H2020 – LC – SC3 – EE – 2019 – GA 894240

Operating System for Smart Services in Buildings

D3.2 domOS Common Ontology, Initial Version

WP3 Common Ontology and Semantics

	Name	Date
Prepared by	Amir Laadhar (AAU), Christian Thomsen (AAU)	02.08.2021
Peer reviewed by	Martin Vodnik (INEA)	05.08.2021
Reviewed and approved by	Dominique Gabioud (HES-SO)	25.08.2021

This project has received funding from the European Union’s Horizon 2020 research and innovation program under grant agreement No 894240.

Distribution list

External		Internal	
European Commission	x	Consortium partners	x

Change log

Version	Date	Remark / changes
1.1	02.08.2021	First version by AAU
2.1	16.08.2021	Second version by AAU
3.1	27.08.2021	Final version by HES-SO

To be cited as

"D3.2 domOS Common Ontology, Initial Version" of the HORIZON 2020 project domOS, EC Grant Agreement No 894240.

Disclaimer

Table of contents

List of figures

List of tables

Terms, definitions, and abbreviated terms

JSON	JavaScript Object Notation
JSON-LD	JavaScript Object Notation for Linked Data
RDF	Resource Description Framework
IoT	Internet of things
dCO	domOS Common Ontology

Executive Summary

This document is a deliverable of the domOS project, funded by the European Commission under its Horizon 2020 Research and innovation program (Grant Agreement No. 894240), reporting the results of the activities carried out during task T3.2, namely the "Development of domOS Common Ontology". This deliverable focuses on the initial version of the domOS Common Ontology (dCO). The main objective of WP3 is to provide a common information model, i.e., the dCO, to share a unified understanding of the IoT ecosystem in domOS and ensure semantic interoperability between machines. This will decouple the domOS infrastructure from the software services (e.g., flexibility, energy efficiency, district heating).

1. Introduction

According to the energy efficiency report of the European Union (EU), buildings are responsible for approximately 40% of the energy consumption and 36% of the CO_2 emissions of the EU (European Commission, 2021). 75% of the existing building stock in the EU has been assessed as energy inefficient. Energy efficiency systems have wide benefit from IoT technology, which aims to collect and connect all data from different items such as sensors and in-building appliances. By collecting real time data, the energy efficiency can be increased through the use of smart applications. Five of the key architectural requirements of smart buildings automation systems reported by the United States Department of Energy are interoperability, scalability, deployment, open and plug-and-play (Katipamula, et al., 2012). These components are closely related to semantic interoperability in smart buildings addressed by this deliverable. In this deliverable, we focus on ensuring semantic interoperability to ensure energy efficient systems for smart buildings. Semantic interoperability denotes the ability of different machines to have the same understanding of the meaning of the shared data through the use of a common vocabulary. In Semantic Web, ontologies define formal naming of entities and their properties. An ontology is formally defined as an explicit specification of a conceptualization. Ontologies are often represented as a directed, labelled graph. Different machines can refer to an ontology for the set of definitions and relationships in a target domain, which helps to reduce ambiguity and ensure the semantic interoperability.

A number of ontologies have been proposed for smart buildings. Most of these ontologies are domain ontologies, which focuses on a specific domain (e.g., building domain, energy domain, IoT devices). Some domain ontologies are standardized (e.g., (Daniele, et al., 2015), (Haller, et al., 2018)). However, standardized domain ontologies usually include general terms that are common across a single domain. Therefore, domain ontologies are reused to build application ontologies. Application ontologies are engineered for a specific use or application focus. The dCO is an application ontology that can be used in the scope of domOS project. We reviewed in Deliverable 3.1 the state-of-the-art standardized domain ontologies for IoT and smart buildings.

1.1. Purpose of the Deliverable

An objective of the domOS project is to enable the interoperability of data and services for smart buildings. Interoperability can be enabled by using a common information model, i.e., an ontology, to share a unified understanding of the information structure among people and software agents. Consequently, the main purpose of this deliverable is to develop the first version of the dCO to decouple building infrastructure from the actual services.

1.2. Structure of the Document

The rest of the deliverable is structured as follows:

- **Section 2** presents the used methodology for the development of the dCO, initial version.
- **Section 3** provides a description of the design of the dCO, initial version.
- **Section 4** includes a description of the implementation of the initial version of the dCO to deliver different capabilities such as semantic discovery, semantic annotation, and compliancy checking.
- **Section 5** presents our conclusion and future line of work.

2. dCO Development Methodology

In this section, we define the dCO development methodology. The novelty of this methodology that it is based on an agile process for the construction of the ontology. This methodology is organized in three phases with different expectations from each actor throughout the process: Phase 1: Ontology requirements specification document, Phase 2: Knowledge engineering, Phase 3: Post development.

2.1. Phase 1: Ontology Requirements Specification Document

The objective of requirements specification is to create a document that covers the scope, use cases and requirements of the ontology. The document is called the Ontology Requirement Specification Document (ORSD). In the following, we define the steps of phase 1.

Step 1: Ontology Scope

The goal is to define the exact scope of the ontology. This will help to define the exact boundaries of the ontology and its level of granularity in terms of the knowledge that it should represent. This step should be done in collaboration between the knowledge scientist and the stakeholders.

Step 2: Use Cases Identification

In this step, the knowledge scientist collaborates with the stakeholders to define a list of use cases that the ontology should fulfil.

Step 3: Identification of Requirements

During this step, the knowledge scientist should define the functional and non-functional requirements. Functional requirements refer to the knowledge to be represented by the ontology, and can be stated as competency questions i.e., the questions that the ontology should be able to answer. An example of competency questions for the NEOGRID demonstration site can be found in the Microsoft Teams directory[1]. Non-functional requirements are the general requirements or aspects that the ontology should fulfil, optionally including priorities for each requirement (Suárez-Figueroa, et al., 2015).

[1] https://bit.ly/3m90eXb

2.2. Phase 2: Knowledge Engineering

Phase 2, knowledge engineering, includes three steps. Each step is composed of several tasks. We summarize in Table 1, the input, output, and actors of every task.

TABLE 1: KNOWLEDGE ENGINEERING PROCESS

Steps and Tasks	Input	Output	Actor
Step1: Knowledge Capture			
Task 1: Terminology	Prioritized list of requirements and use cases	Extraction of the key terms from the ORSD.	Knowledge scientist Domain expert
Task 2: Glossary	Key concepts identified in task 1	Definition of list terms associated textual description, synonyms, and the possible reused ontologies. The glossary terms can be categorized as classes, properties, and instances.	Knowledge scientist Domain expert
Step 2: Knowledge Implementation			
Task 3: Ontology Formalization	Result of tasks 1 - 2	Formalization of the ontology in the Web Ontology Language (OWL) by the creating or reusing existing the knowledge.	Knowledge scientist
Task 4: Mappings to Existing Resources	Formalized OWL ontology	Formalization of the mappings using mapping properties from formal languages like OWL and SKOS (Simple Knowledge Organization System)	Knowledge scientist
Step 3: Knowledge Validation			
Task 5: Pragmatic Validation	Results of tasks 3 and 4	Verification of the fidelity and relevance, irrespectively of its syntax and semantics. Fidelity can be measured by checking if the claims an ontology makes are true in the target domain. Relevance is checked in conjunction with completeness, verifying the correct implementation of the ontology's requirements.	Knowledge scientist Domain expert
Task 6: Syntactic Validation	Result of Task 5	Verify the syntax of the formalized ontology.	Knowledge scientist
Task 7: Semantic Validation	Formalized OWL ontology	Checking the consistency of the ontology using a reasoner.	Knowledge scientist

In the following, we detail each step of the dCO knowledge engineering.

Step 1: Knowledge Capture

Task 1: Terminology Definition: The output of the knowledge capture is a knowledge document. The knowledge document contains the terminology and glossary of the ontology. The knowledge document should be elaborated in collaboration with the stakeholders to define the terminology and glossary of the ontology and any other useful documentation or information that can help in the construction of the ontology. The terminology is a finite list of terms (e.g., device, sensor, appliance). For instance, the ontology glossary contains the term device. In the following is the definition of the term device as defined in the glossary: device (term name), a device is a physical piece of technology (term definition), and SSN (reused ontology for the term device) (Haller, et al., 2018). The terminology and the glossary are stored in a knowledge document, which is a collaborative spreadsheet. The knowledge scientist generates the terminology from the ORSD, and any other provided documentation given by the collaborators. At this level the terminology is a simple, flat, undifferentiated (with respect to, say, nouns and verbs) list of terms. The main challenge is to identify the boundary of a domain to include a term or not in the terminology. This can be decided by assessing the relevance of a term by the knowledge scientist and the domain expert.

Task 2: Glossary Definition: The glossary is a finite set of terms associated with textual definitions, possible synonyms, the possibly reused ontology for each term (e.g., the already existing SSN ontology is reused for the term device), and the semantic type of each term (class, property, and instance). The defined glossary for the dCO can be found in the Microsoft Teams directory[2] of the domOS project. For each term, the knowledge scientist should define a preferred label, definition, and synonyms as alternative labels. The glossary hides rich semantics. Therefore, the knowledge scientist and domain expert should add to the glossary the set of relationships between the different terms. The knowledge scientist should represent the hidden semantics by defining semantic relationships between terms. A property relationship can be subsumption, parthood, or any other domain specific property. Domain specific properties are properties made specifically for a specific domain when they do not exist in the state-of-the-art ontologies. A subsumption property is used to create a hierarchy of classes, typically with a maximally general class at the top, and very specific classes at the bottom. The hierarchy of the classes can be created using the rdfs:subClassOf property. Parthood relationship concerns architectural structure of composite entities, by eliciting their decomposition hierarchy (or part-whole hierarchy based on the owl:partOf relationship (and its inverse owl:hasPart). The goal of defining a glossary is to agree and define clear boundaries of the domain. The main challenge in this task is to agree on common definition and synonyms of a term. This can be challenging since different domain experts can have different point of views. Moreover, the same term can have different contradictory definitions in different resources. Therefore, converging to a unified glossary is essential.

Step 2: Knowledge Implementation

Task 3: Ontology Formalisation. Using the outcome of the previous tasks, the knowledge scientist can create the ontology or reuse and extend the existing ones using the Resource Description Framework (RDF) and Ontology Web Language (OWL). Knowledge engineers encode every term from the glossary as a class, an instance, an object property, or a data property. If the term is encoded as a class, it can use the properties rdfs:label or skos:prefLabel to define the labels of each class. Synonyms can be encoded as alternative labels using the property skos:altLabel. Definitions can be encoded using the property skos:definition. Relationships can be encoded as object properties or data properties. Moreover, the

[2] https://bit.ly/3AFcUcy

knowledge engineer can formalise constraints using OWL. We encourage the use of terms and object properties from the existing ontologies to improve interoperability.

Task 4: Mapping to Existing Resources. FAIRsFAIR H2020 EU project highlighted in its semantic FAIR recommendations that ontology mapping is an important aspect to consider in order to ensure interoperability between ontologies (Hugo, et al., 2020). The knowledge engineer should use mapping properties to define the semantic of a mapping. The use of ontology alignment systems is useful to generate mappings in less time. However, these mappings should be manually verified to check their validity.

Step 3: Knowledge Validation

Task 5: Pragmatic Validation. Pragmatic validation refers to assessing the ontology usefulness for its users regardless of its syntax and semantics. Pragmatic validation is assessed using three criteria: accuracy, comprehensiveness, and relevance. Accuracy is whether the claim an ontology makes are true. Domain experts can evaluate the accuracy of the claims of an ontology. Comprehensiveness refers to including a complete representation of their domain knowledge. Domain experts can manually evaluate the comprehensiveness of an ontology. Relevance evaluates whether an ontology satisfies ontological requirements. The knowledge scientist can assess the relevance of an ontology by developing SPARQL queries that should return the expected knowledge.

Task 6: Syntactic Validation. Syntactic validation refers to evaluating the accuracy of the ontology based on its syntax. A value is syntactically accurate, when it is part of a legal value set for the represented domain, or it does not violate syntactical rules defined for the domain. The knowledge engineer can evaluate the syntactic quality of an ontology by using automated tools such as http://visualdataweb.de/validator/.

Task 7: Semantic Validation The knowledge scientist and the domain expert evaluate the semantic quality of an ontology based on its consistency, and clarity. An ontology is consistent when it is free of logical and formal contradictions with respect to particular knowledge representation and inference mechanisms. A straightforward way to check for consistency is to load the knowledge base into a reasoner and check whether it is consistent. Moreover, the knowledge engineer can also use automated tools to check for inconsistency in an ontology. Clarity is whether the context of a concept is clear. For example, if an ontology claims that class “Chair” has the property “Salary”, an agent must know that this describes academics, not furniture. Domain experts should verify the clarity of a given ontology.

2.3. Phase 3: Post Development

Step 1: Ontology Publication

Perfectly good ontologies can be unused simply because no one knows they exist. Consequently, knowledge scientists should register their ontologies and make them available in the existing ontology repositories. We recommend knowledge scientists to first register their ontologies in domain specific repositories, e.g., LOV4IOT for IOT ontologies, etc. When domain-specific repositories do not exist, knowledge engineers can register their ontologies in generic repositories, such as LOV, FAIRsharing.

Step 2: Ontology Documentation

Once an ontology is published on the web, the next step is to make it findable by its potential users. To help search engine crawlers to understand ontology metadata, an HTML page of an ontology should include in-document annotations through JSON-LD snippets. For instance, a JSON-LD snippet can be included in the HTML page of a documentation to mention the latest version of the ontology and its editor. Some ontology repositories include in-document annotations to be indexed by search engines as it is currently done by AgroPortal. WIDOCO tool (ref) automatically generates rich HTML documentation from ontology metadata. WIDOCO includes JSON-LD snippets with a description of the ontology metadata. These snippets are useful for search engines to find and explore the metadata of the ontology documentation automatically.

Step 3: Ontology Maintenance

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

3. dCO Design

3.1. Ontology Requirements Specification Document

The ORSD was made in collaboration with the partners from the five demo sites (Sion, Paris, Aalborg, Neuchatel, and Skive) using a collaborative spreadsheet. In the ORSD, we state the purpose of the ontology, scope, indented users, use cases, functional requirements, and non-functional requirements.

Purpose: The main purpose of the dCO is to be used for semantic interoperability to decouple the hardware infrastructure form the applications and services.

Scope: The dCO must focus on the IoT domain, more precisely on devices with their Things Descriptions, units of measurements, services, building metadata, and building topology, etc. The level of granularity is directly related to the identified terms from the functional requirements.

Intended End-Users: The main users of the dCO are the IoT developers, IoT architects. IoT developers who needs to annotate and semantically discover IoT devices. IoT architects who needs to design applications and services.

Use Cases: After interviewing partners from the five demonstration sites, we have identified the following use cases of the dCO.

(i) **Semantic Annotation** of Thing Descriptions (TDs) (W3C, 2020) and Building Descriptions (BDs). TDs and BDs are JSON-LD files that describe the metadata of a Web of Thing (WoT) Thing and a building, respectively. These description files should be annotated using the entities of the dCO. The semantic annotation allows humans and machines to correctly understand the meaning of the metadata. This ensures semantic interoperability. TDs contain semantic annotations describing the type of a device, the set of its properties, actions, and events. BDs can contain semantic annotations related the metadata of a building itself, building topology, and existing appliances.

(ii) **Semantic Validation** of the annotated Thing Descriptions and Building Descriptions. Once TDs and BDs are semantically annotated, they should be semantically verified in order to make sure that there are no inconsistencies. For instance, a temperature sensor in a TD should not contain semantic annotations of properties of a power meter. Based on the semantic annotations, TDs and BDs can be serialised into RDF to be stored in a triplestore. A triplestore is a scalable database specially designed for the storage and retrieval of triples through semantic queries. Therefore, the TDs and BDs should be validated in order to generate a valid RDF serialisation.

(iii) **Semantic Discovery** of the TDs. Once TDs and BDs are instantiated into a triplestore, users can semantically search for thing descriptions.

(iv) **Compliancy Checking** of new IoT application for the existing buildings. IoT applications can check their compliancy for a building.

Non-Functional Requirements: we defined the three following non-functional requirements.

(i) The dCO should be compliant with FAIR principles, which refers to the level of Findability, Accessibility, Interoperability, and Reuse of the developed ontology (Wilkinson, et al., 20116).

(ii) The dCO should reuse when possible and extend the state-of-the-art ontologies to address the functional requirements.

(iii) The dCO should be designed in a modular way.

3.2. dCO Knowledge Engineering

3.2.1. Reused Ontologies for dCO.

Based on the input given from the 5 demonstration sites, we built a glossary of terms, which reference the possibly reused existing ontology terms for each term of the glossary. Many external ontologies can represent the same term. As a result, we need to define a guideline to choose a single term. We give priority to reuse terms existing respectively in the following types of ontologies: standardized ontologies (SSN, SOSA, SAREF), upper-level ontologies (BFO), domain ontologies (iottaxolite) and application ontologies (vicinity). If we find a set of term existing in the same type of ontology, we choose the term having the best ranking score on the Linked Open Vocabularies (LOV). LOV is an ontology catalogue of reusable vocabularies. This catalogue is a search-engine of ontology terms and display the terms based on its ranking. The search-engine defines the ranking based on the term frequency inverse document frequency (tf-idf) and a popularity metric. The popularity metric provides an indication on how widely a term is already reused in the catalogue. We classified every term of the glossary into one of the following 5 modules: Building Description module, Thing Description module, device module, units of measurement module, and core module. We have opted to design dCO in modules to increase the maintainability, reusability, and scalability of the ontology. In the following, we state the dCO modules:

- Building Description Module: this module contains the representation of building related metadata and building topology.
- Thing Description Module: this module aims to reuse and extend the Thing Description ontology (ref) to model properties, actions, and events. We include the Thing Description module since domOS ecosystem project relies on the WoT technology (W3C, 2020) to ensure interoperability.
- Device Module: this module represents the needed terms to represent devices involved in the 5 demonstration sites.
- Units of Measurements Module: this module aims to represent all the units of measurements involved in the devices.

- Core Module: this module represents the core terms and allows the integration of the other modules.

In the following table we state the reused ontologies for every module. We report in Table 2 the reused ontologies for each module.

TABLE 2: REUSED ONTOLOGIES BY DCO MODULES

Module	Reused Ontologies
Building Description	W3C BOT (Rasmussen, et al., 2020)
Thing Description	W3C TD (Charpenay, et al., 2020)
Units of Measurement	UM (Rijgersberg, et al., 2013), SAREF (Daniele, et al., 2015)
Devices	SAREF (Daniele, et al., 2015), S4BLDG (Poveda-Villalon, et al., 2018), W3C SSN (Haller, et al., 2018)
Core	SAREF (Daniele, et al., 2015), W3C SSN (Haller, et al., 2018), VICINITY (Cimmino Arriaga, et al., 2019)

Figure 1 shows the main classes and properties of the dCO, which is based on a modular design reusing existing ontologies whenever possible. The full documentation of the dCO is available at: https://www.dco.domos-project.eu/.

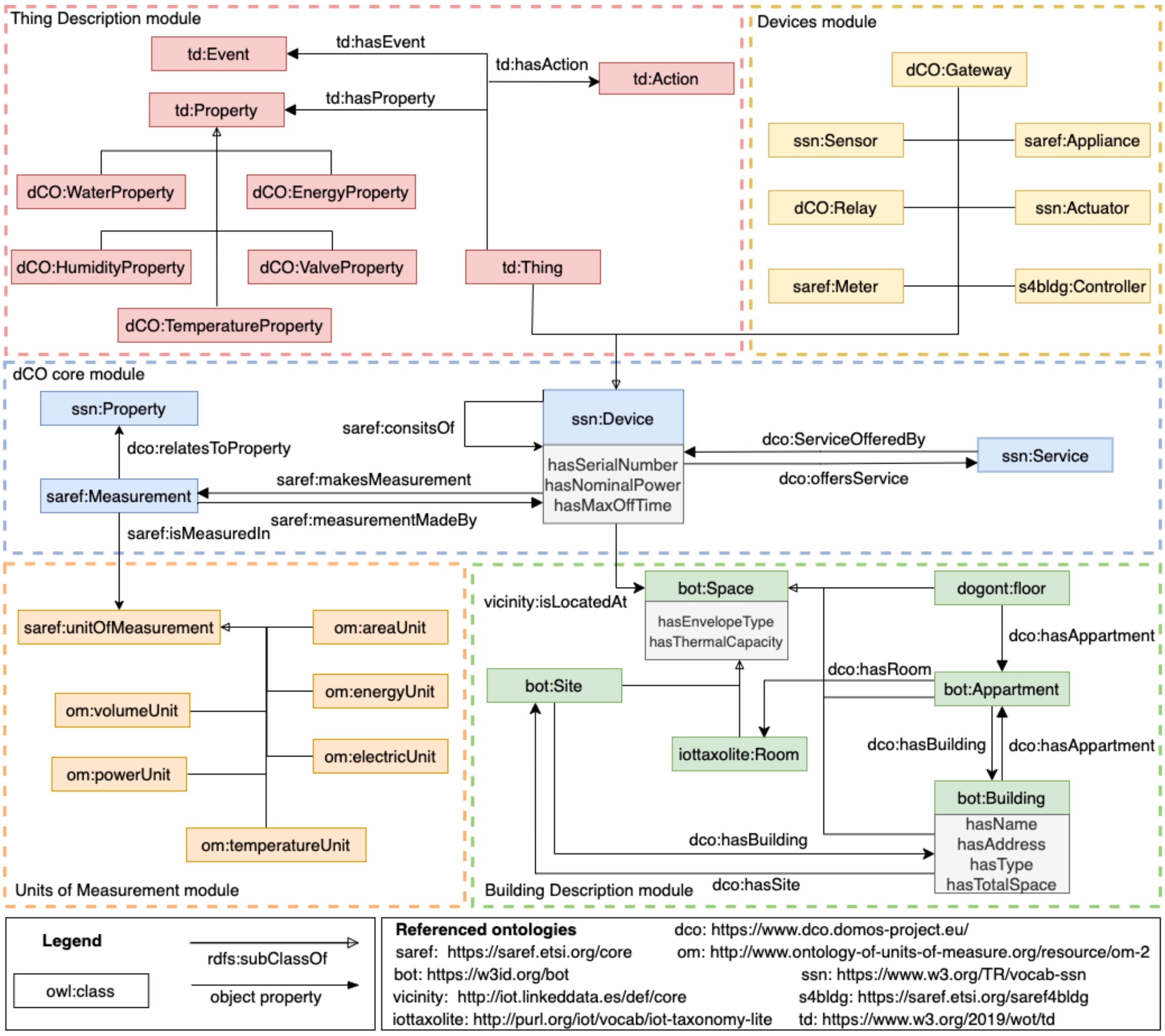


FIGURE 1: DCO DESIGN OVERVIEW

In the following, we detail the design of each module:

<u>Device Module</u>:

Devices are structured in categories (subclasses), that reflect different types of devices (e.g., saref:applicance. ssn:Sensor, saref:Meter, s4bdg:Controller). All subclasses inherits the properties of an ssn:device. Each category can include one or more other devices (sub-classes). For instance, a saref:Meter is categorized in the following meters: energy meter, gas meter, heat meter, power meter, smart meter, and water flow meter. An ssn:device can consist of (srafe:consistsOf) one or more ssn.device. A saref:appliance is an ssn:devices. Therefore, a saref:appliance is a device that can consist of one or more devices. For instance, a heat pump appliance can consist of a heat pump device, a temperature sensor, and a humidity sensor. A saref:appliance can offer one or more services (ssn:service). A ssn:service can be offered by one or more devices.

<u>Core Module</u>:

The core dCO module focuses on the concept snn:Device, which is defined as a tangible object making a measurement expressed using a unit of measurement to observe an snn:Property. An ssn:device can consist of (saref:consistsOf) one or more ssn.device. A saref:Device can offer one or many ssn:Service.

An ssn:Service can be offered by one or more devices. For example, two solar photovoltaic appliances can offer a single solar photovoltaic service. A device can be located at a bot:Space. For instance, a temperature sensor device can be related to both a flexibility service and a heating service and deliver a heating function. This device makes a measurement (saref:measurement) of the temperature property (ssn:property). A device can have some metadata that uniquely characterize it, for instance, its name, manufacturer and serial number (dco:hasName, dco:hasSerialNumber and dco:hasManufacturer, respectively).

Units of Measurement Module:

An ssn:Measurement is a measurement using a saref:unitOfMeasurement, which is structured in categories that reflect the different types of unit of measurements (e.g., om:areaUnit, om:energyUnit, om:temperatureUnit). Each category can include one or more units of measurements. For example, volume units contain cubic meter and cubic meter per hour. We reuse in the units of measurements module the Units of Measurement Ontology (OM).

Building Description Module:

In this module, we mainly reuse the Building Topology Ontology (BOT) of the W3C Linked Building Data Group. We also define in this module a set of metadata related to the characteristics of a bot:building and a bot:space. An ssn:Device can be located at a bot:space, which a limited three-dimensional extent defined physically. We defined 4 types of spaces: bot:site, bot:building, dogont:floor, bot:apartment, and iottaxolite:Room.

A bot:space has several properties that represent energy characteristics related to a space (e.g., building, room, floor). These properties are dco:activePowerExport, dco:hasReactivePowerImport, dco:hasReactivePowerExport, dco:hasConsumedPower, dco:hasEnvelopeType and dco:hasThermalCpacity.

A site can contain one or more buildings. A building can be located at only one site. A building must contain at least one floor, which must contain at least one apartment. An apartment can contain one or more rooms. A room can be categorized into several types, such as living room or a kitchen. A building can have several properties such as name, address, and totalArea.

Thing Description Module:

The purpose of the thing description module is to represent W3C Things Descriptions (TDs). This module allows the semantic annotation of the TDs. Based on these annotations, the TDs can be instantiated in a triplestore to be Exposed through a SPARQL end-point accessible over HTTP. Moreover, the thing description module help in the semantic validation process of the TDs.

A td:thing subsumes a ssn:device. Therefore, the td:thing inherits the object properties and data properties of an ssn:device. Consequently, a td:thing can make a measurement (saref:makesMeasurement), which is measured in (saref:isMeasuredIn) a unit of measurement (saref:unitOFMeasurement). A td:thing can have the hasSerialNumber and hasManufacturer properties, and any other property associated to an ssn:device. According to the subsumption property between a device and its categories, a td:thing can be categorized as one of the devices included in the device module. A td:thing can be related to at least one ssn:service. A td:thing can be associated to one or more td:actions, td:event, and td:property. A td:thing can be located at a bot:space.

4. dCO Implementation: Semantic Interoperability as a Service

4.1. Semantic Discovery Architecture

One of the main challenges of implementing semantic interoperability in an IoT ecosystem is to enable consumers to search for IoT devices without having any prior knowledge about them. To implement such functionality, we make the following assumptions:

- There is a common information model to describe things, namely the dCO. Every Thing Description and Building Description must be instantiated in the triplestore to allow semantic discovery.
- Thing Descriptions and Building descriptions must be used for describing WoT things and building metadata.
- Thing Descriptions and Building descriptions must be semantically annotated using the common information model, i.e., the dCO.

In Figure 2, we depict the steps of the semantic discovery process, which consists of the following sequence:

1. A platform operator sends a set of Thing Descriptions and a Building Description to the API of the Web of Thing Directory. The received input is forwarded to the syntactic and semantic validation process. The platform operator is the person responsible for the operations in an IoT platform.
2. Syntactic and semantic validation ensures that the received JSON-LD files of the Thing Descriptions and Building Descriptions are syntactically and semantically valid. The syntax of the JSON-LD and its included semantic annotations should be correct. Moreover, the received files should not contain any semantic inconsistency. Once the Thing Descriptions and the Building Description are validated, they are forwarded to the next step. Otherwise, they are sent back to the platform operator to ensure the necessary rectification. If the Thing Descriptions and the Building Description are validated, they should be stored in a document database (2b) and a triplestore (2a). The document database ensures the storage and retrieval of the JSON-LD of the Thing Descriptions and the Building Description. The triplestore allows the semantic discovery of the Thing Descriptions and the Building Description. The Thing Descriptions and the Building Description should be serialized into RDF before their storage in the triplestore.
3. In this step, the serialization engine serializes the JSON-LD of the Thing Descriptions and the Building Description into RDF. These files are forwarded to the SPARQL API in order to be instantiated in the triplestore.
4. If Thing Descriptions and the Building Description are stored in the document database as a JSON-LD files and the triplestore as RDF files, then the applications and services can send semantic discovery queries to the SPARQL API.

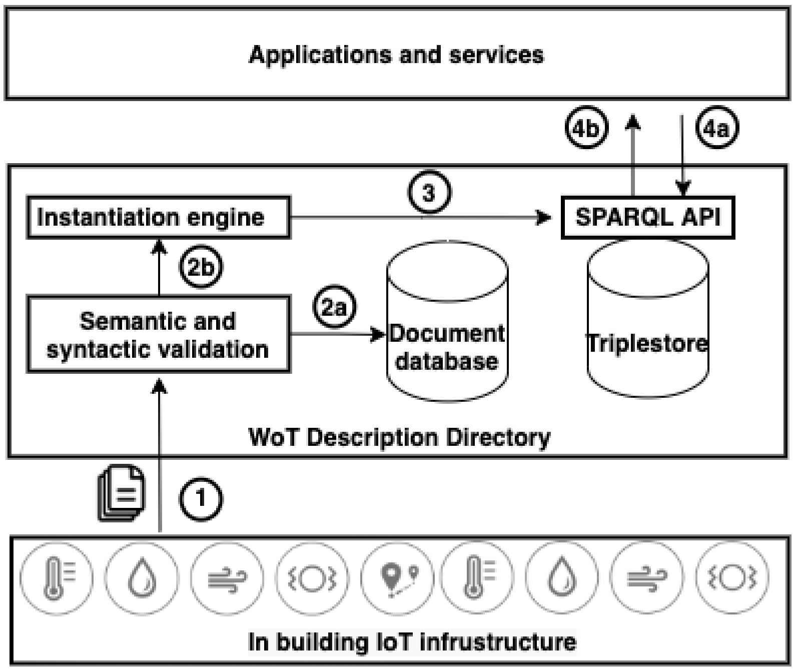

Applications and services
4b
4a
3
Instantiation engine
SPARQL API
2b
2a
Semantic and syntactic validation
Document database
Triplestore
WoT Description Directory
1
In building IoT infrustructure

FIGURE 2: SEMANTIC DISCOVERY ARCHITECTURE

We rely on several existing software components and technologies to ensure this architecture, namely:

- **Knowledge Engineering**: we use WebProtégé, which is an ontology development environment for the Web that makes it easy to create, upload, modify, and share the ontology for collaborative viewing and editing. We also use Protegé Desktop that features a richer ontology editing environment but lacks collaborative capabilities.
- **Triplestore**: we rely on the scalable and high performant triplestore of Apache Jena, which is a free and open-source framework for building Semantic Web and Linked Data applications.
- **Document Database**: we use the document database of MongoDB to store JSON-LD files.
- **SPARQL Engine**: We use Apache Jena Fuseki as a SPARQL server. It exposes the dCO as a SPARQL end-point accessible over HTTP. Fuseki provides REST-style interaction with RDF data.
- **Semantic Validation**: we check the semantic validation of the ontology by running the OOPS! tool (http://oops.linkeddata.es/response.jsp?uri). This tool helps to detect inconsistencies in ontologies and report them.
- **Syntactic Validation**: we rely on the OWL Manchester validator (http://visualdataweb.de/validator/) to validate the syntax of the ontology and instantiated triples.

4.2. IoT Applications Compliancy Checking

One of the main challenges in IoT ecosystems is allowing new external applications to provide services to users based on the heterogeneity existing in building IoT infrastructure (Balaji, et al., 2016). In many cases, adding new applications to an existing IoT ecosystem is a cumbersome task due to the lack of IoT semantic interoperability. Moreover, usually, new applications providers do not understand the existing IoT ecosystems and the variety of its installed devices. As a result, a service provider does not know if its application is compliant or not with a given IoT ecosystem. To solve this issue, we propose an IoT application compliance check process. To implement this process, we make the same assumptions as in the semantic discovery process (c.f. section 5.1).

In Figure 3, we depict the sequence diagram of the application compliance check process.

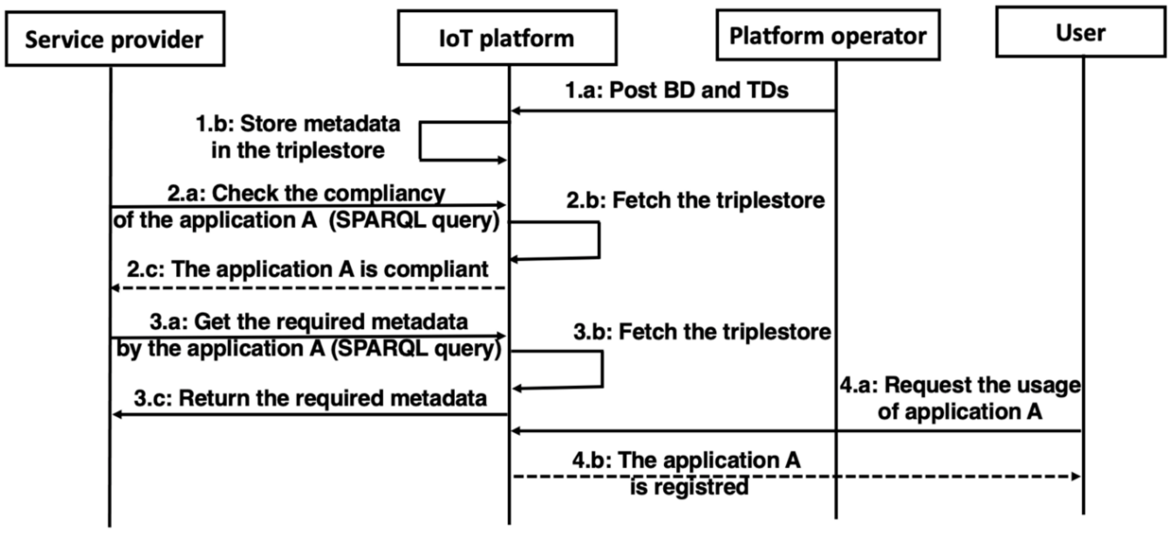

Service provider
IoT platform
Platform operator
User
1.a: Post BD and TDs
1.b: Store metadata in the triplestore
2.a: Check the compliancy of the application A (SPARQL query)
2.b: Fetch the triplestore
2.c: The application A is compliant
3.a: Get the required metadata by the application A (SPARQL query)
3.b: Fetch the triplestore
3.c: Return the required metadata
4.a: Request the usage of application A
4.b: The application A is registred

FIGURE 3: IOT APPLICATIONS COMPLIANCE CHECKING

The application compliance checking process consists of in the following sequence of steps:

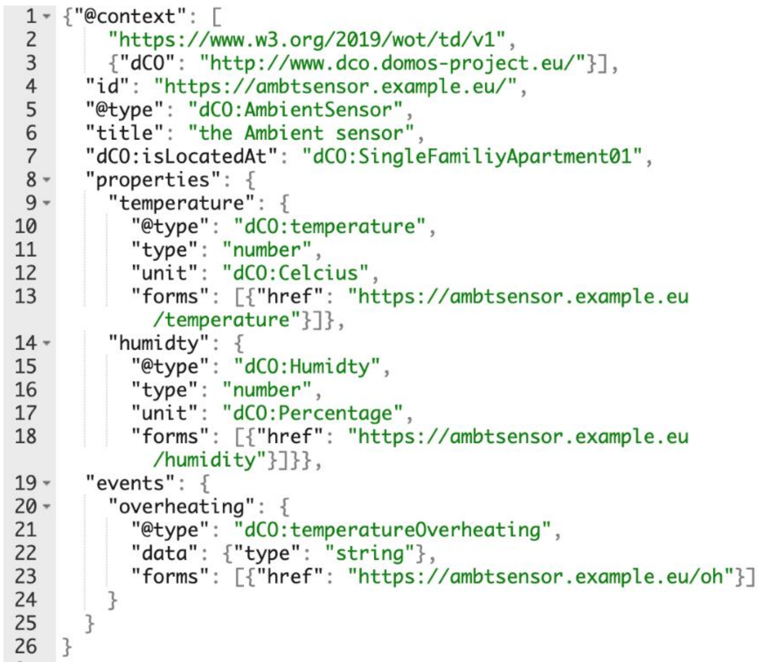
1. The process begins by posting the Thing Descriptions and a Building Description of a building in the WoT Directory. The Thing Descriptions and a Building Description should be syntactically and semantically validated before their storage in the WoT Description Directory.
2. Once the directory contains the Thing Descriptions and a Building Description related to the metadata of a building and its devices, the service provider should send a SPARQL query to the IoT platform, which includes the directory. The goal of the SPARQL query is to check the compliance of an IoT application in relation to the existing IoT infrastructure of a building. For instance, a service provider would like to check the compliance of its smart heating application for a given user in a given building. As a result, the service provider should send a SPARQL query to the WoT directory to check if the user apartment contains the required devices for the application, e.g., dCO:heatPump, dCO:temperatureSensor, dCO:heatPumpRelay, and dCO:powerMeter. The triplestore fetches the received SPARQL query from the service provider. If the application is compliant with the existing IoT infrastructure, the IoT platform returns a success message to the service provider. For example, the apartment contains the required devices to run the smart heating application.
3. Once an application is compliant with the IoT platform of a building, the service provider can ask for the required metadata for the application, e.g., the Thing Descriptions related to a heat pump appliance. Therefore, the service provider sends a SPARQL query to the IoT platform to ask for the required metadata. The IoT platform returns to the service provider the requested metadata. For instance, the service provider asks for the metadata related to the user building metadata and the Thing Descriptions of the installed devices related to some specific appliances. The Thing Descriptions contain a description of the WoT API access to the installed devices. As a result, the service provider will be able to run its smart heating application for the building.

4. Once a given application is compliant with a building, a user can request the usage of the application. For instance, a building user can ask for the usage of the compliant smart heating application. Then, the user can be registered to this smart application to run in their apartment.

4.3. Semantic Annotation of Things Descriptions and Buildings Descriptions

Semantic annotation is needed to allow humans and machines to have a common understanding of Thing Descriptions and Building Descriptions. In this section, we present an example of a heat pump to demonstrate a real-world use case of implementing semantic interoperability in a building. This building contains 2 floors and 2 apartments on every floor. Every apartment contains a heat pump appliance. Each appliance offers two services, a domestic hot water heating service, and a space heating service. The heat pump heats the water of a water boiler. This boiler delivers heat to the apartment floor to ensure space heating. Moreover, the water boiler delivers the hot water for the domestic hot water heating service. A heat pump relay enables and disables the appliance. A power meter measures the power, voltage, and energy consumption of the heat pump. An ambient sensor measures the ambient temperature and humidity of the heated space. A temperature sensor measures the water temperature of the boiler. The apartment contains also a smart meter, which measures the energy and power consumption of the entire apartment. To leverage semantic interoperability to the heat pump appliance, we define a set of Thing Descriptions for each of the former devices and a Building Description to describe the metadata of the apartment and its appliance. We present in the following JSON-LD of figure 4 and figure 5, respectively, the Thing Description of an ambient sensor and a heat pump relay from the heat pump use case. Semantic annotations are referred in the JSON-LD using the prefix dco of the dCO. Each JSON-LD document begins by the context attribute. In this attribute we mention the reused semantic resource (dCO) to annotate the document. For simplicity reasons, we do not include the WoT security scheme in these examples.

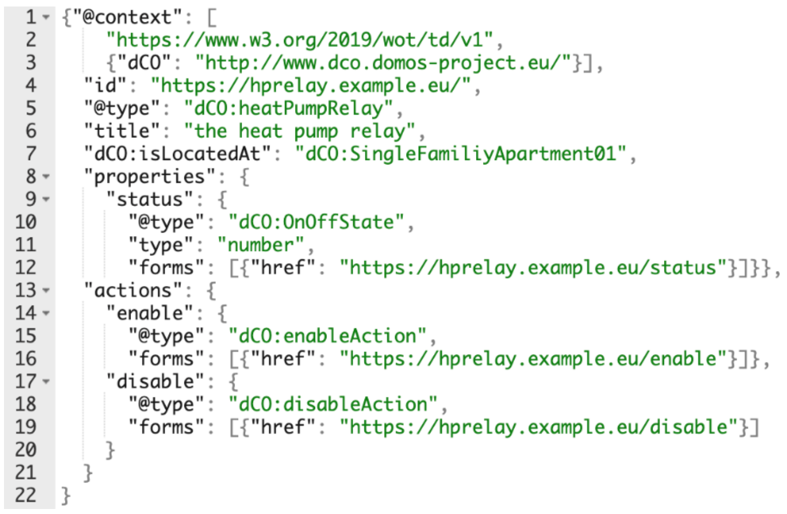
In the Thing Description JSON-LD snippet of Figure 4 is an ambient sensor located at a single-family apartment. This sensor observes the temperature and humidity properties (line 10 and line 15) measured respectively in Celsius and percentage (line 12 and line 17). This Thing Description includes an overheating event. Using the dCO, we semantically annotate the temperature property, humidity property, overheating event (line 21), units of measurement, location of the sensor (line 7), and the sensor type (line 5).



```
{"@context": [
    "https://www.w3.org/2019/wot/td/v1",
    {"dCO": "http://www.dco.domos-project.eu/"}],
  "id": "https://ambtsensor.example.eu/",
  "@type": "dCO:AmbientSensor",
  "title": "the Ambient sensor",
  "dCO:isLocatedAt": "dCO:SingleFamiliyApartment01",
  "properties": {
    "temperature": {
      "@type": "dCO:temperature",
      "type": "number",
      "unit": "dCO:Celcius",
      "forms": [{"href": "https://ambtsensor.example.eu
        /temperature"}]},
    "humidty": {
      "@type": "dCO:Humidty",
      "type": "number",
      "unit": "dCO:Percentage",
      "forms": [{"href": "https://ambtsensor.example.eu
        /humidity"}]}},
  "events": {
    "overheating": {
      "@type": "dCO:temperatureOverheating",
      "data": {"type": "string"},
      "forms": [{"href": "https://ambtsensor.example.eu/oh"}]
    }
  }
}
```

FIGURE 4: SEMANTIC ANNOTATION OF A THING DESCRIPTION OF AN AMBIENT SENSOR

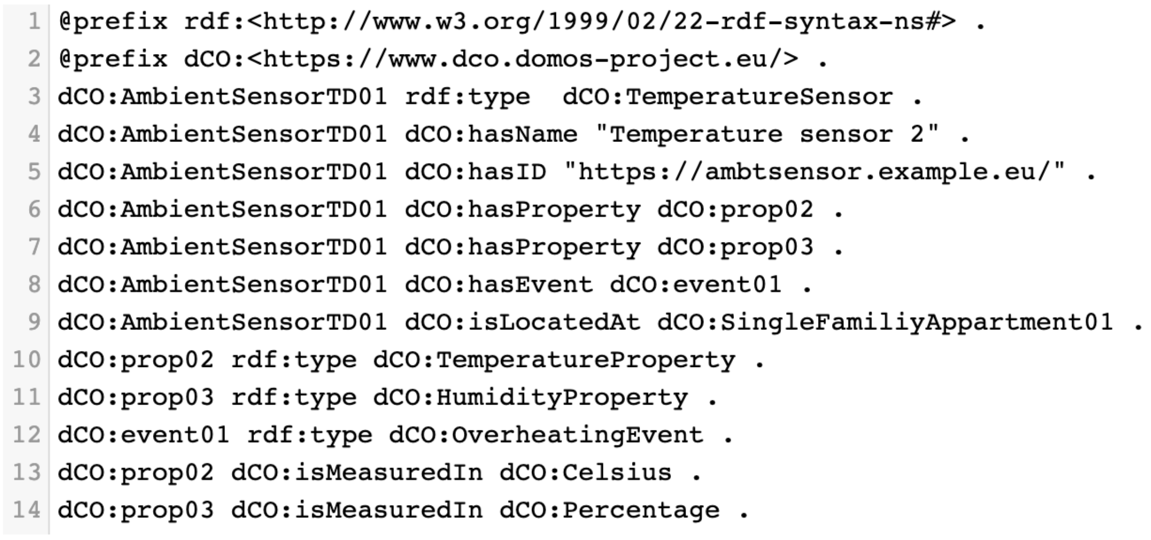
The Thing Description JSON-LD snippet of Figure 5 is a heat pump relay located at a single-family apartment. This relay has a status property, an enable action and a disable action. Using the dCO, we semantically annotate the status property (line 10), enable action (line 15), disable action (line 18), location of the relay (line 7) and its type (line 5).



```
 1 {"@context": [
 2     "https://www.w3.org/2019/wot/td/v1",
 3     {"dCO": "http://www.dco.domos-project.eu/"}],
 4   "id": "https://hprelay.example.eu/",
 5   "@type": "dCO:heatPumpRelay",
 6   "title": "the heat pump relay",
 7   "dCO:isLocatedAt": "dCO:SingleFamiliyApartment01",
 8   "properties": {
 9     "status": {
10       "@type": "dCO:OnOffState",
11       "type": "number",
12       "forms": [{"href": "https://hprelay.example.eu/status"}]}},
13   "actions": {
14     "enable": {
15       "@type": "dCO:enableAction",
16       "forms": [{"href": "https://hprelay.example.eu/enable"}]},
17     "disable": {
18       "@type": "dCO:disableAction",
19       "forms": [{"href": "https://hprelay.example.eu/disable"}]
20     }
21   }
22 }
```

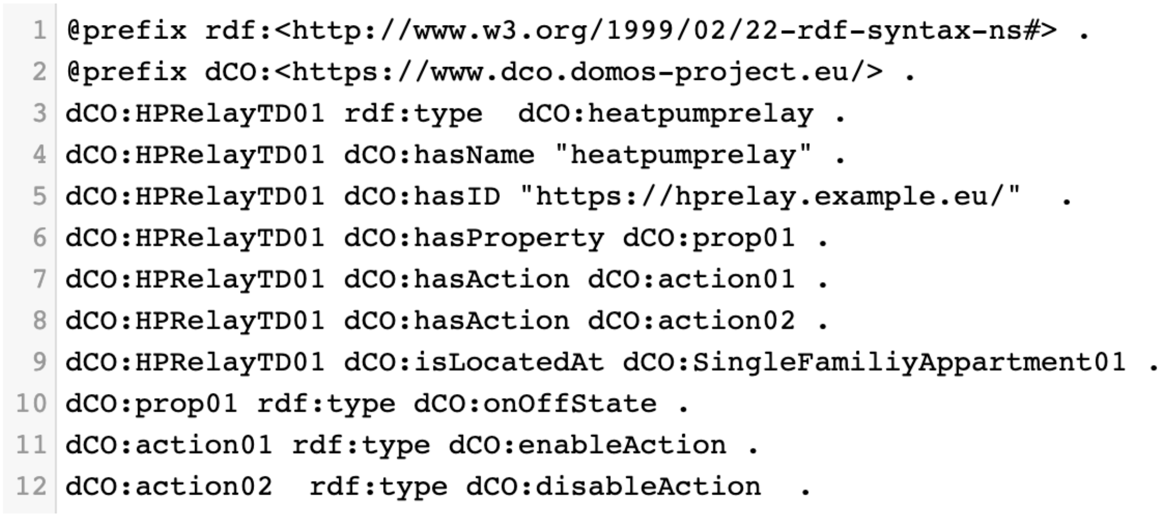
FIGURE 5: SEMANTIC ANNOTATION OF A THING DESCRIPTION OF HEAT PUMP RELAY

Based on the dCO and a serialization script, we automatically serialize the JSON-LD Thing Descriptions into RDF triples. RDF triples are instantiated in the triplestore to allow the semantic discovery capability. We depict in Figure 6 and Figure 7 the generated RDF from the Thing Descriptions of the ambient sensor and the heat pump relay, respectively.



```
@prefix rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix dCO:<https://www.dco.domos-project.eu/> .
dCO:AmbientSensorTD01 rdf:type  dCO:TemperatureSensor .
dCO:AmbientSensorTD01 dCO:hasName "Temperature sensor 2" .
dCO:AmbientSensorTD01 dCO:hasID "https://ambtsensor.example.eu/" .
dCO:AmbientSensorTD01 dCO:hasProperty dCO:prop02 .
dCO:AmbientSensorTD01 dCO:hasProperty dCO:prop03 .
dCO:AmbientSensorTD01 dCO:hasEvent dCO:event01 .
dCO:AmbientSensorTD01 dCO:isLocatedAt dCO:SingleFamiliyAppartment01 .
dCO:prop02 rdf:type dCO:TemperatureProperty .
dCO:prop03 rdf:type dCO:HumidityProperty .
dCO:event01 rdf:type dCO:OverheatingEvent .
dCO:prop02 dCO:isMeasuredIn dCO:Celsius .
dCO:prop03 dCO:isMeasuredIn dCO:Percentage .
```

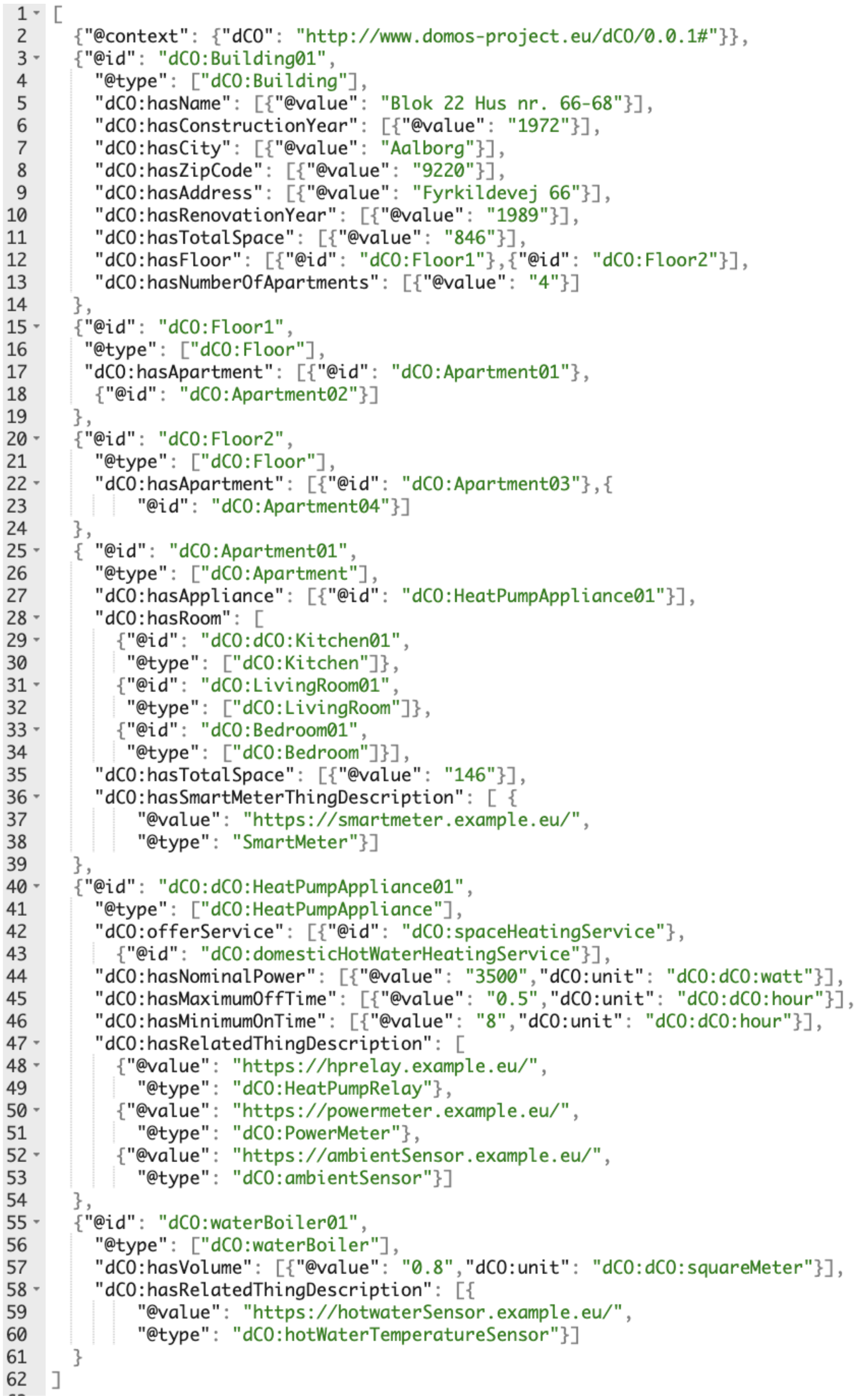
FIGURE 6: RDF TRIPLES SERIALISATION OF THE THING DESCRIPTION OF THE HEAT AMBIENT SENSOR



```
@prefix rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix dCO:<https://www.dco.domos-project.eu/> .
dCO:HPRelayTD01 rdf:type  dCO:heatpumprelay .
dCO:HPRelayTD01 dCO:hasName "heatpumprelay" .
dCO:HPRelayTD01 dCO:hasID "https://hprelay.example.eu/"  .
dCO:HPRelayTD01 dCO:hasProperty dCO:prop01 .
dCO:HPRelayTD01 dCO:hasAction dCO:action01 .
dCO:HPRelayTD01 dCO:hasAction dCO:action02 .
dCO:HPRelayTD01 dCO:isLocatedAt dCO:SingleFamiliyAppartment01 .
dCO:prop01 rdf:type dCO:onOffState .
dCO:action01 rdf:type dCO:enableAction .
dCO:action02  rdf:type dCO:disableAction  .
```

FIGURE 7: RDF TRIPLES SERIALISATION OF THE THING DESCRIPTION OF THE HEAT PUMP RELAY

Figure 8 shows an example of a JSON-LD Building Description semantically annotated using the dCO. This Building Description represents a building that contains 4 apartments (line 13) from the former use case. For simplicity reasons, we represent in the Building Description the metadata of only the first apartment (e.g., existing appliances in the apartment and rooms). Other apartments follow the same representation using the dCO. The JSON-LD Building Description file begins by presenting metadata related to the building itself (line 3-13) such as its name (line 5), construction year (line 6), address (line 9), existing floors (line 12), and the number of apartments. The file states the apartments for each floor. Then, the building description states the heat pump appliance (line 27) of the first apartment. The building description file also describes the heat pump appliance existing the first apartment. The dco:HeatPumpAppliance01 offers two services dco:spaceHeatingService and dCO:domesticHotWaterHeatingService.



```
[
  {"@context": {"dCO": "http://www.domos-project.eu/dCO/0.0.1#"}},
  {"@id": "dCO:Building01",
    "@type": ["dCO:Building"],
    "dCO:hasName": [{"@value": "Blok 22 Hus nr. 66-68"}],
    "dCO:hasConstructionYear": [{"@value": "1972"}],
    "dCO:hasCity": [{"@value": "Aalborg"}],
    "dCO:hasZipCode": [{"@value": "9220"}],
    "dCO:hasAddress": [{"@value": "Fyrkildevej 66"}],
    "dCO:hasRenovationYear": [{"@value": "1989"}],
    "dCO:hasTotalSpace": [{"@value": "846"}],
    "dCO:hasFloor": [{"@id": "dCO:Floor1"},{"@id": "dCO:Floor2"}],
    "dCO:hasNumberOfApartments": [{"@value": "4"}]
  },
  {"@id": "dCO:Floor1",
   "@type": ["dCO:Floor"],
   "dCO:hasApartment": [{"@id": "dCO:Apartment01"},
    {"@id": "dCO:Apartment02"}]
  },
  {"@id": "dCO:Floor2",
    "@type": ["dCO:Floor"],
    "dCO:hasApartment": [{"@id": "dCO:Apartment03"},{
        "@id": "dCO:Apartment04"}]
  },
  { "@id": "dCO:Apartment01",
    "@type": ["dCO:Apartment"],
    "dCO:hasAppliance": [{"@id": "dCO:HeatPumpAppliance01"}],
    "dCO:hasRoom": [
      {"@id": "dCO:dCO:Kitchen01",
       "@type": ["dCO:Kitchen"]},
      {"@id": "dCO:LivingRoom01",
       "@type": ["dCO:LivingRoom"]},
      {"@id": "dCO:Bedroom01",
       "@type": ["dCO:Bedroom"]}],
    "dCO:hasTotalSpace": [{"@value": "146"}],
    "dCO:hasSmartMeterThingDescription": [ {
        "@value": "https://smartmeter.example.eu/",
        "@type": "SmartMeter"}]
  },
  {"@id": "dCO:dCO:HeatPumpAppliance01",
    "@type": ["dCO:HeatPumpAppliance"],
    "dCO:offerService": [{"@id": "dCO:spaceHeatingService"},
      {"@id": "dCO:domesticHotWaterHeatingService"}],
    "dCO:hasNominalPower": [{"@value": "3500","dCO:unit": "dCO:dCO:watt"}],
    "dCO:hasMaximumOffTime": [{"@value": "0.5","dCO:unit": "dCO:dCO:hour"}],
    "dCO:hasMinimumOnTime": [{"@value": "8","dCO:unit": "dCO:dCO:hour"}],
    "dCO:hasRelatedThingDescription": [
      {"@value": "https://hprelay.example.eu/",
        "@type": "dCO:HeatPumpRelay"},
      {"@value": "https://powermeter.example.eu/",
        "@type": "dCO:PowerMeter"},
      {"@value": "https://ambientSensor.example.eu/",
        "@type": "dCO:ambientSensor"}]
  },
  {"@id": "dCO:waterBoiler01",
    "@type": ["dCO:waterBoiler"],
    "dCO:hasVolume": [{"@value": "0.8","dCO:unit": "dCO:dCO:squareMeter"}],
    "dCO:hasRelatedThingDescription": [{
        "@value": "https://hotwaterSensor.example.eu/",
        "@type": "dCO:hotWaterTemperatureSensor"}]
  }
]
```

FIGURE 8: BUILDING DESCRIPTION JSON-LD SEMANTICALLY ANNOTATED

5. Conclusion

This document presents the first version of the dCO. We first presented the used methodology for the dCO development. We provided a description of the Ontology Requirement Specification Document (ORSD) collected from our collaborators. The ORSD includes the scope, purpose, uses cases, and functional and non-functional requirements of the dCO. Based on the ORSD, we built and described the first version of the dCO. We finished the deliverable by presenting the implementation of semantic discovery and compliancy checking based on the dCO. We also presented a use case of employing the semantic annotation of Thing Description and Building Description. As future work, we will continue to maintain and extend the dCO based on the new requirements that may arise.

6. References

Balaji B. [et al.] Brick: Towards a unified metadata schema for buildings. [Conference] // Proceedings of the 3rd ACM International Conference on Systems for Energy-Efficient Built Environments.. - 2016.

Charpenay V. and Käbisch S. On Modeling the Physical World as a Collection of Things: The W3C Thing Description Ontology [Book Section]. - 2020.

Cimmino Arriaga A. [et al.] VICINITY: IoT Semantic Interoperability Based on the Web of Things [Conference] // 15th International Conference on Distributed Computing in Sensor Systems (DCOSS). - 2019.

Daniele L., den Hartog F. and Roes J. Created in Close Interaction with the Industry: The Smart Appliances REFerence (SAREF) Ontology [Conference] // International Workshop Formal Ontologies Meet Industries. - 2015.

European Commission Energy performance of buildings directive [Online]. - July 12, 2021. - https://ec.europa.eu/energy/topics/energy-efficiency/energy-efficient-buildings/energy-performance-buildings-directive_en.

Haller A. [et al.] The Modular SSN Ontology: A Joint W3C and OGC Standard Specifying the Semantics of Sensors, Observations, Sampling, and Actuation [Journal] // Semantic Web. - 2018. - Vol. 10.

Hugo W. [et al.] FAIR Semantics Recommendations Second Iteration [Online]. - December 18, 2020. - https://zenodo.org/record/4314321#.YSeZ4HUzZph.

Katipamula S. [et al.] Small-and medium-sized commercial building monitoring and controls needs: a scoping study, Tech. [Report]. - [s.l.] : U.S. Department of Energy, 2012.

Poveda-Villalon M. and Garcia-Castro R. Extending the SAREF ontology for building devices and topology [Conference] // Proceedings of the 6th Linked Data in Architecture and Construction Workshop. - 2018.

Rasmussen M.H. [et al.] BOT: the Building Topology Ontology of the W3C Linked Building Data Group [Journal] // Semantic Web. - 2020.

Rijgersberg H., Assem M. and Top J. Ontology of units of measure and related concepts [Journal] // Semantic Web. - 2013. - Vol. 4.

Suárez-Figueroa M.C., Gómez-Pérez A. and Fernández-López M. The NeOn Methodology framework: A scenario-based methodology for ontology development [Journal] // Applied ontology. - 2015. - Vol. 10.

W3C Web of Things (WoT) Architecture [Online] // W3C Web Site. - April 9, 2020. - https://www.w3.org/TR/wot-architecture/.

W3C Web of Things (WoT) Thing Description [Online] // W3C Web Site. - April 9, 2020. - https://www.w3.org/TR/wot-thing-description/.

Wilkinson M.D. [et al.] The FAIR Guiding Principles for scientific data management and stewardship [Journal] // Scientific data. - 20116. - Vol. 3.