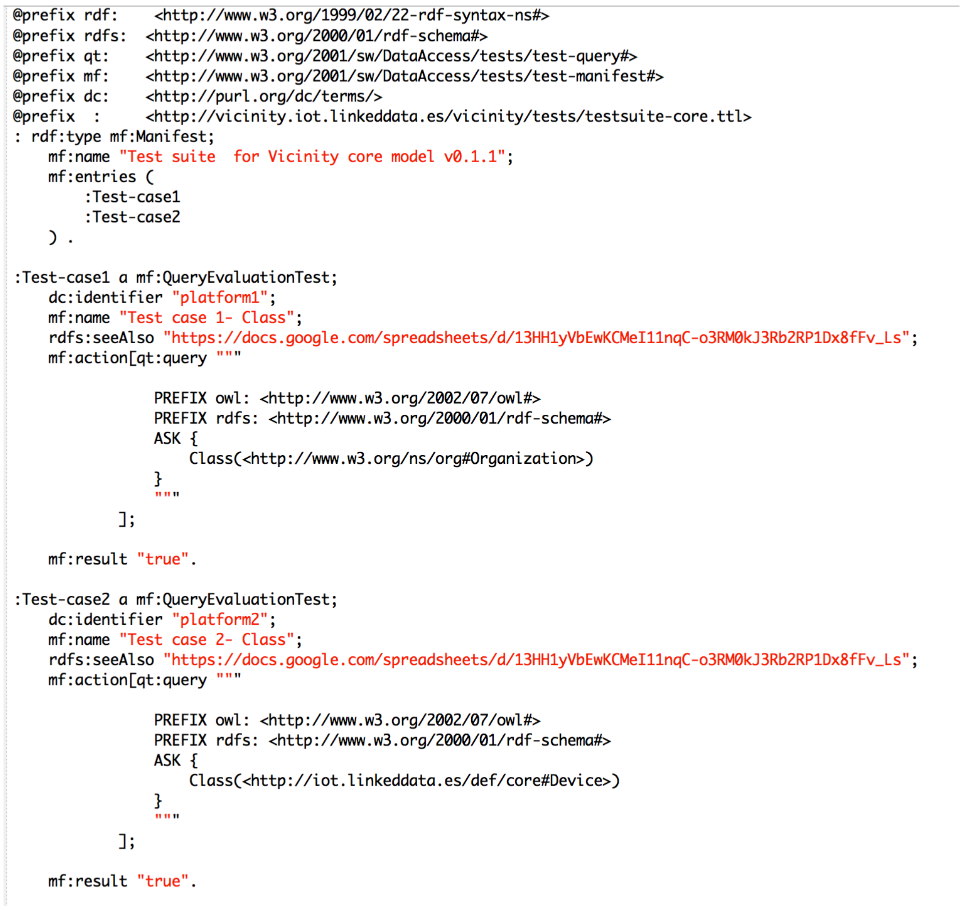


Here you can find the list of the requirements identified for Vicinity core model and their main features.

Identifier	Sprint	Competency Question	Answer	Extracted from	Priority
platform1	2	What is an organization?		D1.5 - Use Case 1 UC0100	
platform2	2	What is an IoT device?		D1.5 - Use Case 1 UC0100	
platform3		What is an add-value service?		D1.5 - Use Case 1 UC0100	
platform4	2	What is a partnership?		D1.5 - Use Case 1 UC0100	
platform5	2	What attributes has a partnership?		D1.5 - Use Case 1 UC0100	
platform8	2	Which are the relationships a partnership is involved in ?		D1.5 - Use Case 1 UC0100	
platform9	2	How many organizations can have a partnership?		D1.5 - Use Case 1 UC0100	
platform10		What attributes has an added value service?		D1.5 - Use Case 1 UC0100	
platform11	2	What is the relation between organization and devices?		D1.5 - Use Case 1 UC0100	

Figure 16. VICINITY Core ontology requirements in an HTML document.

Regarding the ontological requirements formalization, an RDF file where the test cases are store is created. Each test case follows the same structure. Figure 16 shows an excerpt of the test cases generated for VICINITY Core ontology.



```
@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
@prefix rdfs:   <http://www.w3.org/2000/01/rdf-schema#>
@prefix qt:     <http://www.w3.org/2001/sw/DataAccess/tests/test-query#>
@prefix mf:     <http://www.w3.org/2001/sw/DataAccess/tests/test-manifest#>
@prefix dc:     <http://purl.org/dc/terms/>
@prefix  :      <http://vicinity.iot.linkeddata.es/vicinity/tests/testsuite-core.ttl>
: rdf:type mf:Manifest;
    mf:name "Test suite  for Vicinity core model v0.1.1";
    mf:entries (
        :Test-case1
        :Test-case2
    ) .

:Test-case1 a mf:QueryEvaluationTest;
    dc:identifier "platform1";
    mf:name "Test case 1- Class";
    rdfs:seeAlso "https://docs.google.com/spreadsheets/d/13HH1yVbEwKCMeI11nqC-o3RM0kJ3Rb2RP1Dx8fFv_Ls";
    mf:action[qt:query """

                PREFIX owl: <http://www.w3.org/2002/07/owl#>
                PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
                ASK {
                    Class(<http://www.w3.org/ns/org#Organization>)
                }
                """
            ];

    mf:result "true".

:Test-case2 a mf:QueryEvaluationTest;
    dc:identifier "platform2";
    mf:name "Test case 2- Class";
    rdfs:seeAlso "https://docs.google.com/spreadsheets/d/13HH1yVbEwKCMeI11nqC-o3RM0kJ3Rb2RP1Dx8fFv_Ls";
    mf:action[qt:query """

                PREFIX owl: <http://www.w3.org/2002/07/owl#>
                PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
                ASK {
                    Class(<http://iot.linkeddata.es/def/core#Device>)
                }
                """
            ];

    mf:result "true".
```

Figure 17. Examples of VICINITY Core ontology requirements formalized

5.2. Infrastructure for ontology implementation

To support the ontology implementation phase, the ontology developers use several tools to edit, store and evaluate the ontology.

The ontology development team uses an **ontology editor**, such as Protégé, to generate the ontology code. Protégé allows the creation, visualization and manipulation of ontologies in various representation formats.

As solution for **ontology storage**, each VICINITY ontology is stored in a GitHub repository. For each ontology, its GitHub repository includes:

- A folder with the implementation of the ontology.
- A folder with the ontology modelling diagrams.
- A folder with the documentation of the ontology.

- A folder with the tests generated from the ontological requirements.

The ontology developers may use the GitHub or Git proposed workflow to develop the ontology, which can be summarized in the next steps:

1. Create a new branch from the master branch.
2. Add changes to the ontology and commit them. Each commit has to be associated with a commit message, which is a description explaining why a particular change was made.
3. Open a pull request to start a discussion over the changes.
4. If the pull request is approved, merge the new branch into the master branch.

The development team uses OnToology[14] to generate the **documentation** and to **evaluate** the ontology. OnToology, which integrates Widoco[15], OOPS! [15][16] and AR2DTool[17], generates automatically a folder in the GitHub repository which includes all the resources: diagrams, documentation and evaluation report.

The structure of the folders generated by OnToology for each ontology contained in the Github repository is represented in Figure 18:

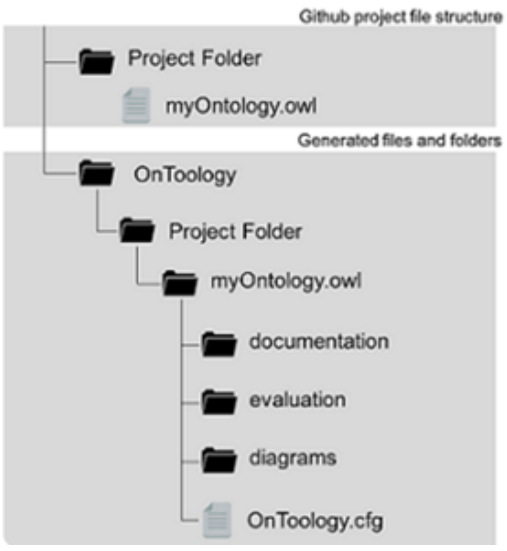


Figure 18. OnToology folder structure.

[14] http://ontoology.linkeddata.es/
[15] https://github.com/dgarijo/Widoco/
[16] http://oops.linkeddata.es/
[17] https://github.com/idafensp/ar2dtool

The documentation of the ontology is generated using Widoco, which provides a description of the classes that must be completed by the domain experts. The diagrams of the ontology are generated using AR2DTool, which includes both taxonomy and entity relationship diagrams. The evaluation is provided by OOPS!, which detects the most common pitfalls that appear when developing ontologies. All the resources generated by OnToology are updated whenever there is an alteration in the repository.

Besides this documentation and evaluation, whenever there is a pull request in the GitHub repository over the ontology the SPARQL queries of the test cases are executed. GitHub will inform the developer by means of a message in the pull request if there are some queries which do not provide the expected result indicated in the test cases. If this occurs, it means that there are some requirements which are not fulfilled by the ontology.

5.3. Infrastructure for ontology publication

To support the ontology publication phase, the ontology developers publish the ontology online to be accessible to everyone and they also publish the releases in the **the online VICINITY ontologies portal**[18]. The ontology development team publishes the releases of the ontologies in to such portal to make the ontology and its documentation accessible to all the users. The portal has different sections to provide different information, namely:

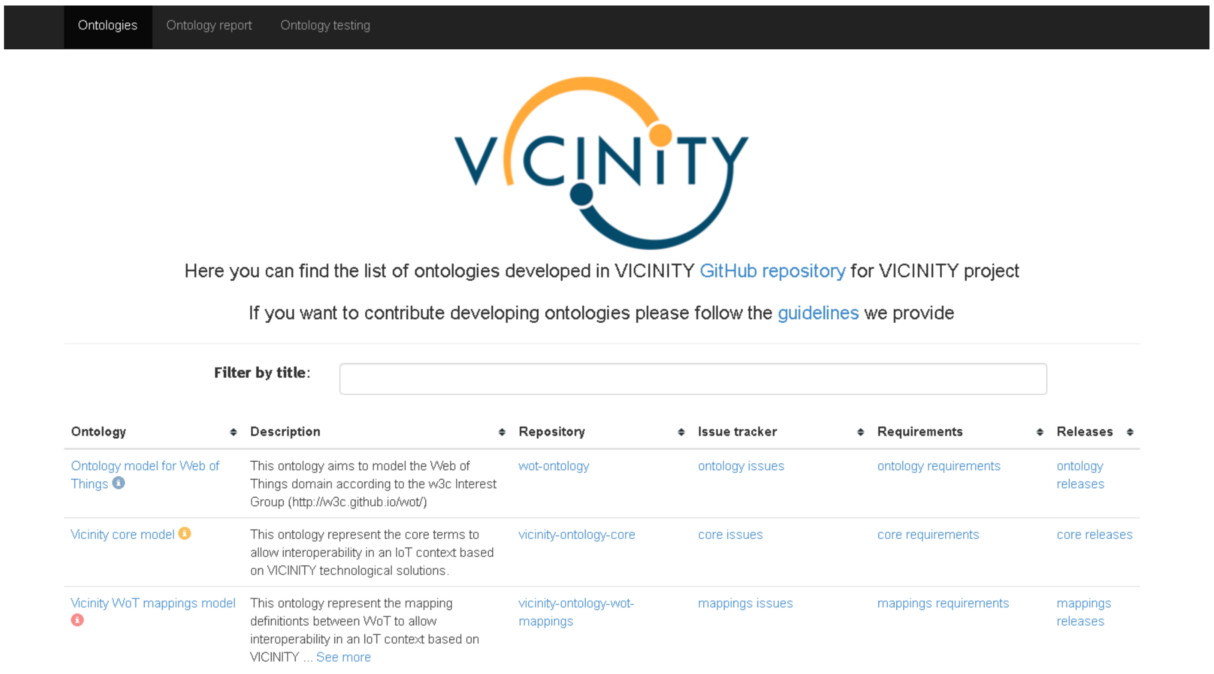
1. Ontologies
2. Good practices
3. Ontology testing

The **Ontologies** section is the main section of the portal, which shows the main information about the ontologies created. The section follows a tabular approach which includes:

- Link to the ontology documentation published on the Web, which is generated by Widoco;
- Link to the ontology evaluation generated by OOPS!;
- Ontology description;
- Link to each Github repository;
- Links to each GitHub issue tracker;
- HTML description of the requirements identified by the domain experts;
- Link to each ontology releases.

Figure 19 shows an overview of the information exposed in the VICINITY ontology portal.

[18] http://vicinity.iot.linkeddata.es



Here you can find the list of ontologies developed in VICINITY GitHub repository for VICINITY project

If you want to contribute developing ontologies please follow the guidelines we provide

Filter by title:

Ontology	Description	Repository	Issue tracker	Requirements	Releases
Ontology model for Web of Things	This ontology aims to model the Web of Things domain according to the w3c Interest Group (http://w3c.github.io/wot/)	wot-ontology	ontology issues	ontology requirements	ontology releases
Vicinity core model	This ontology represent the core terms to allow interoperability in an IoT context based on VICINITY technological solutions.	vicinity-ontology-core	core issues	core requirements	core releases
Vicinity WoT mappings model	This ontology represent the mapping definitions between WoT to allow interoperability in an IoT context based on VICINITY ... See more	vicinity-ontology-wot-mappings	mappings issues	mappings requirements	mappings releases

Figure 19. Overview of VICINITY ontology portal.

This section of the portal includes a diagram which reflects the main concepts of each ontology and how they are related with each other, as it will be further described in Section 6.

Regarding the **good practices** section, the portal also provides the users a brief overview of the proposed process for developing ontologies and some guidelines[19], which should be followed by the domain experts and ontology developers who wants to contribute. This section includes information about:

- How the repository should be structured;
- The tools recommended to be used during the ontology development process;
- Ontology versioning;
- Issues management.

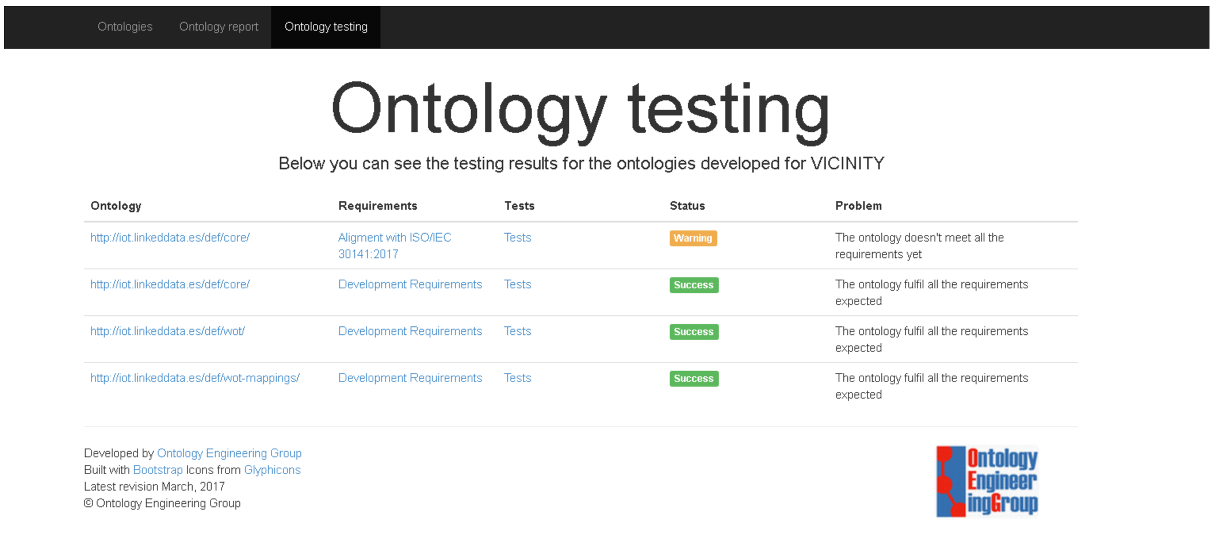
Finally, the VICINITY ontology portal also has an **ontology testing** section[20] which follows a tabular approach and includes:

- The link to the ontology published on the Web;

[19] http://vicinity.iot.linkeddata.es/vicinity/howwework.html
[20] http://vicinity.iot.linkeddata.es/vicinity/testing.html

- The tests generated to verify the ontology requirements;
- An HTML file with the description of the ontological requirements and the results of the tests for each requirement;
- The status of each ontology regarding its compliance with the requirements;
- The problem identified according to the tests, if exists.

An example of testing results for the VICINITY ontologies is shown in Figure 20.



Ontology testing

Below you can see the testing results for the ontologies developed for VICINITY

Ontology	Requirements	Tests	Status	Problem
http://iot.linkeddata.es/def/core/	Aligment with ISO/IEC 30141:2017	Tests	Warning	The ontology doesn't meet all the requirements yet
http://iot.linkeddata.es/def/core/	Development Requirements	Tests	Success	The ontology fulfil all the requirements expected
http://iot.linkeddata.es/def/wot/	Development Requirements	Tests	Success	The ontology fulfil all the requirements expected
http://iot.linkeddata.es/def/wot-mappings/	Development Requirements	Tests	Success	The ontology fulfil all the requirements expected

Developed by Ontology Engineering Group
Built with Bootstrap Icons from Glyphicons
Latest revision March, 2017
© Ontology Engineering Group

Figure 20. Overview of the testing section of the VICINITY ontology portal.

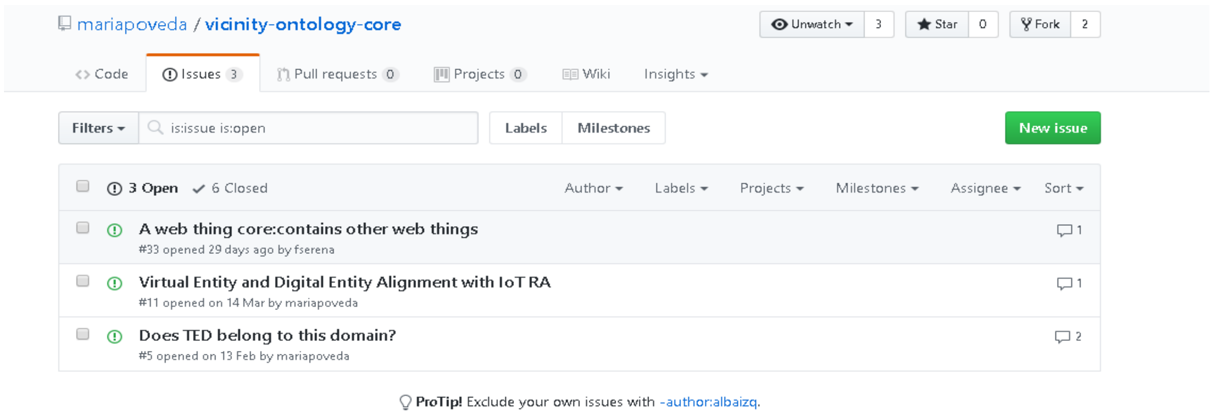
5.4. Infrastructure for ontology maintenance

To support the maintenance of the ontology, the ontology developers use an **issue tracker** which manages and maintain the list of issues identified by the domain experts and ontology developers.

All the changes and improvements over the ontology need to be agreed by all the ontology development team. The GitHub issue tracker[21] is used to discuss improvements and issues about the domains. If domain experts or ontology developers want to add new concepts to the ontology they have to create a new issue in the GitHub issue tracker, which will be used to start a discussion about the approval of the proposal.

[21] https://help.github.com/articles/about-issues/

GitHub issues will be marked as "open" and "closed". The issues can also have an assignee which is the person that is responsible for moving the issue forward. Figure 21 shows the list of the opened issues associated to the VICINITY Core ontology.


mariapoveda / vicinity-ontology-core

Filters ▾ is:issue is:open | Labels | Milestones | New issue

3 Open ✓ 6 Closed

A web thing core:contains other web things
#33 opened 29 days ago by fserena — 1

Virtual Entity and Digital Entity Alignment with IoT RA
#11 opened on 14 Mar by mariapoveda — 1

Does TED belong to this domain?
#5 opened on 13 Feb by mariapoveda — 2

ProTip! Exclude your own issues with -author:albaizq.

Figure 21. GitHub issues related to the VICINITY Core ontology.

The ontology developers should label each issue according to its topic or status in order to organize the different types of issues. GitHub comes with a few labels by default: bug, duplicate, enhancement, invalid, question, and won't fix. Those issues that represent new requirements for the ontology will be labelled by the developers as a "requirement" issue. In addition, ontology developers can create new labels if they think they can improve issue management. The ontology developers are also responsible for closing the issues created that have already been addressed.

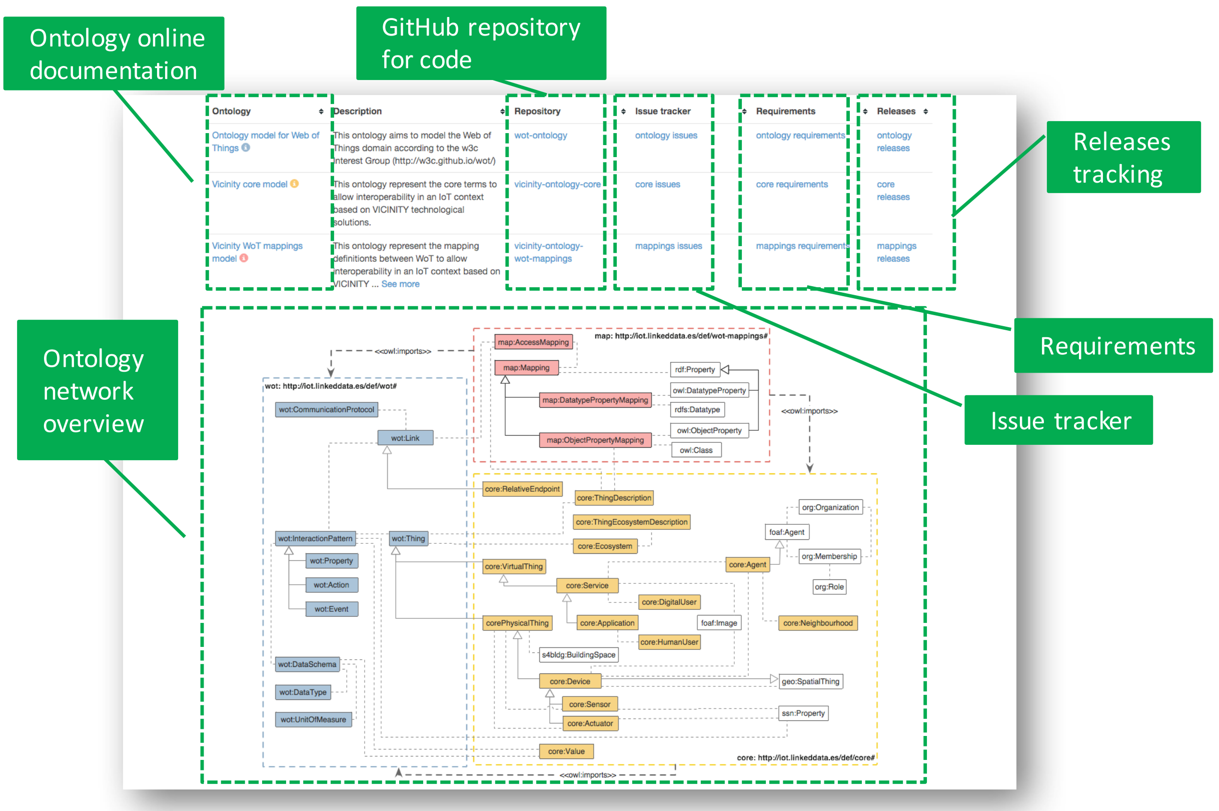
The new requirements which are proposed in a GitHub issue are automatically added to the ORSD associated to the ontology when the ontology development team approves them. The requirement is considered approved when the ontology developers tag the issue as "requirement" and add a comment associating an identifier and a competency question.

6. Overview of the VICINITY ontology network

This section is devoted to the ontology network developed for the VICINITY platform and provides an overview of the network while each ontology module will be described in detail in the following sections.

The ontology network portal is available online at http://vicinity.iot.linkeddata.es/ and, as shown in Figure 22, it provides for each ontology the following information:

- a link to its online documentation
- a link to the GitHub repository in which the code is managed
- a link to the issue tracker
- a link to the online information about requirements
- a link to the list of its releases



Ontology	Description	Repository	Issue tracker	Requirements	Releases
Ontology model for Web of Things	This ontology aims to model the Web of Things domain according to the w3c Interest Group (http://w3c.github.io/wot/)	wot-ontology	ontology issues	ontology requirements	ontology releases
Vicinity core model	This ontology represent the core terms to allow interoperability in an IoT context based on VICINITY technological solutions.	vicinity-ontology-core	core issues	core requirements	core releases
Vicinity WoT mappings model	This ontology represent the mapping definitionts between WoT to allow interoperability in an IoT context based on VICINITY ... See more	vicinity-ontology-wot-mappings	mappings issues	mappings requirements	mappings releases

Figure 22. VICINITY ontology network portal.

The ontology network[22] developed consists so far of three ontology modules, namely VICINITY Core, Web of Things (WoT) and WoT Mappings. Figure 23 provides a graphical overview of the VICINITY ontology network showing the concepts defined in each module and a summary of which concepts are related. The figure also shows the import relation between modules; for example, the Core ontology imports the Web of Things (WoT) ontology. The main hierarchies between concepts are also included. This hierarchies are represented by arrows with white endings (triangles), and is read as follows: the class in the origin of the arrow is a subclass of the class in the end of the arrow.

The ontologies are published under the following URIs:

- Core: http://iot.linkeddata.es/def/core/
- WoT: http://iot.linkeddata.es/def/wot/
- WoT Mappings: http://iot.linkeddata.es/def/wot-mappings/

As it is shown in Figure 23, the ontology modules are related between them. First, ad-hoc relationships are defined between terms belonging to different modules, for example the concept `wot:Link` in the WoT module is somehow related to the `map:AccessMapping` concept in the WoT Mappings ontology. In addition, there is another type of relationship between modules, that is, the `owl:imports` relation. By means of this relation an ontology is included into another ontology; for example, in the VICINITY case, the Core ontology imports the WoT module, this means that the whole content of the WoT module is also part of the Core ontology. It can also be observed that the WoT Mappings ontology imports both the WoT and the Core ontologies.

It is worth noting that the different modules are represented with different color, however this information is also represented by the prefixes attached to each class. In this sense, classes defined in the core ontology are painted in yellow (e.g., `core:VirtualThing`), in the WoT ontology in blue (e.g., `wot:Link`), and in the WoT mappings ontology in pink (e.g., `map:Mapping`). The classes in white with other prefixed are defined in the ontologies indicated by such prefixes. The list of prefixes, and corresponding ontologies, reused along the ontologies and that might appear in this document are listed in Table 4.

It is worth noting that complete and up to date documentation of each ontology module is provided online and is accessible through each of the ontologies' URI. In the rest of this document only the main concepts or modelling decisions are detailed. That is, the main reference for the ontology is available online since the information in this deliverable just contains a snapshot of the current development and will not be updated while the ontology network evolves during the project.

[22] http://vicinity.iot.linkeddata.es/vicinity/

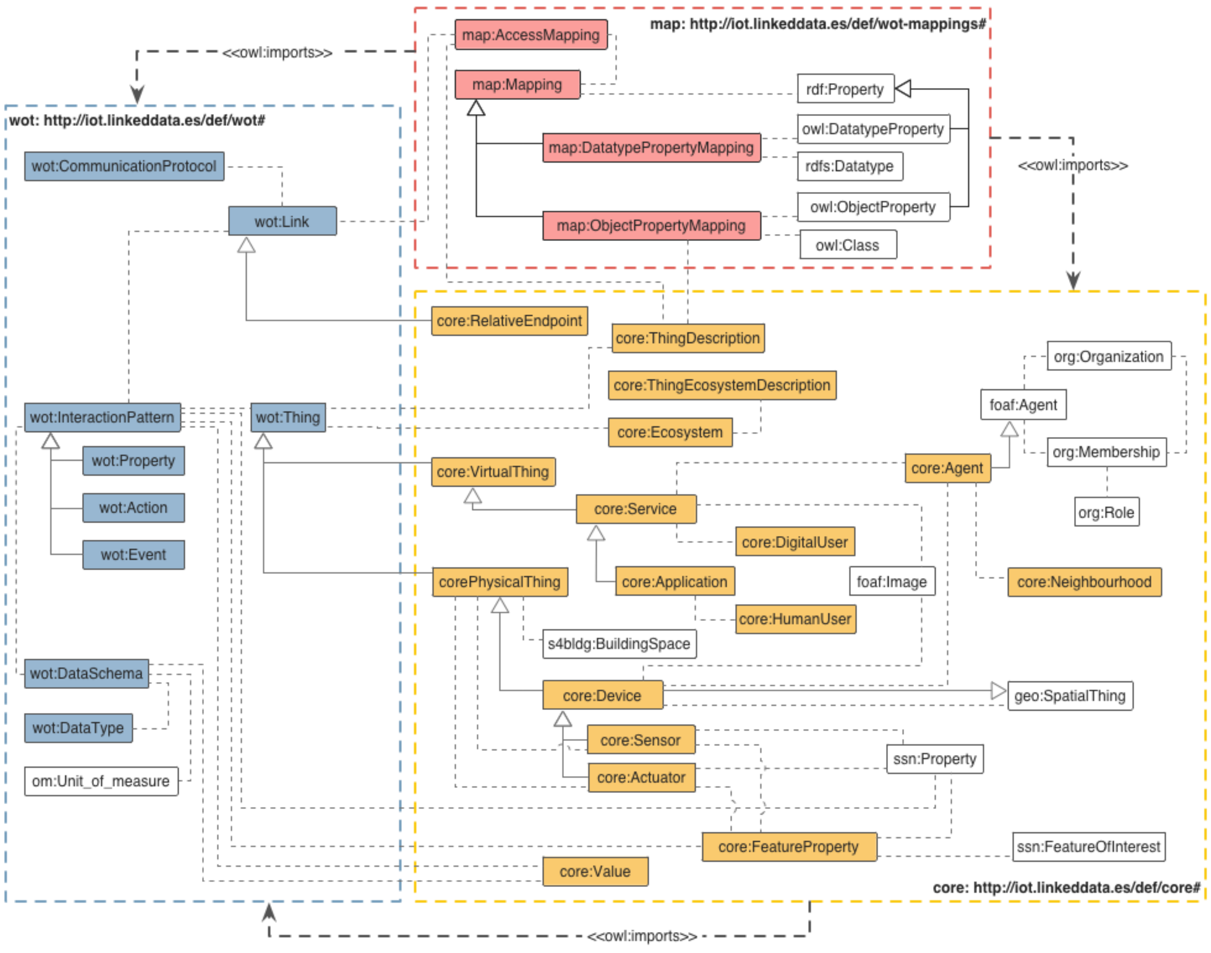


Figure 23. VICINITY ontology network overview.

Prefix	Ontology namespace
foaf	http://xmlns.com/foaf/0.1/
geo	http://www.w3.org/2003/01/geo/wgs84_pos#
om	http://www.wurvoc.org/vocabularies/om-1.8/
org	http://www.w3.org/ns/org#
owl	http://www.w3.org/2002/07/owl#
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs	http://www.w3.org/2000/01/rdf-schema#
ssn	http://purl.oclc.org/NET/ssnx/ssn#
s4bldg	https://w3id.org/def/saref4bldg

Table 4. Prefixes of reused ontologies and their corresponding namespaces.

In order to give an idea of the development status of the ontology network, Table 5 provides an overview of the ontology requirements extracted for each ontology module. This table shows where the requirements have been extracted from and how many of them have been been defined (implemented, discarded and pending).

	Requirements				
Ontology	Extracted from	Defined	Implemented	Discarded	Pending
WoT[23]	• W3C Web of Things IG	35	16	11	8
Core[24]	• D1.5 VICINITY • ISO/IEC IoT RA • Bratislava and later meetings/emails with partners • Partners' devices characterizations	172	84	17	71
WoT Mappings[25]	• Gateway API • Developers	16	15	1	0
		223	**115**	**29**	**79**

Table 5. Overview of the ontology network requirements origin and current status.

Apart from the domain-specific ontological requirements, the VICINITY ontologies development is based on the following non-functional requirements:

- **Reuse**: existing ontologies or standard models will be reused when possible increasing interoperability with external systems that might be already using such ontologies. This point is also applied at a meta-level by using standard technologies to implement the ontologies themselves.

[23] See http://vicinity.iot.linkeddata.es/vicinity/requirements/report-wot.html for the online version of the VICINITY wot ontology requirements.
[24] See http://vicinity.iot.linkeddata.es/vicinity/requirements/report-core.html for the online version of the VICINITY core ontology requirements.
[25] See http://vicinity.iot.linkeddata.es/vicinity/requirements/report-map.html for the online version of the VICINITY mapping ontology requirements.

- **Modularity**: the ontology should be designed as a network in which modules might be interconnected and refer to others.
- **Extensibility**: the ontologies should allow the development of third-party extensions.
- **Good practices**: the ontologies will be developed following methodologies and best practices commonly used in ontological engineering in order to address ontology development activities such as design, implementation, evaluation, publication, and documentation, among others.

Each ontology module will be detailed in the following subsections. In the figures presented in the such subsections and the examples in Section 7 the following graphical conventions are used:

Colored rectangles are used to denote classes created in the ontology being described while white rectangles denote reused classes. For all the entities, it is indicated in which ontology they are defined by the prefix included before their identifier.

Arrows are used represent properties between classes and to represent some rdf, rdfs and owl constructs, more precisely:

- Plain arrows with white triangles represent the `rdfs:subClassOf` relation between two classes. The origin of the arrow is the class to be declared as subclass of the class at the destination of the arrow.
- Plain arrows between two classes indicate that the object property has declared as domain the class in the origin and as range the class in the destination of the arrow. The identifier of the object property is indicated within the arrow.
- Dashed labelled arrows between two classes indicate that the object property can be instantiated between the classes in the origin and the destination of the arrow. The identifier of the object property is indicated within the arrow.
- Dashed arrows with identifiers between stereotype signs (i.e., “<< >>”) refer to OWL constructs that are applied to some ontology elements, that is, they can be applied to classes or properties depending on the OWL construct being used.
- Dashed arrows with no identifier are used to represent the `rdf:type` relation, indicating that the element in the origin of the arrow is an instance of the class in the destination of the arrow.

Datatype properties are denoted by rectangles attached to the classes, in an UML-oriented way. Dashed boxes represent datatype properties that can be applied to the class it is attached to while plain boxes represent that the domain of the datatype property is declared to be the class attached.

Individuals are denoted by rectangles in which the identifier is underlined.

Literals are denoted by rectangles in which the value is included between quotation marks.

The representation of additional property axioms (functional, inverse functional, transitive, and symmetric) that are being used in the diagram are shown in the overview ontology legends.

In addition, each ontology overview picture includes a legend to remind the reader the graphics meaning.

6.1. Web of Things ontology

The Web of Things (WoT) ontology has been developed to define “what", “where" and “how" things can be discovered or accessed in the Web of Things. In this sense, the shared conceptualization to be represented in this ontology is the domain of the Web of Things, that is, it will describe the virtual counterpart of physical objects according to the Web of Thing Model discussed in the W3C.

The current conceptual model defined by the WoT ontology is depicted in Figure 24. This ontology introduces some new concepts closely related to the WoT domain, namely:

- **Thing.** This concept represents anything (both physical and non-physical) which has a distinct and independent existence and can have one or more web representations.
- **Interaction pattern**. This concept represents, in the context of WoT, an exchange of data between a web client and a Thing. This data can be either given as input by the client, returned as output by the Thing or both.
- **Data format**. This concept represents the input data or output data of a given interaction pattern which includes information such as the data type used and which unit of measurement is the data represented in, if needed.
- **Endpoint**. This concept indicates the web location where a service can be accessed by a client application.

The main concepts defined in the ontology, as shown in Figure 24, are `wot:Thing`, `wot:InteractionPattern`, `wot:DataSchema` and `wot:Link` according to the above definitions. It is worth noting that the class `wot:Thing` defines things in the context of the Web of Things and does not intend to be the top class of all possible concepts as `owl:Thing` does. According to the model, a particular thing is linked to the interaction patterns it provides by means of the object property `wot:providesInteractionPattern`. An interaction pattern can be either a property, an action or an event, represented by the concepts `wot:Property`, `wot:Action` and `wot:Event`, respectively.

As shown in Figure 24, a thing or an interaction pattern can be associated to one or more endpoints either directly or through its interaction patterns by means of the object property `wot:isAccessibleThrough`. The main information provided by the endpoint class is about the web location in which the service is provided which is indicated by the attribute `wot:href`. Every endpoint should have a value and only one value for such attribute. Attached to such endpoint the information about the expected media type can be specified by means of the property `wot:isProvidedOverProtocol` which links instances of endpoints to the individuals that represent the possible web protocols.

Finally, some interaction might have input or output data associated, or both; for example, for writable properties. In order to model that, the relationships `wot:hasInputData` and `wot:hasOutputData` were created. These properties allow the connection from a given interaction pattern to an instance that will be linked to a certain data type and a certain unit of measure by means of the properties `wot:isMeasuredIn` and `wot:hasValueType`, respectively. This modelling decision responds to the use of the ontology design pattern for representing n-ary relationships as it is needed to relate the given interaction patterns with both the unit of measure and the expected data type.

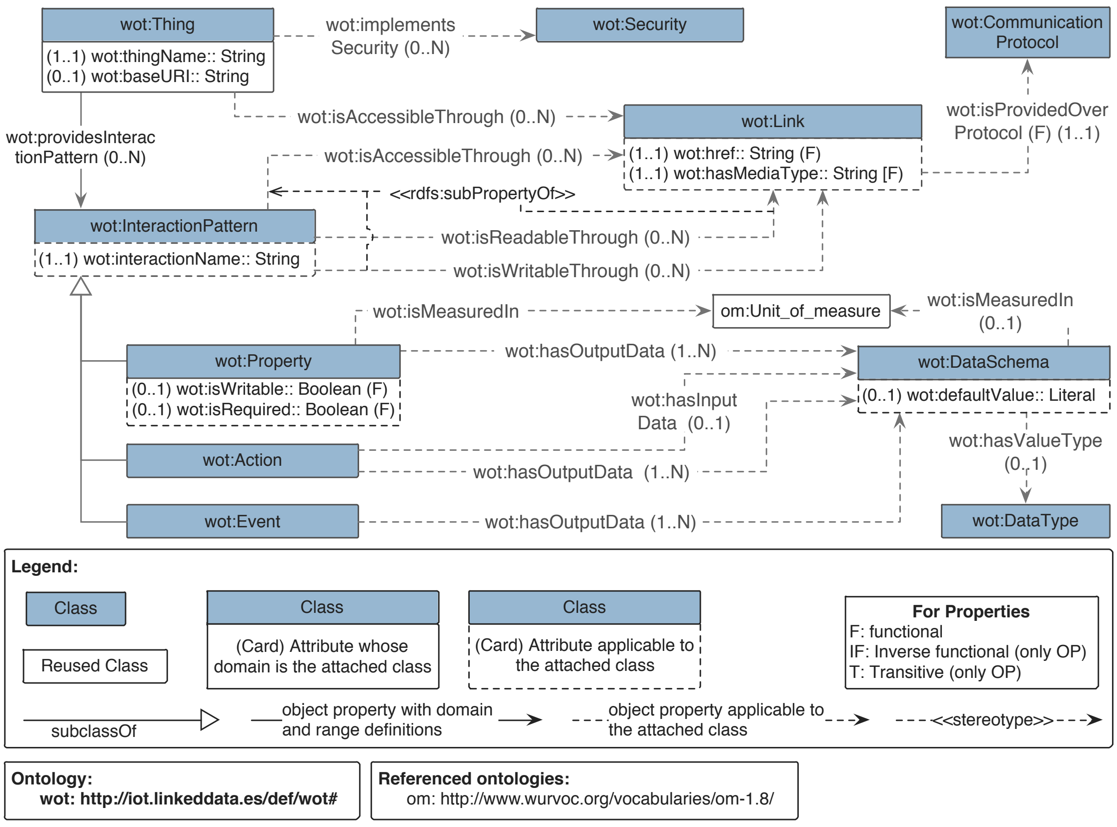

wot:Thing
(1..1) wot:thingName:: String
(0..1) wot:baseURI:: String
wot:implements Security (0..N)
wot:Security
wot:Communication Protocol
wot:isAccessibleThrough (0..N)
wot:Link
(1..1) wot:href:: String (F)
(1..1) wot:hasMediaType:: String [F)
wot:isProvidedOver Protocol (F) (1..1)
wot:providesInteractionPattern (0..N)
wot:isAccessibleThrough (0..N)
<<rdfs:subPropertyOf>>
wot:InteractionPattern
(1..1) wot:interactionName:: String
wot:isReadableThrough (0..N)
wot:isWritableThrough (0..N)
wot:isMeasuredIn
om:Unit_of_measure
wot:isMeasuredIn (0..1)
wot:Property
(0..1) wot:isWritable:: Boolean (F)
(0..1) wot:isRequired:: Boolean (F)
wot:hasOutputData (1..N)
wot:DataSchema
(0..1) wot:defaultValue:: Literal
wot:hasInput Data (0..1)
wot:Action
wot:hasOutputData (1..N)
wot:hasValueType (0..1)
wot:Event
wot:hasOutputData (1..N)
wot:DataType
Legend:
Class
Reused Class
Class
(Card) Attribute whose domain is the attached class
Class
(Card) Attribute applicable to the attached class
For Properties
F: functional
IF: Inverse functional (only OP)
T: Transitive (only OP)
subclassOf
object property with domain and range definitions
object property applicable to the attached class
<<stereotype>>
Ontology:
wot: http://iot.linkeddata.es/def/wot#
Referenced ontologies:
om: http://www.wurvoc.org/vocabularies/om-1.8/

Figure 24. General overview of the WoT ontology.

It should be mentioned that the presented ontology is under development and new concepts might be included or extended. Some ongoing lines of work on the ontology include the modelling of more complex datatypes, to detail security aspects, and to further describe the actions and events as they are defined in the W3C working group.

6.2. VICINITY Core ontology

The VICINITY Core ontology aims at modelling the information needed to exchange IoT descriptor data between peers through the VICINITY platform. This ontology is created following a cross-domain approach; therefore, it could be extended by domain ontologies that would cover vertical domains such as health, transport, buildings, etc. This section provides the documentation for the VICINITY platform-oriented ontology.

The current conceptual model defined by the WoT ontology is depicted in Figure 25. This ontology introduces some new concepts closely related to the WoT domain, namely:

- **Ecosystem.** This concept represents the collection of things that co-exist in a given environment.
- **Thing Ecosystem Description**. This concept represents a digital representation that encapsulates an ecosystem that is accessible via web services.
- **Device**. This concept is defined according to the ISO/IEC Reference Architecture as follows: "An IoT device is a digital entity which bridges between real-world physical entities and the other digital entities of an IoT system. IoT device interacts with one or more networks through which interactions are made with other entities. IoT device exposes one or more endpoints by which interactions are made".
- **Service**. This concept is defined according to the ISO/IEC Reference Architecture as follows: "A service is a set of distinct capabilities provided by a software component through a defined interface, which may be composed of other services. A service is implemented by one or more components. A service defines network interfaces and exposed by an Endpoint".

Apart from these main terms that have been defined for the particular case of the VICINITY platform use case, there are other concepts that are needed in the VICINITY approach to complement the WoT ontology, namely: neighbourhood, sensor, actuator, relative endpoint or value.

For example, it can be observed that the concept `wot:Thing` is specialized in the Core ontology by means of the concepts `core:VirtualThing` and `core:PhysicalThing`. Such concepts are further specialized by `core:Service` and `core:Device` supporting the use case of the VICINITY registration of devices and services and aligning the modelling with the ISO/IEC RA standard.

The VICINITY platform is based on the idea of retrieving the set of web things that might be useful to answer a given request made to the system. In this sense, the idea of collections of things needs to be represented. For this reason, the class `core:Ecosystem` is included in the Core ontology. It gathers together the set of things that might be used in a query by means of the property `core:hasComponent`, which links ecosystems to the web things, and its inverse `core:isComponentOf`.

Another example of specialization is the case of `core:RelativeEndpoint` which extends the definition of the `wot:Link` concept for those cases in which an endpoint is defined in a relative way to another endpoint. This particular situation that is allowed in the VICINITY platform is not taken into account in the WoT Working Group specification (at least at the moment of writing this document); therefore, it is included as part of the Core ontology.

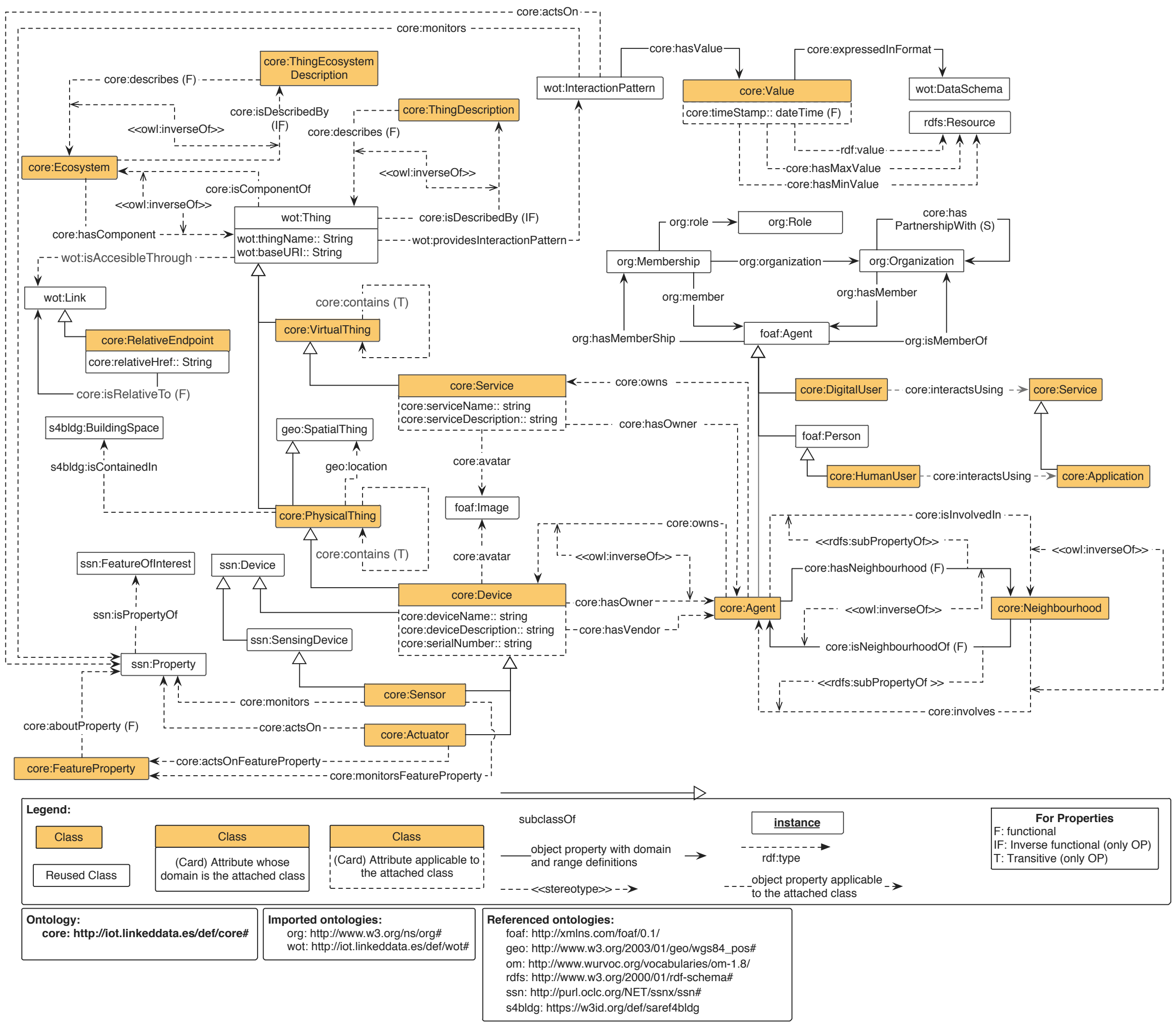


Figure 25. General overview of the VICINITY Core ontology.

Another requirement raised by VICINITY use cases is to represent the geolocation of devices, as it might be needed to ask for all the devices in a given pilot site. For this reason, the wgs84 geo positioning vocabulary[26] and the SAREF4BLDG ontology[27] has been reuse. From the wgs84 vocabulary the class `geo:SpatialThing` is reused to represent anything with spatial extent. In order to represent geographical coordinates associated to a spatial thing, the vocabulary defines the properties `geo:lat`, `geo:long` and `geo:alt`. In addition, the property `geo:location` allows

[26] https://www.w3.org/2003/01/geo/wgs84_pos
[27] https://w3id.org/def/saref4bldg

the linking from any entity to a spatial thing where the coordinates of such entity can be represented. Besides, the class `s4bldg:BuildingSpace` and the relationship `s4bldg:isContainedIn` are reused to allow the representation of devices being located in specific building spaces. It should be mentioned that the building spaces are not defined in detail in such ontology. Therefore, it is used as a bridge term just to represent individuals belonging to that class that would be further described by using other ontologies.

In addition, it is foreseen within the VICINITY use cases the retrieval of values from the web services where the web things expose their interaction patterns. For example, some user might ask about the latest value of a given property. For this reason, the class `core:Value` and the relationship `core:hasValue` have been incorporated to the ontology. This class is intended to represent a given value by means of the property `rdf:value` or a range of values using `core:hasMinValue` and `core:hasMaxValue`, in a given point in time (`core:timeStamp`). It is also possible to link to the data schema in which the value is represented through the property `core:expressedInFormat`.

Finally, as the VICINITY platform manages the concept of neighbourhoods, it is represented in the Core ontology. More precisely, agents, represented by `core:Agent`, are related among them in neighbourhoods. For doing so, the relations `core:isInvolvedIn`, `core:involves`, `core:hasNeighbourhood` and `core:isNeighbourhoodOf` have been defined. In this sense, an instance of `core:Neighbourhood` will point to all the agents belonging to such neighbourhood through the relation `core:involves` and, accordingly, all the agents involved in the neighbourhood will point to it through the inverse property `core:isInvolvedIn`. Each neighbourhood is built taking an agent as its central point. For this reason, there is a specific relation between such agent and the neighbourhood being represented. This relation is implemented by means of the properties `core:hasNeighbourhood` (stated from the agent to the neighbourhood) and its inverse `core:isNeighbourhoodOf` (stated from the neighbourhood to the agent).

Since this ontology builds on top and extends the WoT ontology, the latter is imported into the Core ontology by means of the `owl:imports` statement. In addition, the Organization ontology[28] of the W3C is also imported in order to support the representation of organizations, roles and memberships.

The `core:Device` concept is intended to be extended according to the given use cases both within VICINITY and in external projects that might reuse the VICINITY ontology. In order to support the use cases and pilot defined by VICINITY a preliminary hierarchy of devices has been defined. As it can be observed in Figure 26, main subclasses of devices are `core:Sensor` and `core:Actuator`. However, when a device is not easily classified under at least one of these classes, it can be classified directly under `core:Device`.

[28] http://www.w3.org/ns/org#

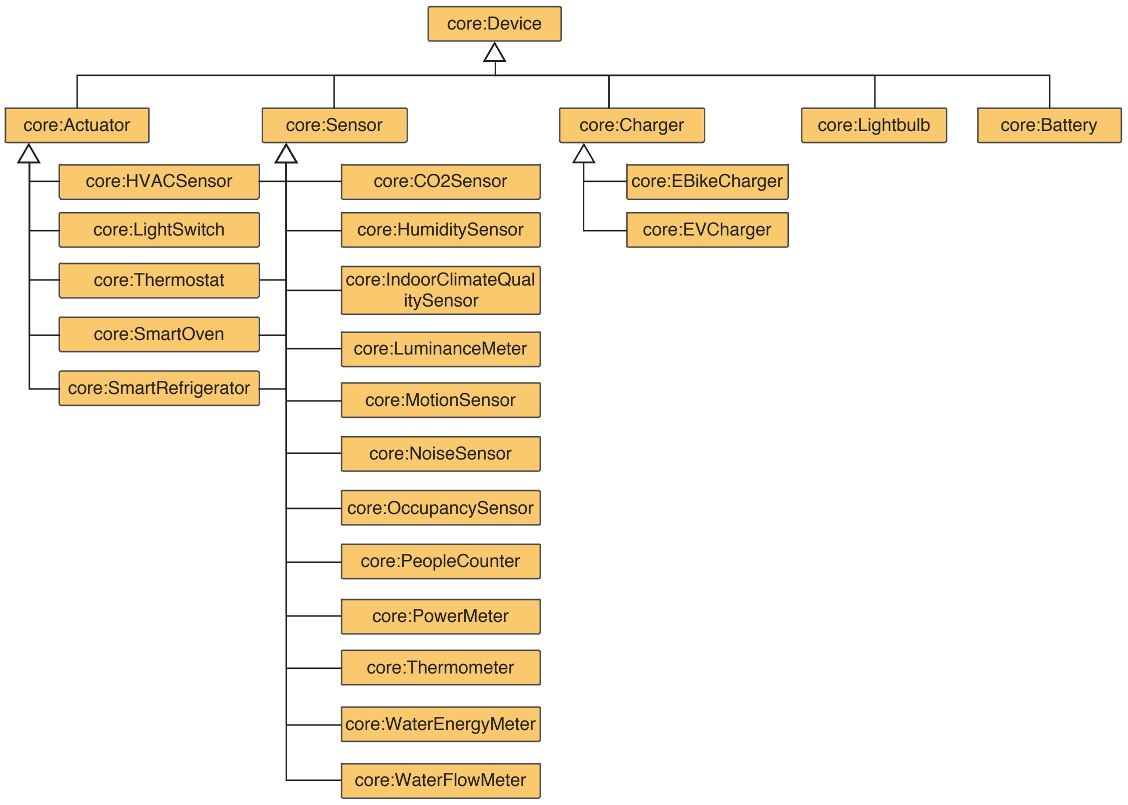


Figure 26. Device hierarchy for VICINITY use cases.

Most of the devices have the ability of observing or acting, or both, over one or more properties. It is worth mentioning that in this context the meaning of "property" is understood as an observable quality of a thing. In order to create a preliminary list of such properties, those that can be observed or modified by the devices that will be used in the release 0.1 of the VICINITY platform have been included in the ontology. These properties have been classified under the `ssn:Property` reused class. The resulting hierarchy including classes and instances is shown in Figure 27.

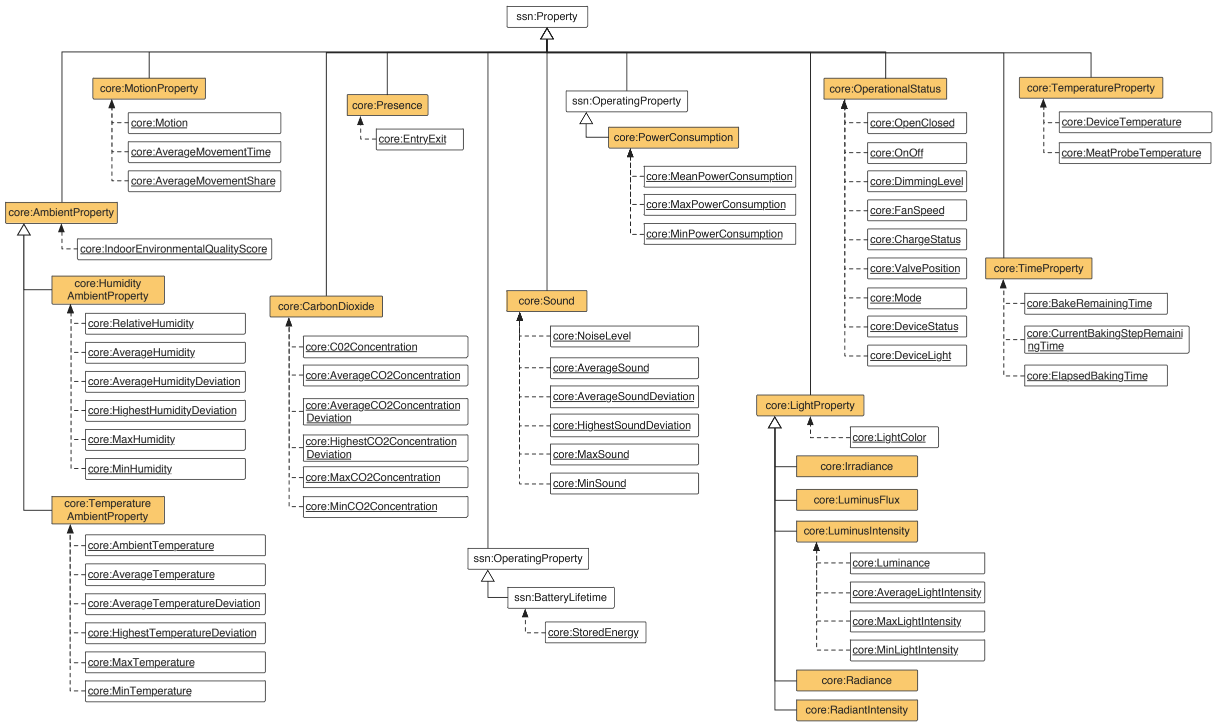


Figure 27. Hierarchy of properties and exemplary instances.

In some cases, it will be needed to represent that a sensor or an interaction pattern (property or event) monitors a given property (as defined in the SSN ontology) of a particular feature of interest instead of a generic one. This also could be the case for actuators or actions acting on properties of particular features of interest. In order to allow the representation of this information being able of unequivocally identify to which pair of "property-feature of interest" is being monitored/affected, it is needed to include an auxiliary class following a n-ary class pattern. In this sense, if a device observes 3 properties about 2 features of interest, it will be possible to identify which particular properties are observed for each feature of interest. Figure 28 illustrates the model proposed for this n-ary pattern represented by the class `core:FeatureProperty`. It is worth noting that the following property chains[29] have been defined:

- `core:monitorsFeatureProperty`, `core:aboutProperty` → `core:monitors`
- `core:actsOnFeatureProperty`, `core:aboutProperty` → `core:actsOn`

[29] https://www.w3.org/TR/owl2-rdf-based-semantics/#Semantic_Conditions_for_Sub_Property_Chains

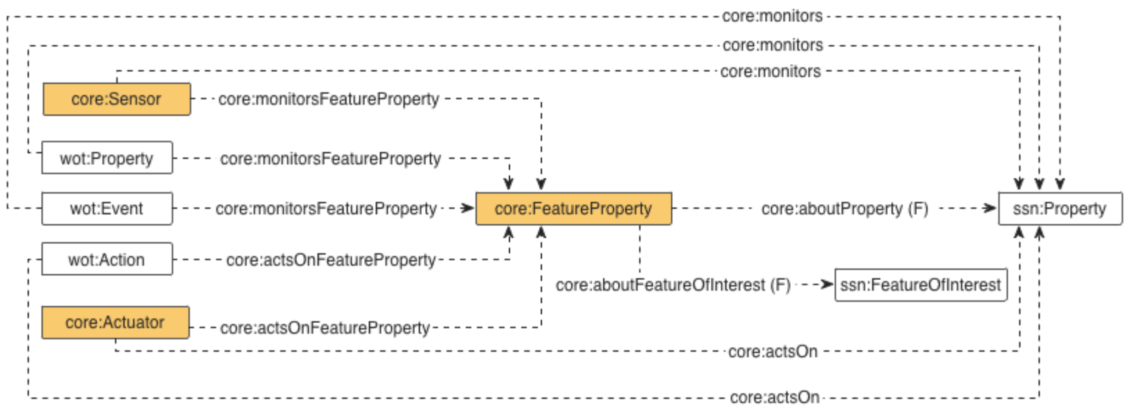


Figure 28. N-ary pattern to specify which property is monitored/affected about a specific feature of interest.

Finally, the VICINITY Core ontology takes advantage of the reuse of the SSN ontology in order to represent some device characteristics. In this case, the representation of a value for a given device property is made through the class `core:Value` (already explained for the use case of values provided for a given property through a web service) and the relationship `ssn:hasValue` as shown in Figure 29.

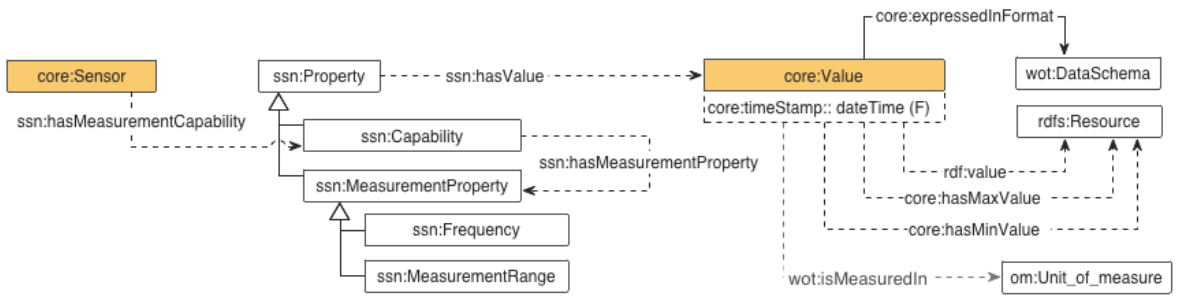


Figure 29. Submodel used for describing sensor capabilities.

6.3. WoT Mappings ontology

Not all thing attributes can be expressed in a static and shareable description; mainly because they are dynamic, protected or both. For instance, the geo-location of certain physical things can be considered as sensitive and only be obtained under specific security and privacy constraints, through its endpoints. Besides, its value may dynamically vary as the physical thing changes its position. Therefore, if this casuistry is not taken into account, the location-based discovery of such kind of things will not be possible.

A solution to this would involve describing as well how data provided by secured endpoints map to specific thing attributes. By following this approach, descriptions might inform on how to automatically and securely retrieve and map their own missing attribute values, by means of what it is called *access mappings*.

The adoption of access mappings in the data model leads to a wider scope solution: to gather values for any kind of thing attributes from its own web interfaces, significantly extending the support to interoperability in the IoT ecosystem. In order to achieve this, data models for web things should also support describing the exchanged data with the mentioned links or endpoints, i.e., they should not just describe its format but also its content. Thus, rather than expecting to receive data from endpoints in a specific syntax, descriptions would inform consumers on how to process responses and extract useful information.

For instance, the description of a temperature sensor may state that the data received after invoking an endpoint contains the latest measured value in Celsius and where to find such value in the response. Thanks to this, discovery clients might be able to issue search criteria for things measuring temperature in Celsius and get, extract and interpret values from the discovered things' endpoints.

This VICINITY approach for semantic discovery has been taken into account within the VICINITY ontology models. More precisely, the WoT Mappings ontology, which is described in this section, aims at providing support for the semantic description of such mappings.

The conceptualization to be represented in the WoT Mappings ontology is the mechanism for accessing the values provided by web things. In this sense what is needed is to represent the mappings between the values provided under a given endpoint for example in JSON format to common semantic vocabularies.

The current conceptual model defined by the WoT Mappings ontology is depicted in Figure 30. In order to model this information, it should be first established what does a mapping mean in this context:

- **Mapping.** A mapping indicates the relation between a given key (provided as structured data in an on-line resource) and the RDF property to which the values should be mapped and the target type of object.

Taking this definition as starting point and together with sample data, the ontology shown in Figure 30 was designed. The main concepts defined in such ontology are `map:Mapping` and `map:AccessMapping`. The former corresponds to the mapping concept above-defined allowing the connection between a key provided within structured data in an on-line resource, represented by the datatype property `map:key`, to the RDF property to which it should be mapped to, represented by the object property `map:predicate`.

The instances of the class `map:Mapping` can be further classified into one of its two subclasses, `map:ObjectPropertyMapping` and `map:DatatypePropertyMapping`, depending on whether the predicate attached to them is an `owl:ObjectProperty` or an `owl:DatatypeProperty`, respectively. As it can be observed, `map:Mapping` is defined as the disjoint union of both subclasses, since an instance of `map:Mapping` can belong to any of the subclasses but only can belong to one of them.

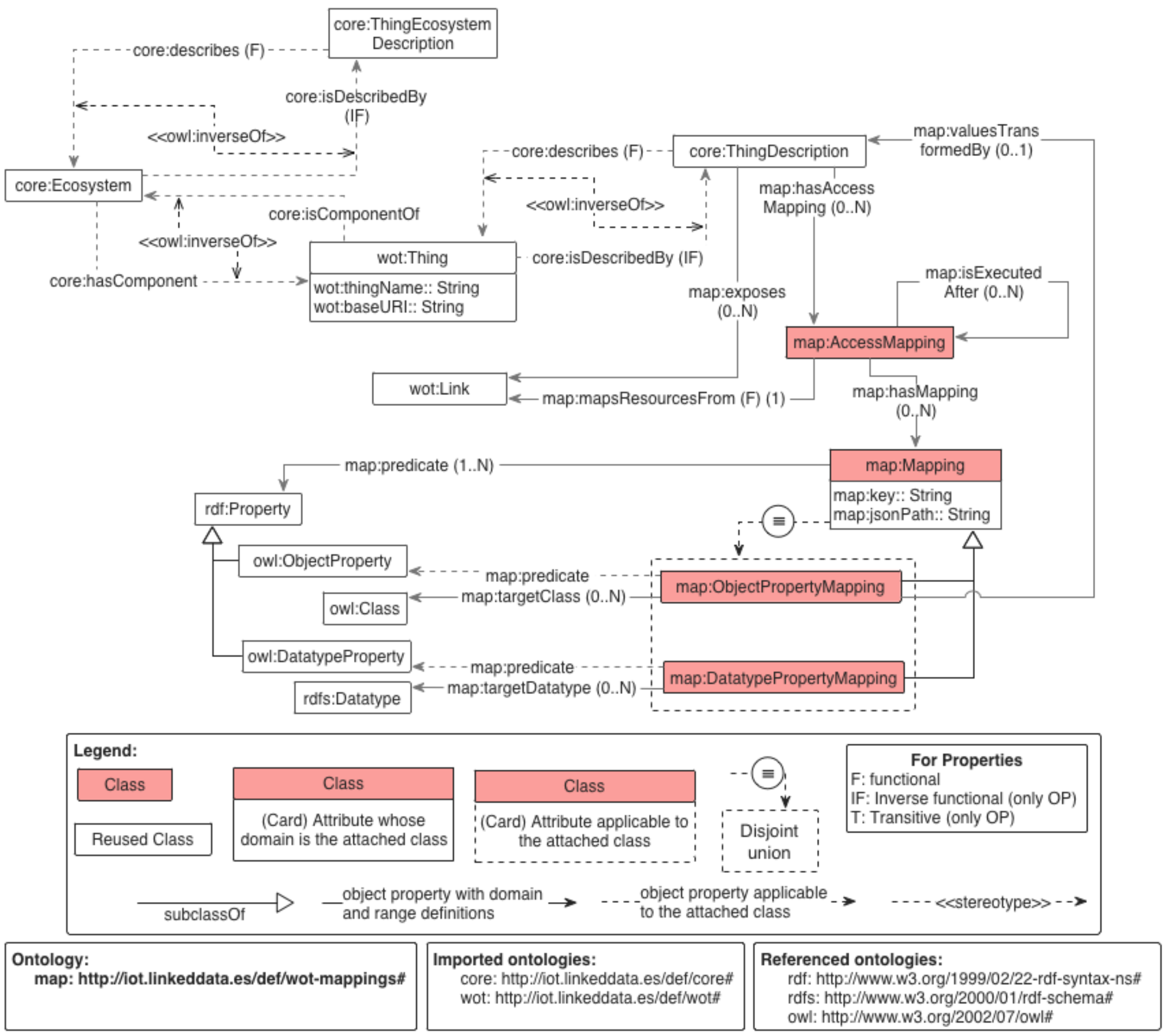

core:ThingEcosystem Description
core:describes (F)
core:isDescribedBy (IF)
<<owl:inverseOf>>
core:Ecosystem
core:isComponentOf
<<owl:inverseOf>>
core:hasComponent
wot:Thing
wot:thingName:: String
wot:baseURI:: String
core:describes (F)
core:ThingDescription
map:valuesTransformedBy (0..1)
map:hasAccessMapping (0..N)
<<owl:inverseOf>>
core:isDescribedBy (IF)
map:exposes (0..N)
map:isExecutedAfter (0..N)
map:AccessMapping
wot:Link
map:mapsResourcesFrom (F) (1)
map:hasMapping (0..N)
map:predicate (1..N)
map:Mapping
map:key:: String
map:jsonPath:: String
rdf:Property
owl:ObjectProperty
map:predicate
map:targetClass (0..N)
owl:Class
map:ObjectPropertyMapping
owl:DatatypeProperty
map:predicate
map:targetDatatype (0..N)
rdfs:Datatype
map:DatatypePropertyMapping
Legend:
Class
Reused Class
Class
(Card) Attribute whose domain is the attached class
Class
(Card) Attribute applicable to the attached class
Disjoint union
For Properties
F: functional
IF: Inverse functional (only OP)
T: Transitive (only OP)
subclassOf
object property with domain and range definitions
object property applicable to the attached class
<<stereotype>>
Ontology:
map: http://iot.linkeddata.es/def/wot-mappings#
Imported ontologies:
core: http://iot.linkeddata.es/def/core#
wot: http://iot.linkeddata.es/def/wot#
Referenced ontologies:
rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs: http://www.w3.org/2000/01/rdf-schema#
owl: http://www.w3.org/2002/07/owl#

Figure 30. General overview of the VICINITY WoT Mappings ontology.

As it can be observed in the figure, another difference between the subclasses of `map:Mapping` is the target element expected for the values transformed; for the case of `map:ObjectPropertyMapping` the expected target should be an instance of `owl:Class` while for the case of `map:DatatypePropertyMapping` it should map values to instances of the class `rdfs:Datatype`. The mappings are linked to these target elements by means of the properties `map:targetClass` and `map:targetDatatype`, respectively.

The class `map:AccessMapping` is included in the model in order to link one or more mappings that are executed with a given endpoint (represented by `wot:Link`). This allows the definition of the mappings independently of the endpoint in which they can be executed since the link to the endpoint is established from the access mapping. A thing description, represented by the class `core:ThingDescription`, may have zero or more access mappings attached by means of the object property `map:hasAccessMapping`. The object property `map:isExecutedAfter` indicates dependency on the order of execution between access mappings.

Finally, the object property `map:valuesTransformedBy`, which can only be applied to `map:ObjectPropertyMapping` instances, is used to state that the obtained values from a resource when applying a mapping should be transformed according to the referenced `core:ThingDescription`. This predicate is oriented to support the WoT discovery feature proposed at the W3C IG, taken literally "*The relationship between things provides a further basis for discovery. The relationships are defined through the models for things, where a thing has properties whose values are other things*."[30]

[30] See for further information https://www.w3.org/WoT/IG/wiki/Discovery_TF

7. Example of use of the VICINITY modules

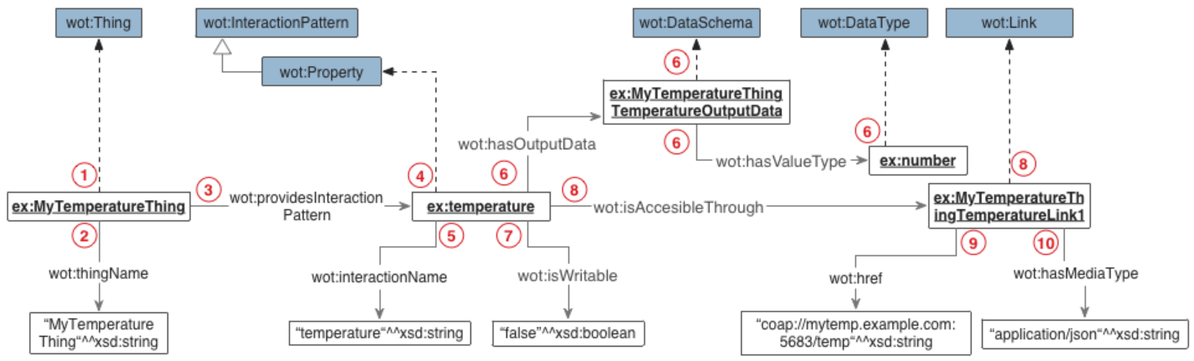
It is always advisable to provide examples that ease the ontology population activity to potential (re)users. For this reason, this chapter includes some examples about how to use the three ontologies above-described.

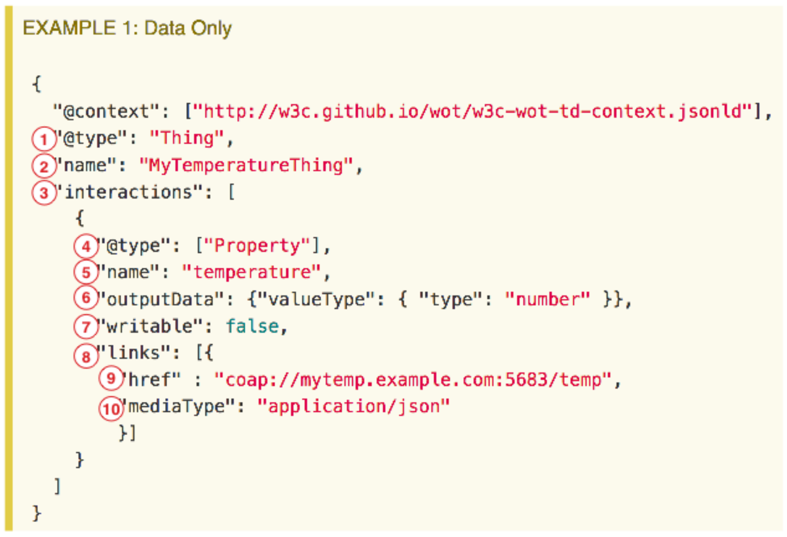
For the WoT ontology, some examples are extracted from the WoT Current Practices document [19] provided by the W3C WoT Interest Group[31] while for the core and mappings ontologies the examples are extracted from different use cases.

7.1. Example 1: Data only

The example "*Example 1: Data only*" provided by the W3C WoT Interest Group is depicted in Figure 31. At the top of the figure a graph showing the RDF triples that would represent the information included in the JSON excerpt (at the bottom of the figure) is shown. It should be noted that the numbers in the JSON excerpt might be related to one or more triples in the RDF graph, for example, the number 6 involves 4 triples in RDF. In this case, in order to represent the thing description an instance of the class `wot:Thing` is created (triple labelled with "1"). Such instance has a name associated by means of the datatype property `wot:thingName` (triple labelled with "2"). In order to link the web thing to the interaction pattern it provides, the object property `wot:providesInteractionPattern` is used having as subject the instance representing the web thing, and as object a new instance representing the property interaction pattern "temperature" (triple labelled with "3"). It is worth mentioning that in order to represent the output data schema provided by the interaction pattern a n-ary pattern must be instantiated. In this sense, the relation between the interaction pattern and the output data leads to more than one triple, more precisely those labelled with "6". First, the interaction pattern is linked to an instance of `wot:DataSchema` and then this latter instance is linked to the individual `ex:number`. The instance of `wot:DataSchema` could also be linked to the unit of measure in which the data is provided or to a default value. Finally, the interaction pattern instance is linked to the endpoint such property is provided by means of the object property `wot:isAccesibleThrough` (triple labelled with "8"). The instance representing the endpoint in which the property is provided is linked to the value of the web address or service in which it is provided (triple labelled with "9") and the media type in which the information is retrieved (triple labelled with "10").

[31] https://www.w3.org/WoT/IG/





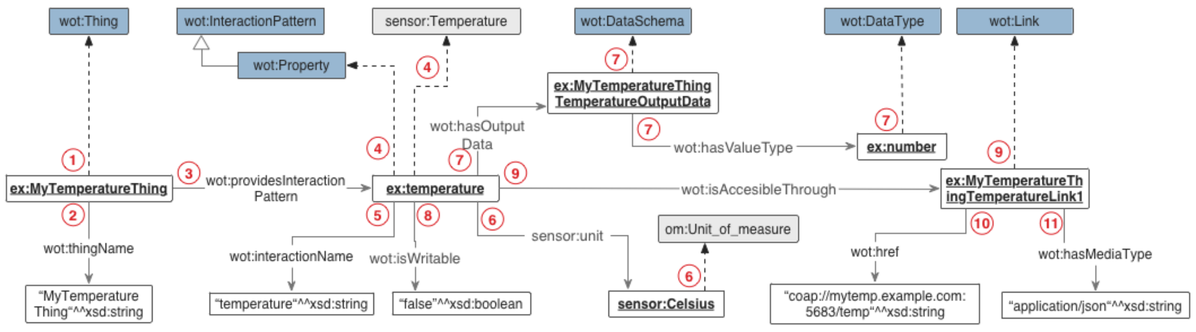
EXAMPLE 1: Data Only

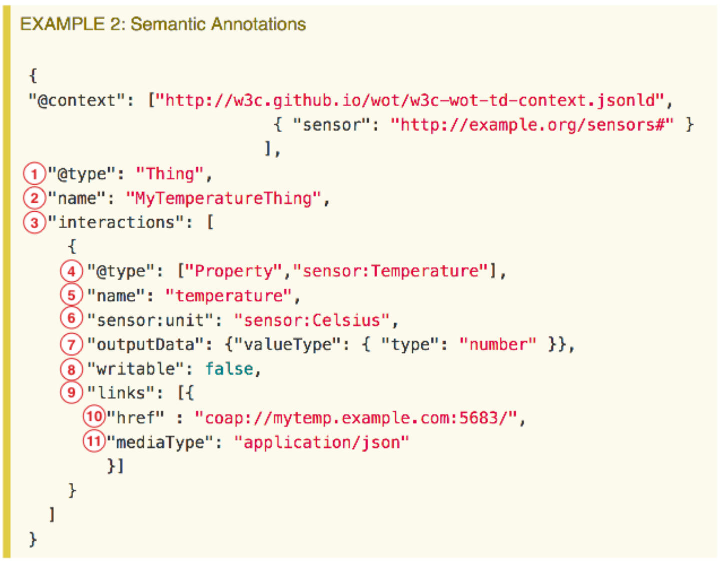
```
{
  "@context": ["http://w3c.github.io/wot/w3c-wot-td-context.jsonld"],
  "@type": "Thing",
  "name": "MyTemperatureThing",
  "interactions": [
    {
      "@type": ["Property"],
      "name": "temperature",
      "outputData": {"valueType": { "type": "number" }},
      "writable": false,
      "links": [{
        "href" : "coap://mytemp.example.com:5683/temp",
        "mediaType": "application/json"
      }]
    }
  ]
}
```

Figure 31. Example of a Thing interaction pattern from only data example.

7.2. Example 2: Semantic Annotations

Another example extracted from the W3C WoT Interest Group's Current Practices document is the "*Example 2: Semantic Annotations*", shown in Figure 32. In this case, apart of being annotated by the WoT ontology, the data is annotated also using a sensor ontology and an ontology for representing units of measure as indicated by the classes `sensor:Temperature` and `om:Unit_of_measure`, respectively.




EXAMPLE 2: Semantic Annotations

```
{
"@context": ["http://w3c.github.io/wot/w3c-wot-td-context.jsonld",
                         { "sensor": "http://example.org/sensors#" }
                        ],
①"@type": "Thing",
②"name": "MyTemperatureThing",
③"interactions": [
    {
   ④"@type": ["Property","sensor:Temperature"],
   ⑤"name": "temperature",
   ⑥"sensor:unit": "sensor:Celsius",
   ⑦"outputData": {"valueType": { "type": "number" }},
   ⑧"writable": false,
   ⑨"links": [{
     ⑩"href" : "coap://mytemp.example.com:5683/",
     ⑪"mediaType": "application/json"
       }]
    }
  ]
}
```

Figure 32. Example of a Thing interaction pattern from data and semantics example.

7.3. Example 3: More Capabilities

The last example taken from the W3C WoT Interest Group's Current Practices document is the "*Example 3: More Capabilities*" and it is depicted in Figure 33. In this example, four interaction patterns associated to a LED are described in detail. For example, the value types provided as output data or expected as input data are provided, also the unit of measure in which the data is expressed or the media type used in the online services.

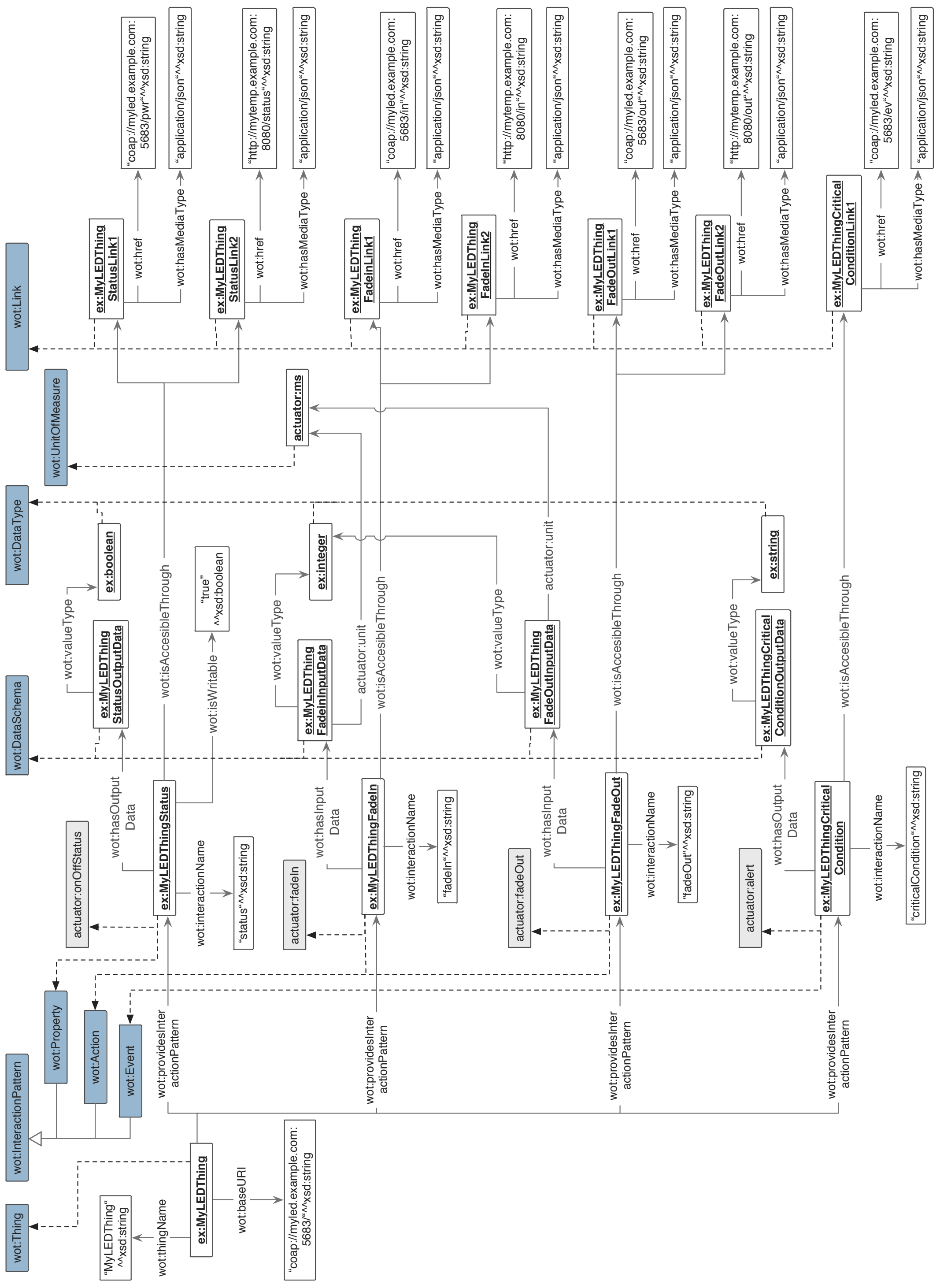

wot:Thing
wot:InteractionPattern
wot:Property
wot:Action
wot:Event
wot:DataSchema
wot:DataType
wot:UnitOfMeasure
wot:Link
ex:MyLEDThing
wot:thingName
"MyLEDThing"^^xsd:string
wot:baseURI
"coap://myled.example.com:5683/"^^xsd:string
wot:providesInteractionPattern
ex:MyLEDThingStatus
actuator:onOffStatus
wot:interactionName
"status"^^xsd:string
wot:hasOutputData
ex:MyLEDThingStatusOutputData
wot:valueType
ex:boolean
wot:isWritable
"true"^^xsd:boolean
wot:isAccesibleThrough
ex:MyLEDThingStatusLink1
ex:MyLEDThingStatusLink2
wot:href
wot:hasMediaType
"coap://myled.example.com:5683/pwr"^^xsd:string
"http://mytemp.example.com:8080/status"^^xsd:string
"application/json"^^xsd:string
ex:MyLEDThingFadeIn
actuator:fadeIn
"fadeIn"^^xsd:string
wot:hasInputData
ex:MyLEDThingFadeInInputData
ex:integer
actuator:unit
actuator:ms
ex:MyLEDThingFadeInLink1
ex:MyLEDThingFadeInLink2
"coap://myled.example.com:5683/in"^^xsd:string
"http://mytemp.example.com:8080/in"^^xsd:string
ex:MyLEDThingFadeOut
actuator:fadeOut
"fadeOut"^^xsd:string
ex:MyLEDThingFadeOutInputData
ex:MyLEDThingFadeOutLink1
ex:MyLEDThingFadeOutLink2
"coap://myled.example.com:5683/out"^^xsd:string
"http://mytemp.example.com:8080/out"^^xsd:string
ex:MyLEDThingCriticalCondition
actuator:alert
"criticalCondition"^^xsd:string
ex:MyLEDThingCriticalConditionOutputData
ex:string
ex:MyLEDThingCriticalConditionLink1
"coap://myled.example.com:5683/ev"^^xsd:string

Figure 33. Example of interaction patterns for a LED.

7.4. Example 4: Relative Endpoints

The next example shows how to use the WoT ontology in combination with the Core ontology in order to represent *relative endpoints*, which is a VICINITY particularity. More precisely, Figure 34 shows how the class `core:RelativeEndpoint` is instantiated and how its instances are linked to a non-relative endpoint by means of the object property `core:isRelativeTo`. In addition, an external ontology to represent photometers is used.

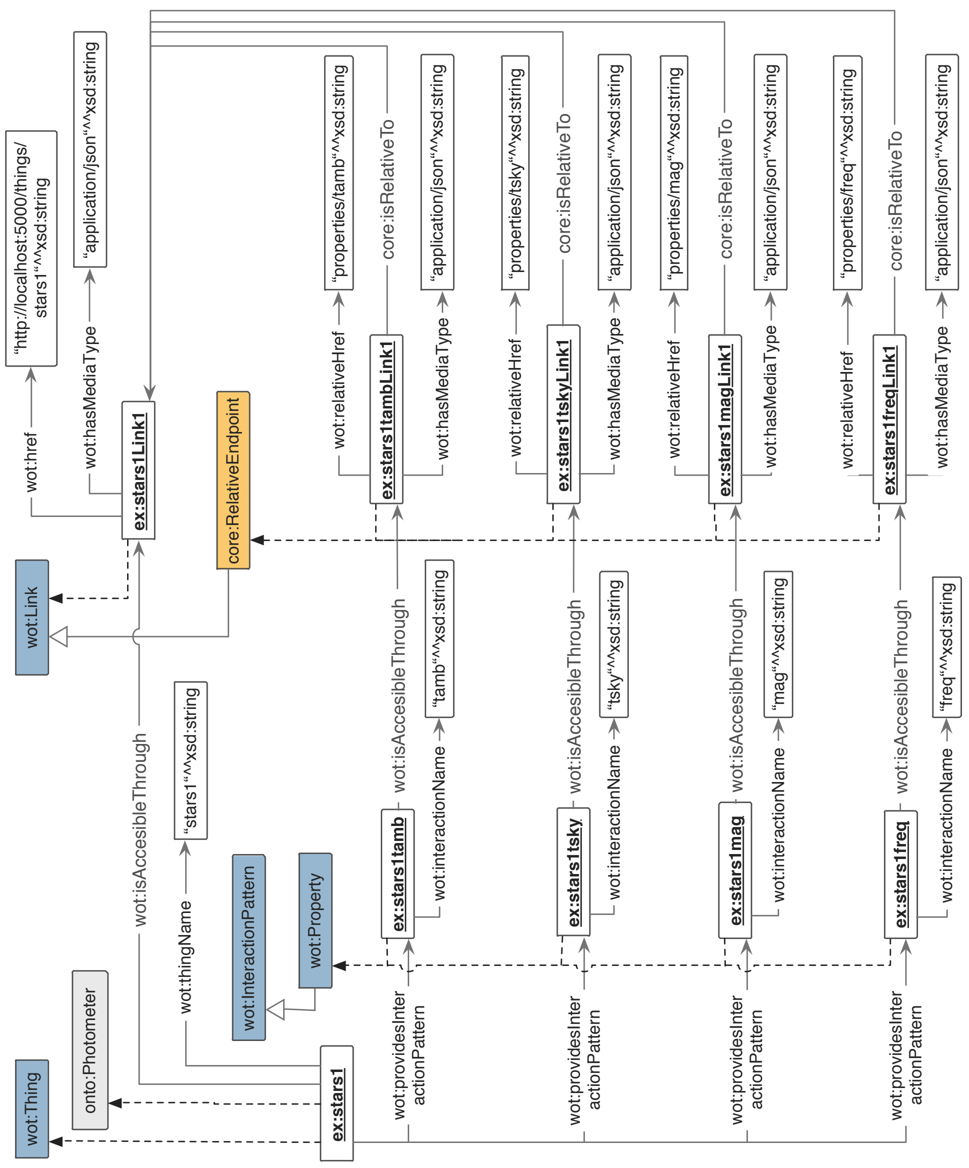


Figure 34. Example of relative endpoints.

7.5. Example 5: Temperature sensor description

The next example shows how to *describe a given sensor*, in this case a thermometer, using the Core ontology and the WoT module. The example is graphically represented in Figure 35. The particular thermometer characteristics are the following:

- **Name**: Temperature Sensor 01
- **Type**: Thermometer
- **GUID**: 5072dd0b-c2f0-4744-9dd4-dff344d8e2bb
- **Properties**:
    - **Temperature**:
        - **Name**: Temperature
        - **Observed Property**: Average temperature
        - **Datatype**: Double
        - **Writable**: no
        - **Units of measure**: ºC
- **Events**:
    - **Temperature**:
        - **Name**: Temperature New Value
        - **Observed Property**: Average temperature
- **Capabilities**:
    - **Frequency**: 100 seconds
    - **Resolution**: 0.3 ºC
    - **Accuracy**: 1 ºC
    - **Measurement range**: 0 - 40 ºC

It is worth noting that the example also represents how to use the SSN ontology to describe device characteristics such as the frequency, resolution or accuracy. This representation is combined with the pattern shown in Section 6.2.

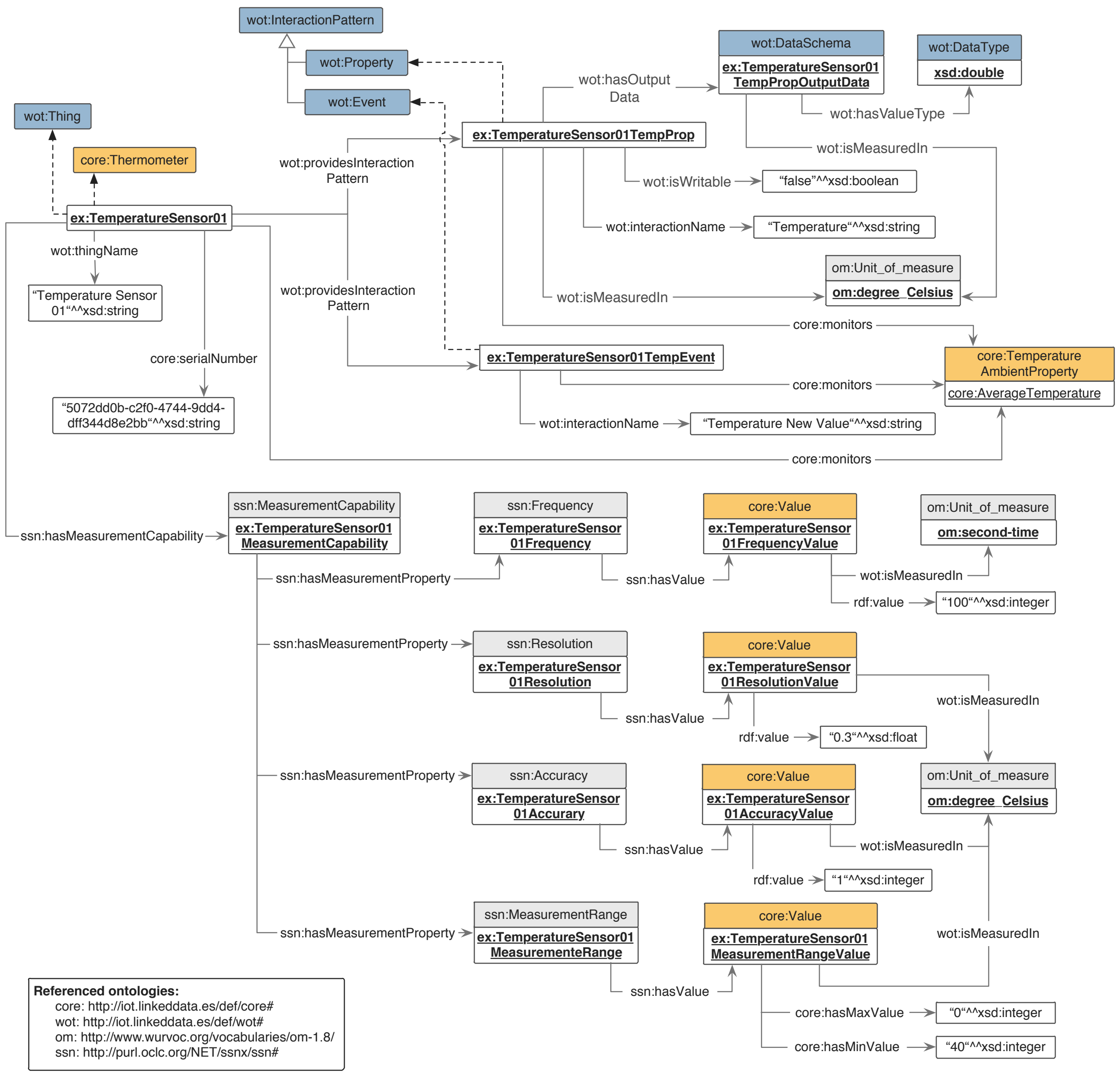

wot:InteractionPattern
wot:Property
wot:Event
wot:Thing
core:Thermometer
ex:TemperatureSensor01
wot:thingName
"Temperature Sensor 01"^^xsd:string
core:serialNumber
"5072dd0b-c2f0-4744-9dd4-dff344d8e2bb"^^xsd:string
wot:providesInteraction Pattern
wot:providesInteraction Pattern
ex:TemperatureSensor01TempProp
wot:hasOutput Data
wot:DataSchema
ex:TemperatureSensor01 TempPropOutputData
wot:DataType
xsd:double
wot:hasValueType
wot:isMeasuredIn
wot:isWritable
"false"^^xsd:boolean
wot:interactionName
"Temperature"^^xsd:string
om:Unit_of_measure
om:degree_Celsius
wot:isMeasuredIn
core:monitors
ex:TemperatureSensor01TempEvent
core:monitors
core:Temperature AmbientProperty
core:AverageTemperature
wot:interactionName
"Temperature New Value"^^xsd:string
core:monitors
ssn:hasMeasurementCapability
ssn:MeasurementCapability
ex:TemperatureSensor01 MeasurementCapability
ssn:Frequency
ex:TemperatureSensor 01Frequency
core:Value
ex:TemperatureSensor 01FrequencyValue
om:Unit_of_measure
om:second-time
ssn:hasMeasurementProperty
ssn:hasValue
wot:isMeasuredIn
rdf:value
"100"^^xsd:integer
ssn:hasMeasurementProperty
ssn:Resolution
ex:TemperatureSensor 01Resolution
core:Value
ex:TemperatureSensor 01ResolutionValue
ssn:hasValue
wot:isMeasuredIn
rdf:value
"0.3"^^xsd:float
ssn:hasMeasurementProperty
ssn:Accuracy
ex:TemperatureSensor 01Accurary
core:Value
ex:TemperatureSensor 01AccuracyValue
om:Unit_of_measure
om:degree_Celsius
ssn:hasValue
wot:isMeasuredIn
rdf:value
"1"^^xsd:integer
ssn:hasMeasurementProperty
ssn:MeasurementRange
ex:TemperatureSensor01 MeasurementeRange
core:Value
ex:TemperatureSensor01 MeasurementRangeValue
wot:isMeasuredIn
ssn:hasValue
core:hasMaxValue
"0"^^xsd:integer
core:hasMinValue
"40"^^xsd:integer
Referenced ontologies:
core: http://iot.linkeddata.es/def/core#
wot: http://iot.linkeddata.es/def/wot#
om: http://www.wurvoc.org/vocabularies/om-1.8/
ssn: http://purl.oclc.org/NET/ssnx/ssn#

Figure 35. Thermometer description example.

7.6. Example 6: Mappings for geo location

In order to show how to use the WoT *Mappings ontology* in combination with the Core and WoT modules, the *example* in Figure 36 is provided. In this case, a thing description can be linked to access mappings using the object property `map:hasAccessMapping`. The `map:AccessMapping` class is instantiated to represent two mappings, which are represented by individuals of the class `map:Mapping`. These mappings indicate how to transform specific field values from a JSON document, indicated by the object property `map:key`, into RDF triples of the properties `geo:lat` and `geo:long`, indicated by the object property `map:predicate`. The endpoint from which the JSON structures would be retrieved from is indicated from the access mapping instance by means of the object property `map:mapResourcesFrom`.

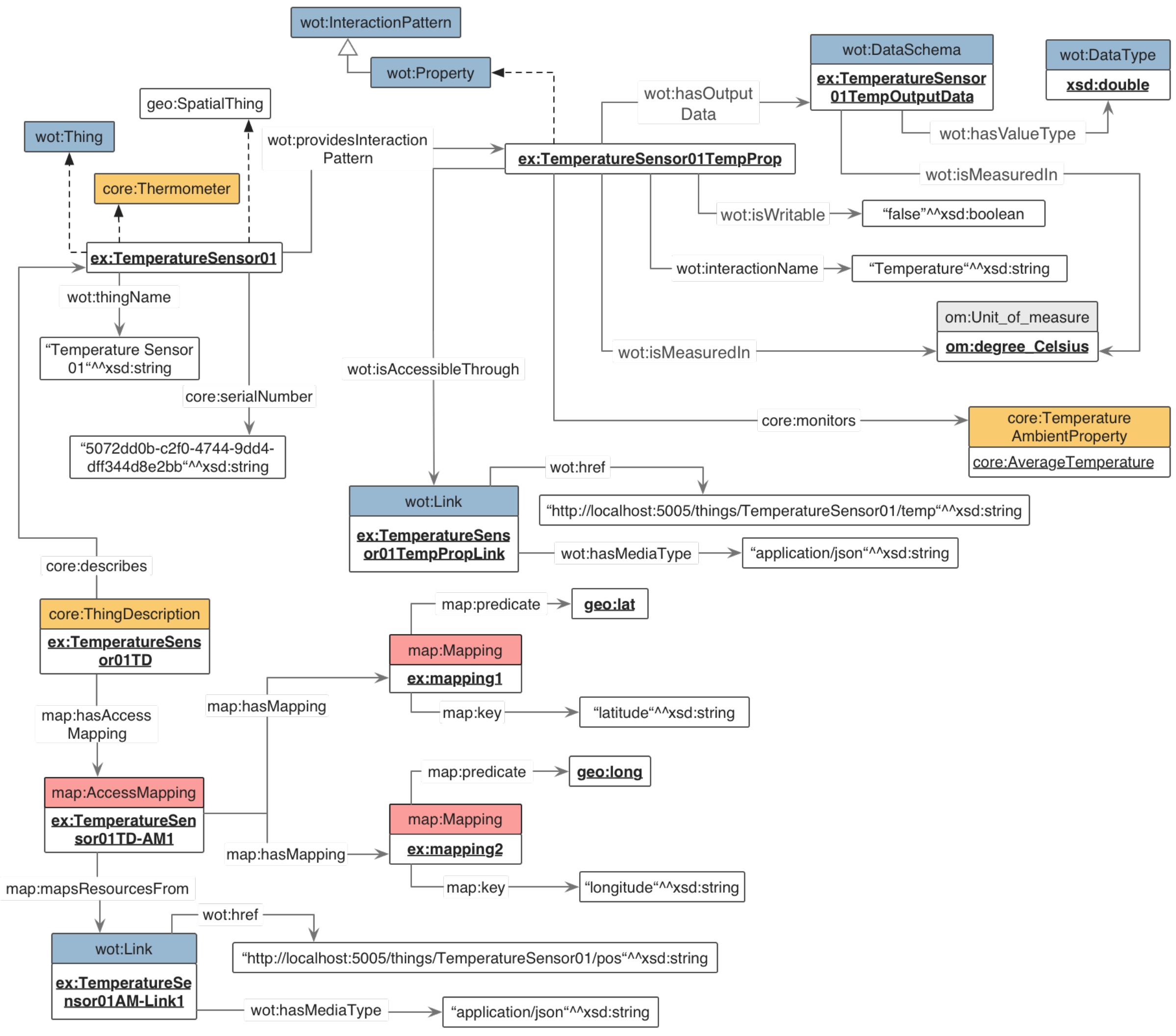


Figure 36. Mappings for geo coordinates example.

An example of interpretation of WoT mappings by the Gateway API is shown in Figure 37. Such figure includes the definition of the mappings already shown in Figure 36, an example of JSON file containing some geo coordinates, and the resulting RDF triples produced by the Gateway API after applying the mappings defined to the JSON document. Such figure also indicates the steps carried out in order to generate the RDF triples for the geo coordinates. These steps are:

1. The Gateway API should be aware of the semantic description of the device, at least the part of the mappings, highlighted in the red box.
2. The Gateway API accesses the endpoint or link in which the data is provided, getting, for example the data in the JSON shown in Figure 37.
3. The Gateway API retrieves the JSON document.
4. By means of applying the mappings to that data, the Gateway API is able to generate the corresponding triples in RDF shown at the bottom-right part of Figure 37.

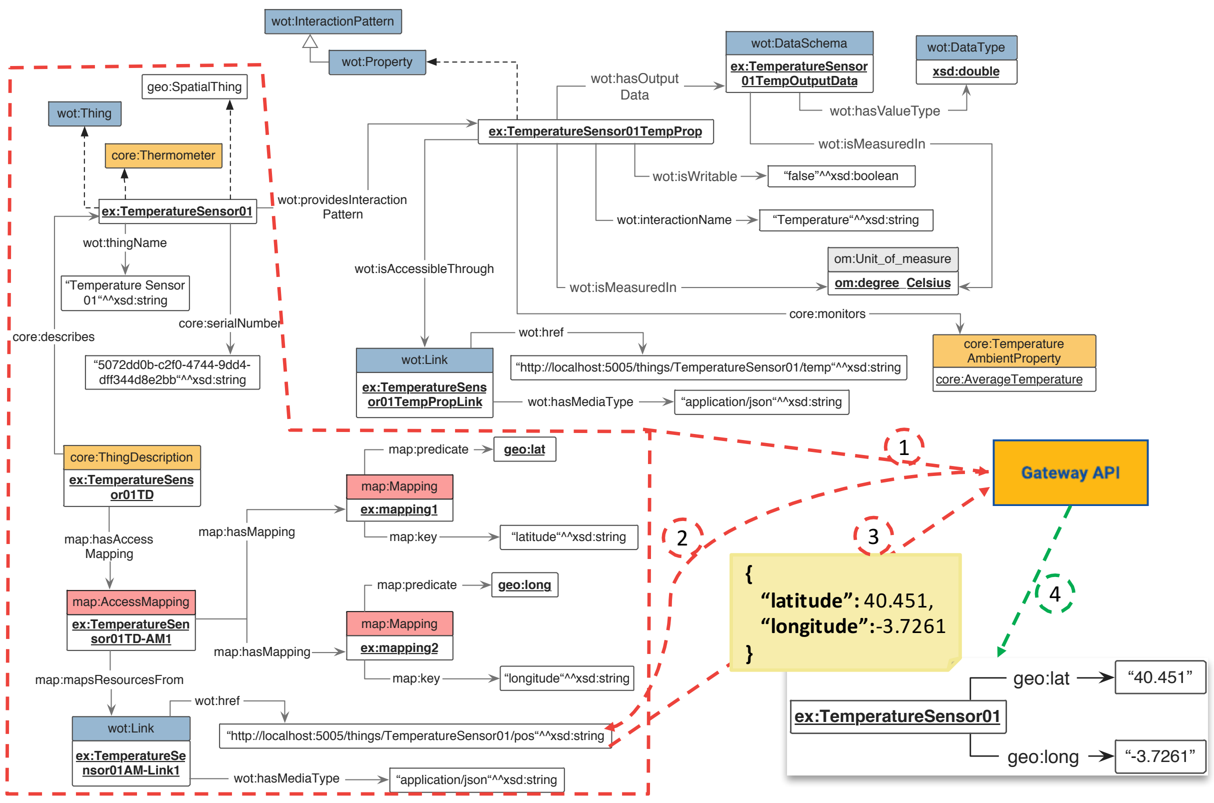


Figure 37. Translation of geo coordinates by using WoT mappings ontology example

7.7. Example 7: Instantiation of IoT objects to enable the semantic discovery

According to "D1.5. VICINITY technical requirements specification", the core component responsible for semantic discovery of IoT objects is VICINITY Semantic discovery and Dynamic Agent Configuration Platform. This component is implemented as the set of services above a semantic triplestore. Such semantic triplestore contains VICINITY ontology and all instances of available IoT objects acquired from adopted infrastructures. Instances are mapped into VICINITY ontology and thus are available for semantic search – the semantic discovery.

The instantiation of IoT objects is part of IoT object registration process. In VICINITY, for each infrastructure to be adopted, there exist standalone client node. The registration process is by default triggered when the new client node starts. It is necessary to inform the VICINITY platform about all IoT objects available in underlying infrastructure. Each client node contains the Adapter component, which serves as the proxy/translator between specific infrastructure and VICINITY. This component is responsible for providing the list of IoT objects available to VICINITY in Common Thing Description Format. Extracted IoT objects are registered into VICINITY network and Neighbourhood Manager, but they are also passed in to Semantic discovery and Dynamic Agent Configuration Platform to be instantiated and thus available for semantic discovery. The instantiation process is composed of two steps:

1. Semantic lifting of IoT object descriptions. Each IoT object description is extended with semantic annotations to VICINITY ontology.
2. Translation into semantic formalism and ontology population. The semantically annotated IoT object descriptions are transformed into Notation3[32] language, which already represents the ontology instance in proper semantic formalism. This instance is stored into ontology and available for semantic search.

The particular steps of instantiation process will be illustrated for one simple example of LightBulb device providing one property for reading. The provided examples describe the current prototype implementation of ontology instantiation process done as part of work in T3.2 VICINITY Semantic Discovery and Dynamic Configuration Services.

The first step is the **acquisition of IoT object descriptions from the infrastructure**. In this case, the adapter component provides the list of all IoT objects available to VICINITY in Common Thing Description Format. The Thing Description format provided in WoT working group current practices document [19] was used as the core of this description language and was slightly adjusted to fulfill the VICINITY requirements. The example of LightBulb device in Common Thing Description Format extracted from adapter is in Listing 1.

In this example, there is presented the example adjustment of core Thing Description form, where general `properties/links` objects were updated to `properties/read_links` and `properties/write_links`, as it was necessary to distinguish which properties are also available only for reading or writing.

The extracted list of IoT object is passed into the Semantic discovery and Dynamic Agent Configuration Platform for ontology instantiation.

[32] https://www.w3.org/TeamSubmission/n3/

```
[{
        "oid": "bulb1",
        "type": "LightBulb",
        "properties": [{
                "pid": "consumption",
                "monitors": "MeanPowerConsumption",
                "output": {
                        "units": "watt",
                        "datatype": "double"
                },
                "read_links": [{
                        "href": "/objects/{oid}/properties/{pid}",
                        "mediatype": "application/json"
                }]
        }],
        "actions": []
}]
```

Listing 1. LightBulb device in Common Thing Description Format.

The **semantic lifting process** takes into account the preconfigured list of keys in JSON schema. The current implementation of semantic lifting constructs the JSON-LD format by adding semantic annotation to Common Thing Description Format provided by adapter by:

- adding annotation for newly generated unique instance identifier
- adding annotation to identified ontology class of IoT object
- adding annotation to identified ontology classes for interaction patterns to semantically distinguish properties, actions and events in IoT object description
- adjusting references to identified ontology instances which represent, which phenomenon's are monitored of affected by properties, actions and events in IoT object description
- adjusting references to identified units used by interaction patterns

The example of semantic lifting for LightBulb example is illustrated in Listing 2.

For each IoT object there is generated unique identifier of new ontology instance. In actual prototype implementation, it is represented by `data:UUID`. The data prefix represents the ontology namespace for all instances available in semantic model, mapped to *http://vicinity.eu/data#* URI.

For each IoT object lifting, there is added the `@context` annotation referring to the JSON-LD context to be used for translation of lifted object into Notation3 language.

The semantically lifted IoT object description is translated into Notation3 formalism using JSON-LD context provided in @context annotation. It provides the set of mapping rules between JSON-LD and Notation3 using VICINITY ontology concepts and relations. The example containing just small part of JSON-LD context transformation rules is illustrated in Listing 3.

```
{
       "oid": "bulb1",
       "@id": "data:d9406af9-0afe-45cd-b717-41f0727bb669",
       "@type": "core:Lightbulb",
       "@context": "thing.jsonld",
       "properties": [{
              "pid": "consumption",
              "@type": "wot:Property",
              "monitors": "core:MeanPowerConsumption",
              "output": {
                     "datatype": "xsd:double",
                     "units": "core:watt"
              },
              "read_links": [{
                     "href": "/objects/{oid}/properties/{pid}",
                     "mediatype": "application/json"
              }]
       }],
       "actions": []
}
```

Listing 2. Semantically lifted LightBulb device.

```
{
        "@context": {
                "@vocab": "http://iot.linkeddata.es/def/wot#",
                "core": "http://iot.linkeddata.es/def/core#",
                "wot": "http://iot.linkeddata.es/def/wot#",
                "xsd": "http://www.w3.org/2001/XMLSchema#",
                "data": "http://vicinity.eu/data#",
                "properties": {
                        "@id": "wot:providesInteractionPattern",
                        "@type": "@id"
                },
                "actions": {
                        "@id": "wot:providesInteractionPattern",
                        "@type": "@id"
                },
                "read_links": {
                        "@id": "wot:isReadableThrough",
                        "@type": "@id"
                },
                "write_links": {
                        "@id": "wot:isWritableThrough",
                        "@type": "@id"
                },
}
```

Listing 3. Illustration of the part of JSON-LD context.

The translation result into Notation3 of the IoT object instance is illustrated in Listing 4.

Once the Notation3 is generated, it can be used for direct population of ontology with new IoT object instance and it is available for semantic discovery.

```
# Prefix definition
@prefix core:    <http://iot.linkeddata.es/def/core#> .
@prefix wot:    <http://iot.linkeddata.es/def/wot#> .
@prefix data:    <http://vicinity.eu/data#> .

@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xsd:     <http://www.w3.org/2001/XMLSchema#> .

# LightBulb instance
data:d9406af9-0afe-45cd-b717-41f0727bb669
      rdf:type core:Lightbulb ;
      wot:thingName "bulb1"^^xsd:string ;
      wot:providesInteractionPattern _:b0 .

# interaction pattern enabling both reading and writing capability
_:b0
      rdf:type wot:Property ;
      core:monitors core:MeanPowerConsumption ;
      wot:interactionName "consumption"^^xsd:string ;
      wot:hasOutputData _:b1 ;
      wot:isReadableThrough _:b2 ;

# instantiation of data model
_:b1
      wot:hasValueType "double"^^xsd:string ;
      wot:isMeasuredIn core:watt .

# instantiation of endpoint reference for reading
_:b2
      wot:href "/objects/{oid}/properties/{pid}"^^xsd:string .
```

Listing 4. Illustration of the IoT object description in Notation3.

Finally, IoT object instances contain semantic references to VICINITY ontology classes and instances. Using this references, VICINITY ontology can be fully used for **semantic discovery** performed as semantic search.

8. Conclusions

Along this document the methodology, infrastructure and current version of the VICINITY ontology network have been detailed. A review of existing standards and some examples about how to use the VICINITY ontologies have also been provided. The current ontology network is composed by three modules, namely the VICINITY Core ontology, the WoT ontology and the WoT Mappings ontology.

The VICINITY ontology network in its version 0.1 not only covers all the requirements defined for the release 0.1 of the VICINITY platform but also anticipates the modelling of some concepts that might be used later in the project, as for example the neighbourhood concepts or the mappings. However, there are remaining requirements to be addressed, moreover for the Core ontology. In this case, future steps will be deciding about which of those requirements are needed to support the VICINITY platforms and which ones should be discarded. Once this decision is made, the ontology will be updated accordingly, including the new concepts, properties or even modules needed. The WoT Mappings will be updated according to VICINITY platforms new needs, if any.

Apart from covering the project requirements, the WoT ontology developed within VICINITY has been taken as base model for the W3C Web of Things Interest Group model. In this sense, The WoT ontology will be updated as long as the thing description specification from the Interest Group.

In terms of modelling and evolving the current network, another future line of work is to further describe types of services, in particular added value services. It might be also needed to extract modules for vertical domains if needed in the project pilots or in the open call projects when they are launched. The ontology will be then validated within real scenarios as well as through the VICINITY platform.

Other next steps include the analysis of the interoperability or conformance level between the VICINITY ontologies and existing ontologies such as SOSA/SSN, SAREF or oneM2M.

In summary, this deliverable presents the first release of the VICINITY ontology, which is mainly focused on satisfying the requirements of the first release of the VICINITY platform. During the project lifetime, new requirements will appear and the VICINITY ontology will be further developed continuing with the strategy of feeding from standards and contributing to them. All the artifacts produced during the ontology development are available online, and will be updated over time, so they represent the most current view of the ontology to anyone interested.

References

[1] Bonino, D. Corno, F. DogOnt – Ontology Modelling for Intelligent Domotic Environments. ISWC 2018. LNCS, vol. 5318, pp. 790-803. Springer (2008)

[2] Charpenay, V., Käbisch, S., & Kosch, H. (2016). Introducing Thing Descriptions and Interactions: An Ontology for the Web of Things. In SR+ SWIT@ ISWC (pp. 55-66).

[3] Compton, M., Barnaghi, P., Bermudez, L., García-Castro, R., Corcho, O., Cox, S., Graybeal, J., Hauswirth, M., Henson, C., Herzog, A., Huang, V., Janowicz, J., Kelsey, W.D., Le Phouc, D., Lefort, L., Leggieri, M., Neuhaus, H., Nikolov, A., Page, K., Passant, A., Sheth, A., Taylor, K. (2012). The SSN ontology of the W3C semantic sensor network incubator group. Web semantics: science, services and agents on the World Wide Web, 17, 25-32.

[4] ETSI TR 103 411- v1.1.1 - "SmartM2M; Smart Appliances; SAREF extension investigation". (February, 2017)

[5] ETSI TS 103 264 - v1.1.1 - "SmartM2M; Smart Appliances; Reference Ontology and oneM2M mapping". (November, 2015)

[6] ETSI TS 103 264 - v2.1.1 - "SmartM2M; Smart Appliances; Reference Ontology and oneM2M mapping". (March, 2017)

[7] ETSI TS 103 410-1 – v1.1.1 - "SmartM2M; Smart Appliances Extension to SAREF; Part1: Energy Domain". (January, 2017)

[8] ETSI TS 103 410-2 – v1.1.1 - "SmartM2M; Smart Appliances Extension to SAREF; Part1: Environment Domain". (January, 2017)

[9] ETSI TS 103 410-3 – v1.1.1 - "SmartM2M; Smart Appliances Extension to SAREF; Part1: Building Domain". (January, 2017)

[10] ETSI TS 118 112 – v2.0.0 – "oneM2M; Base Ontology (oneM2M TS-0012 version 2.0.0 Release 2)". (September, 2016)

[11]Grimm, C., & Bonino, D. (2014, January). Towards standardization of M2M communication in Smart Appliances. IN EEBuilgind Data Models. Energy Efficiency Vocabularies & Ontologies. ICT for Sustainable Places.

[12]Grüniger, M. and Fox, M. (1995) Methodology for the design and evaluation of ontologies. In Skuce, D. (ed.) IJCAI95 Workshop on Basic Ontological Issues in Knowledge Sharing, pp 6.1-6.10

[13] Haller, A., Janowicz, K., Cox, S., Le Phouc, D., Taylor, K., Lefrançois, M., Atkinson, r., García-Castro, R., Lieberman, J., Stadler, C. (July, 2017). Semantic Sensor Network Ontology. W3C Candidate Recommendation. 11 July 2017. https://www.w3.org/TR/2017/CR-vocab-ssn-20170711/

[14]Mathew, S. S., Atif, Y., Sheng, Q. Z., & Maamar, Z. (2011, October). Web of things: Description, discovery and integration. In Internet of Things (iThings/CPSCom), 2011 International Conference on and 4th International Conference on Cyber, Physical and Social Computing (pp. 9-15). IEEE.

[15]Poveda-Villalón, M., Gómez-Pérez, A., & Suárez-Figueroa, M. C. (2014). OOPS!(ontology pitfall scanner!): An on-line tool for ontology evaluation. International Journal on Semantic Web and Information Systems (IJSWIS), 10(2), 7-34.

[16]Studer, R., Benjamins, V.R., Fensel, D.: Knowledge engineering: Principles and methods. Data & Knowledge Engineering 25(1-2) (1998) 161-197

[17]Suárez-Figueroa, M. C., Gómez-Pérez, A., & Fernández-López, M. (2012). The NeOn methodology for ontology engineering. In Ontology engineering in a networked world (pp. 9-34). Springer Berlin Heidelberg.

[18]VICINITY project website http://www.vicinity-h2020.eu

[19]WoT Current Practices (Unofficial Draft). http://w3c.github.io/wot/current-practices/wot-practices.html#quick-start-td-samples

Keywords