Ref. Ares(2023)1659814 - 07/03/2023

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

Operating System for Smart Services in Buildings

D3.5 domOS Common Ontology, Final Version

WP3 Common Ontology and Semantics

	Name	Date
Prepared by	Amir Laadhar (AAU), Christian Thomsen (AAU), Fabio Lilliu (AAU), Torben Bach Pedersen (AAU)	17.01.2023
Peer reviewed by	Dominique Gabioud (HES-SO)	24.01.2023
Reviewed and approved by	Dominique Gabioud (HES-SO)	03.03.2023

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	17.01.2023	First version by AAU
2.1	14.02.2023	Second version by AAU
3.1	03.03.2023	Final version by HES-SO

To be cited as

"D3.5 domOS Common Ontology, Final Version" of the HORIZON 2020 project domOS, EC Grant Agreement No 894240.

Disclaimer

This document's contents are not intended to replace the consultation of any applicable legal sources or the necessary advice of a legal expert, where appropriate. All information in this document is provided "as is" and no guarantee or warranty is given that the information is fit for any particular purpose. The user, therefore, uses the information at its sole risk and liability. For the avoidance of all doubts, the European Commission has no liability in respect of this document, which is merely representing the authors' view.

Table of Contents

Executive Summary **5**

1. Introduction **5**

2. Relevant Existing Ontologies for domOS Common Ontology **6**

2.1. Smart Applications REFerence Ontology (SAREF) 6

2.2. SAREF4ENER: An Extension of SAREF for the Energy 7

2.3. W3C Thing Description (TD) Ontology 7

2.4. The Building Topology Ontology (BTO) 7

2.5. The Ontology of Units of Measure (OM) 8

3. dCO Ontology Engineering Methodology and Development **8**

4. Ontology Requirements Specification **9**

5. dCO Design **10**

5.1. dCO Ontology Design Overview 10

5.2. Building Description Ontological Design 12

5.3. Energy Flexibility Ontological Design 16

6. dCO in Action **20**

6.1. TD Annotation 20

6.2. Building Description Example 20

6.3. FlexOffer Representation Example Using dCO 21

7. Conclusion **25**

8. References **26**

List of figures

Figure 1: dCO Design Overview 12

Figure 2: Building Description Design Overview 13

Figure 3: Building Structure Design 14

Figure 4: Devices Metadata Design and Integration With Thing Description 15

Figure 5: Energy, Electricity, and Heating Flux 16

Figure 6: Energy Flexibility Design 19

Figure 7: An Annotated Thing Description 20

Figure 8: Example of a Standard FlexOffer 22

Figure 9: Example of Total Energy Constraint FlexOffer 22

Figure 10: Example of Dependency FlexOffer 23

Figure 11: Example of Dependency FlexOffer, in a *Switching* Case 23

Terms, definitions, and abbreviated terms

GA	Grant Agreement
ICT	Information and Communication Technology
FO	FlexOffer
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 T3.3, namely the "D3.5 domOS Common Ontology, Final Version". This deliverable focuses on the final version of the domOS Common Ontology (dCO). The main objective of WP3 is to provide a common information model, i.e., the dCO ontology, 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

Buildings are responsible for approximately 40% of EU energy consumption and 36% of the greenhouse gas emissions. Buildings are the single largest energy consumer in Europe. Two-thirds of the building consumption is for residential buildings. Therefore, the domOS project focuses on residential buildings. Between 2000-2019, space heating is the dominant household consumption with 66% of the total consumption Driven by the availability of inexpensive IoT devices, buildings are being integrated with a variety of IoT devices to achieve energy efficiency using smart IoT services. IoT services collect real-time data and communicate with different IoT sensors to effectively reduce energy consumption while ensuring household comfort.

For instance, a smart space heating service integrates different IoT devices in a building. This service reduces energy consumption by dynamically controlling a heater based on intelligent algorithms and different sensors such as room temperature and humidity. For example, the heater is only enabled if the room temperature is below a given threshold. Therefore, the heating time of an apartment can be reduced without affecting the household comfort while reducing the energy consumption.

IoT devices available on the Web have grown in the last decade. One of the main challenges in the IoT landscape is achieving semantic interoperability between IoT data producers and IoT data consumers. To ensure semantic interoperability in the Web of Things (WoT), ontologies allow humans and machines to have a common understanding of the meaning of an IoT ecosystem through the use of a common vocabulary.

IoT consumers cannot be expected to be aware of every possible metadata of a smart building. It is necessary to rely on a semantic discovery capability to search and discover the required metadata of a smart building. For instance, a new service provider would like to know the existing installed devices, and their energy characteristics in an apartment. W3C Web of Things Thing Descriptions (TD) provide metadata that describes the affordances (properties, actions and events) and semantics of IoT devices. However, before a TD can be interpreted and used, it needs to be discovered. A WoT Thing Description Directory provides a searchable repository of a collection of Thing Descriptions. W3C WoT Discovery specifies an API for a WoT Thing Description Directory service that provides a search mechanism for collections of WoT Thing Descriptions that can be dynamically registered. The Thing Description Directory serves with a prescribed API that allows the registration, management, and search of a database of Thing Descriptions. However, this directory has limited search capabilities, since it stores only Thing

Descriptions without including the metadata of smart buildings, which can include the building space topology, the topology of energy meters, the relationships between IoT devices, the location of each device, and their ensured tasks.

The large penetration of renewable energy sources (e.g., wind energy, solar energy) results in challenges in the planning of energy production, transmission, and distribution. Therefore, it is challenging to control renewable energy production when the energy consumption is variable. This usually leads to a waste of energy. To overcome this challenge, one of the main solutions is the shift from energy production control to energy demand control. To ensure this shift, the potential of smart buildings should be extended beyond the buildings themselves to play their role as a flexible component of the electrical grid. Therefore, the energy flexibility service is responsible to run the home appliances flexibly. For example, when the wind is strong, homes can be heated up due to the availability of renewable energy.

To cope with the lack of building metadata in the WoT, we specify a Building Description based on the domOS Common Ontology (dCO) available at https://www.w3id.org/dco, which integrates the Thing Description ontology, SAREF core ontology, SAREF extension for the Energy domain, and Building metadata. Therefore, IoT service providers can query a common directory in order to search for building metadata in the context of the WoT.

To ensure the semantic interoperability of flexibility, we need to represent energy flexibility in dCO. The FlexOffer (FO) (Pedersen, et al., 2018) concept has been developed in a range of EU and Danish projects to describe flexibility. Consider, for example, a dish washer scenario where the earliest start time is 9pm and latest end time is 5am. During this 8-hour time flexibility period, the dish washer must run with the cheapest price while achieving the lowest CO2 footprint and taking into consideration the grid capacity. These constraints and any other requirements can be encoded in a FlexOffer for the dishwasher. Similarly, there could also be FOs for charging of EVs, heat pumps, and millions of other FOs to balance grid production with consumption. Therefore, in this deliverable we study the requirements of FO modelling, and we integrate the energy flexibility modelling of FOs in the dCO.

2. Relevant Existing Ontologies for domOS Common Ontology

2.1. Smart Applications REFerence Ontology (SAREF)

The Smart Applications Reference Ontology (SAREF) is an ontology for IoT devices and solutions published by ETSI in a series of Technical Specifications initially released in 2015 and updated in 2017. Even if its initial objective was to build a reference ontology for appliances relevant for energy efficiency, SAREF is not limited to this scope and can serve as an upper reference model to enable better integration of data from various vertical domains in the IoT. SAREF has been extended to different domains such as energy (SAREF4ENER), environment (SAREF4ENVI), buildings (SAREF4BLDG), smart cities (SAREF4CITY), agriculture (SAREF4AGRI), industry & manufacturing (SAREf4INMA). SAREF4ENER was created in collaboration with Energy@Home (http://www.energy-home.it) and EEBus (http://www.eebus.org/en), the major Italian and German industry associations, to enable the interconnection of their (different) data models. SAREF4ENER model flexibility. However, we cannot assume that this representation is rich

enough to model the Flex Offer that will be extended in T3.3 to be able to capture predictions, flexibility, and uncertainty of various dCO measures. SAREF is aligned to many other ontologies such as SSN and SOSA. One of the main criticisms of SAREF is that the ontology is not rich enough to model all the information in the IoT ecosystem. For instance, in SAREF and its extensions, the representation of the various entities related to the actuator is limited (Seydoux, 2018). Therefore, knowledge scientists their IoT ontologies by reusing the multiple state-of-the-art ontologies such as SAREF, its extensions, and SSN.

2.2. SAREF4ENER: An Extension of SAREF for the Energy

SAREF4ENER (SAREF for Energy) is an extension of SAREF (Semantic Sensor Network Ontology) that aims to provide a standardized way to describe and interoperate energy-related devices, systems, and services. SAREF4ENER is built on top of SAREF and extends it with new classes, properties, and rules specific to the energy domain. It allows to describe devices such as solar panels, electric vehicles, and smart meters, as well as systems like microgrids, and services like energy management or demand response. By using SAREF4ENER, developers can create applications that can easily integrate and communicate with different energy devices, systems, and services, regardless of the manufacturer or protocol used. This leads to a more efficient and sustainable energy management.

2.3. W3C Thing Description (TD) Ontology

The W3C Thing Description (TD) (Charpenay, et al., 2020)Ontology is an ontology developed by the World Wide Web Consortium (W3C) to describe the characteristics and capabilities of IoT devices and services. It defines a set of classes, properties, and relationships that can be used to describe the physical and functional properties of a device, as well as its communication and security capabilities. The TD Ontology enables device interoperability by providing a common language for describing devices, which allows them to be easily discovered, accessed, and controlled by other devices and applications. Additionally, the TD ontology is in line with other ontologies such as SAREF4ENER, which also aims to provide a standard way to describe IoT devices and systems. By using the TD ontology, developers can create applications that can easily integrate and communicate with different IoT devices and services, regardless of the manufacturer or protocol used, which leads to a more efficient and secure IoT ecosystem.

2.4. The Building Topology Ontology (BTO)

The Building Topology Ontology (BTO) (Rasmussen, 2021) is an ontology that describes the physical structure and layout of buildings, including the rooms, floors, and building components such as walls, doors, and windows. It is designed to provide a standardized way to represent and interoperate building information across different applications and domains. The BTO is based on the OWL (Web Ontology Language) and RDF (Resource Description Framework) standards, which allows it to be easily integrated with other ontologies, such as the SAREF4ENER and the W3C Thing Description Ontology.

The BTO can be used to describe the physical and functional properties of a building, such as its size, shape, and materials, as well as the layout of rooms, corridors, and staircases. Additionally, it can be used to describe the location and properties of building components, such as windows, doors, and walls, which can be important for energy management, security, and emergency response. By using the BTO, developers can create applications that can easily navigate and understand the physical structure of a

building, which can be useful for various applications such as space management, energy management, building automation, and emergency response.

2.5. The Ontology of Units of Measure (OM)

The Ontology of Units of Measure (OM) is a framework that provides a standard way to represent and describe units of measurement and their relationships. It is designed to be used in conjunction with other ontologies such as SAREF4ENER, W3C Thing Description Ontology, Building Topology Ontology, etc, to provide a consistent and precise way of expressing measurements across different domains and applications. The OM is based on the OWL (Web Ontology Language) and RDF (Resource Description Framework) standards and is aligned with the International System of Units (SI) and ISO standards for units of measure.

3. dCO Ontology Engineering Methodology and Development

In this section, we present the development process used for the dCO along with the used ontology engineering methodology. Many ontology engineering methodologies are available and each of them may lack some features. Among the many methodologies of ontology engineering, the NeOn approach (Suárez-Figueroa, et al., 2012)is the most relevant and a mature methodology to adopt for dCO development. However, state-of-the-art ontology engineering methodologies, including NeOn, lack integration of the FAIR principles (Wilkinson, 2016). We reuse the following workflow of activities from the NeOn approach: the Ontology Requirement Specification Activity, the Knowledge Acquisition Activity, the Conceptualization Activity, the Implementation Activity, and the Ontology Support Activities. The Ontology Support Activities should be performed during all the development process. We define an Ontology FAIRification Activity included in the Ontology Support Activities in order to implement FAIR principles. In the following, we present each of these activities and how they are applied to develop the dCO.

Activity 1: Ontology Requirement Specification Activity: The output of this activity is an Ontology Requirement Specification Document (ORSD) that covers the scope, use cases and functional and non-requirements of the ontology. Functional requirements can be stated as Competency Questions (CQs).

Activity 2: Knowledge Acquisition Activity: This activity is not well detailed by NeOn despite its importance. We specify the output of this activity as a knowledge document. This document contains the terminology and glossary of the ontology. Based on the input obtained in collaboration with the five domOS demonstration sites, we built a terminology and a glossary. The terminology is a finite list of terms (e.g., device, sensor, appliance). The glossary is a finite set of terms associated with textual definitions, possible synonyms, the possibly reused state-of-the-art ontology terms, and the semantic type of each term (class, property, and instance). Different external ontologies can represent the same term. As a result, we need to define a guideline to choose a single term to reuse for each term of the glossary. Therefore, we give priority to reuse terms existing respectively in the following types of ontologies: standardized domain ontologies (SSN, SOSA, SAREF), upper-level ontologies (BFO), domain ontologies (iottaxolite), and application ontologies (vicinity). If we find a set of candidate terms to be reused existing

in the same type of ontology, we choose the term having the best ranking score on Linked Open Vocabularies. We also defined a column in the glossary to classify every term into one of the following 5 modules: Building Description module, Thing Description module, device module, units of measurement module, and dCO core module. We have opted to design dCO in modules to increase the maintainability, reusability, and scalability of the ontology.

Activity 3: Conceptualization Activity: In this activity, we make a conceptualization, in which the ontology components are structured in a meaningful model, which can be understood by ontology practitioners. This is a conceptual model of the ontology representing the structure of the classes and property relationships of the ontology.

Activity 4: Implementation Activity: We are based on the knowledge document and the conceptual model as an input to implement dCO using Protege, in which the Ontology Language (e.g., OWL, RDF) is generated. We encode the input given by the glossary and the conceptualization as classes, instances, axioms, object properties, or data properties. To increase the FAIRness of dCO, we include a set of metadata associated with the ontology itself and the metadata associated with its elements (classes, object properties, datatype properties, and individuals). For instance, some of recommended metadata properties to describe ontologies are dcterms:license, dcterms:creator, dcterms:contributor, and owl:priorVersion. The recommended metadata properties to describe classes, properties, data properties and individuals include rdfs:label and rdfs:comment.

Activity 5: Ontology Support Activities: This set of activities includes documentation, assessment, evaluation, and an Ontology FAIRification Activity. The activities of documentation, assessment, and evaluation are as given by NeOn. Therefore, we focus on our new Ontology FAIRification Activity. Garijo et al. (Garijo, et al.)studied the FAIR principles and proposed best practices for implementing FAIR principles in ontologies. We choose to apply these best practices to ensure the FAIRness of the dCO. These best practices include using permanent URIs, using ontology metadata properties, and making the ontology findable on the Web. We note that a general agreement on these best practices is an open issue for the overall community and thus out of the scope for this deliverable.

4. Ontology Requirements Specification

The Ontology Requirement Specification Document (ORSD) is made in a spreadsheet in collaboration with our partners from the five domOS demonstration sites. In the following, we present the purpose and use cases of the ontology.

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

Scope: The dCO ontology has to focus on the IoT domain. The level of granularity is directly related to the identified terms from the functional requirements.

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, which are JSON-LD files that describe the metadata of a WoT Thing. Thing 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 device metadata.

(ii) Semantic validation of the annotated Thing Descriptions. Once Thing Descriptions are semantically annotated, they should be semantically validated to make sure that there are no inconsistencies.

(iii) Semantic discovery of IoT devices and their data based on a context-based search in smart buildings

(iv) Compliance check of new IoT services in the existing smart buildings.

(v) Semantic interoperability of energy flexibility using FlexOffers.

(vi) Elaboration of Building Descriptions as a nameplate for building metadata.

Ontology Requirements:

- **Non-Functional Requirements:**
  - NFR1: The dCO ontology should be compliant with FAIR principles.
  - NFR2: The dCO ontology should be designed in a modular way.
- **Functional Requirements:**
  - The ontology should represent:
    - Energy flexibility
    - Building structure, metadata, and measurements
    - Power, heat, electricity flow in a building

5. dCO Design

5.1. dCO Ontology Design Overview

dCO is based on a modular design to enhance the maintainability of the ontology. More detailed graphical representations of the modules are provided in the dCO ontology documentation https://www.w3id.org/dco. The reused ontologies and their prefixes are mentioned in Figure 1. In the following, we state the dCO modules:

dCO core module: this module reuses the SAREF core ontology and allows the integration of the other modules to ensure semantic interoperability of the WoT ecosystems in smart buildings. This module is integrated with the thing description module and the device module using a subsumption relationship. The core module is integrated with the building description module and the units of measurement module respectively using the property dco:isLocatedAt and dco:isMeasuredIn. Moreover, the four super classes of the power, electricity, and heating module are associated to dco:Device from the core module using the following object properties: dco:HasPartnerDispatcher, dco:IsConsumedBy, dco:HasProducer, dco:IsProducedBy, dco:HasConsumer, and dco:IsStoredIn.

The **building structure module** contains the representation of building-related metadata and building topology. In this module, we reuse some concepts from Building Topology Ontology (BTO) of the W3C

Linked Building Data Group. We also define a set of metadata related to the characteristics of a building. A dco:Device can be located at a dco:Space. We define five types of spaces: dco:Building, dco:Storey, dco:Apartment, dco: Stairway and dco:Room. A building must contain at least one floor, which can contain at least one apartment. An apartment can contain one or more rooms. A room can be categorized into several types (e.g., kitchen, living room).

Thing description module: this module reuses and extends the Thing Description ontology to model the WoT Thing Description interaction affordances, i.e., properties, actions, and events. This module integrates the W3C WoT Things Descriptions ontology with the dco core module using the subsumption relationship to dco:Device. Moreover, this module integrates a dco:InteractionAffordance with a dco:Service, dco:Space, dco:Consumer, and dco.Producer using respectively the properties dco:HasMCP, dco:HasMCP, dco:HasInputMCP, and dco:HasOutputMCP. This integration allows the discovery of interaction affordance related to different contexts without focusing on the related devices.

Device module: this module represents the set of devices in an IoT ecosystem. Devices are structured in categories (subclasses), that reflect different types of devices (e.g., dco:Meter).

Power, electricity, and heating flow module: this module represents the flow of power, electricity, and heating between devices and spaces in a Building. The ontological design of the energy/heating/electricity flux has four main abstraction classes: dco:Hub, dco:Storage, dco:Consumer, and dco:Producer. Each of these classes has three specialization subclasses related to power, electricity, and heating. The subclasses inherit the data and object properties of the four main abstractions classes. For example, the dco:Producer has three subclasses: dco:PowerProducer, dco:ElectricityProducer, and dco:HeatProducer.

Energy flexibility module: this module reuses and extends SAREF4ENER: extension for the Energy domain in order to model FlexOffers (see also Section 5.3).

Units of measurements module: this module represents all the units of measurements of the IoT devices. This module reuses the Units of Measurement Ontology (Rijgersberg, et al.). A saref:Measurement is a measurement using a unit of measurement (saref:unitOfMeasurement), which is structured in categories that reflect the different types of units of measurements (e.g., om:energyUnit).

In Figure 1, we present the main classes and properties of the domOS ontology.

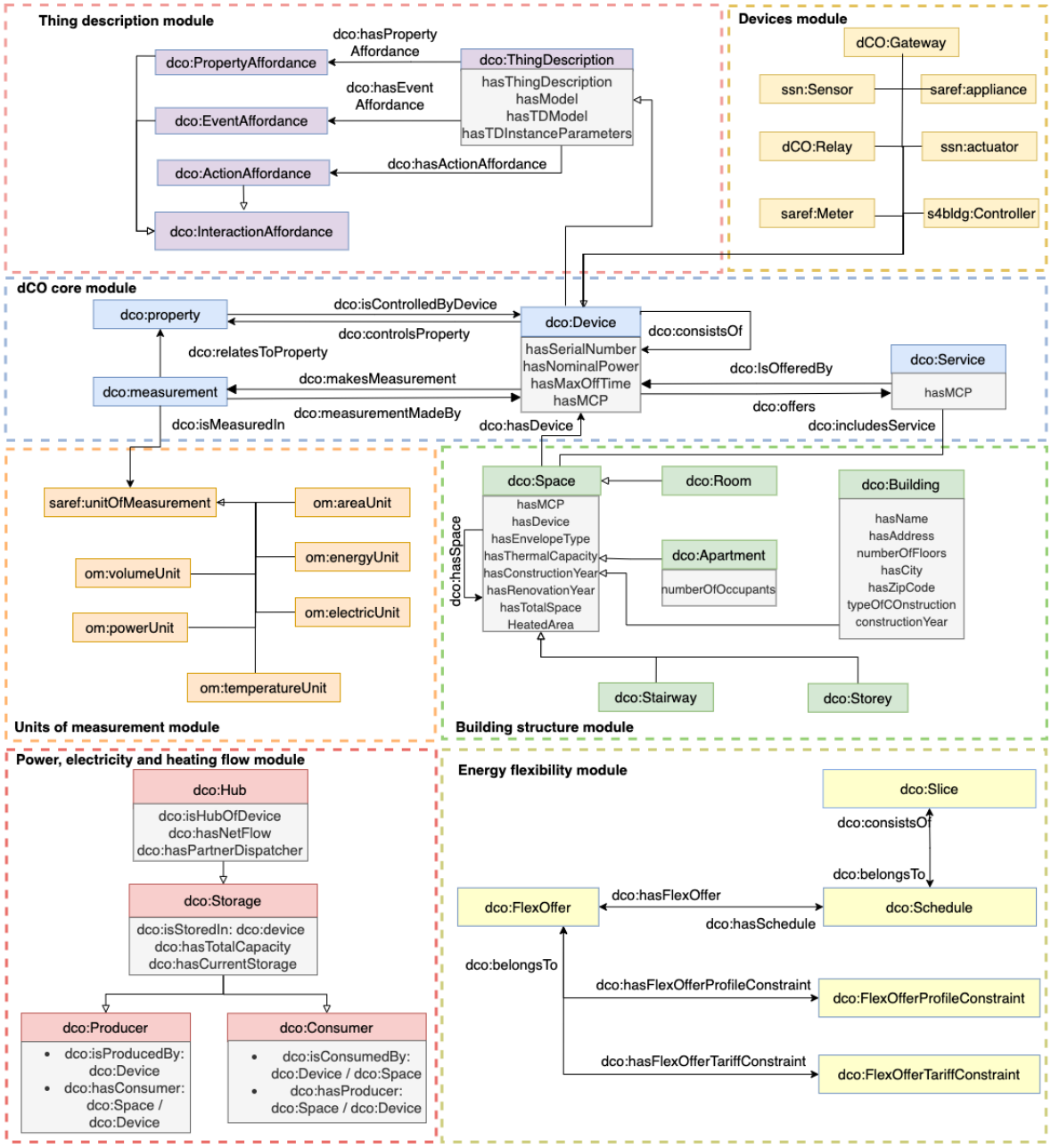


FIGURE 1: DCO DESIGN OVERVIEW

5.2. Building Description Ontological Design

A Building Description (BD) is a machine- readable description that provides an abstract description of a building and its properties as needed by the smart services (or applications) controlling (the energy of) the building. It consists of a description of building level - specific metadata (e.g., total space, thermal capacity, envelope type), a description of the structure of the building (e.g., storeys, apartments, rooms), the energy/electricity/heat flow between spaces and devices in the building, the measurements and control points in that building and links to the things and services installed in the building. Specifically, The Building Description is a JSON-LD file based on the domOS Common Ontology. The Building

Descriptions can be stored in a Building Description Directory (BDD) to allow the CRUD functions and the semantic querying.

The purpose of a BD is to open up a building to enable multiple application smart energy services from different providers to run in the building, and to allow applications (smart services) be interoperable (or easily portable) across domOS platform instances hosting a and variety of buildings with various kinds of equipment installed devices An application should be able to function in a domOS building if the application requirements are met by the building.

In the following, the main components of a Building Description are described:

- A description of **building-specific metadata** (e.g., building type, total heated area, number of apartments, construction year, thermal characteristics, envelope type)
- The **structure of the building** and the relationships between the spaces (e.g., storeys, apartments, rooms) of the building
- **Building services** (e.g., dco:spaceHeatingService). The metadata of services can be defined along with the list of the involved devices. The WoT energy affordances are associated to services.
- **The list of the device types** that are installed in the building, further static properties of the devices, (e.g., heating surface areas, maximum on time, and their capacities). The relationship between devices. The WoT affordance, TD models and TD instances of devices.
- **The energy, electricity and heating flux,** outflow and inflow between devices and spaces.
- **The measurement or control points (MCPs)**

The following Figure 2 gives an overview of the main classes and properties of a Building Description.

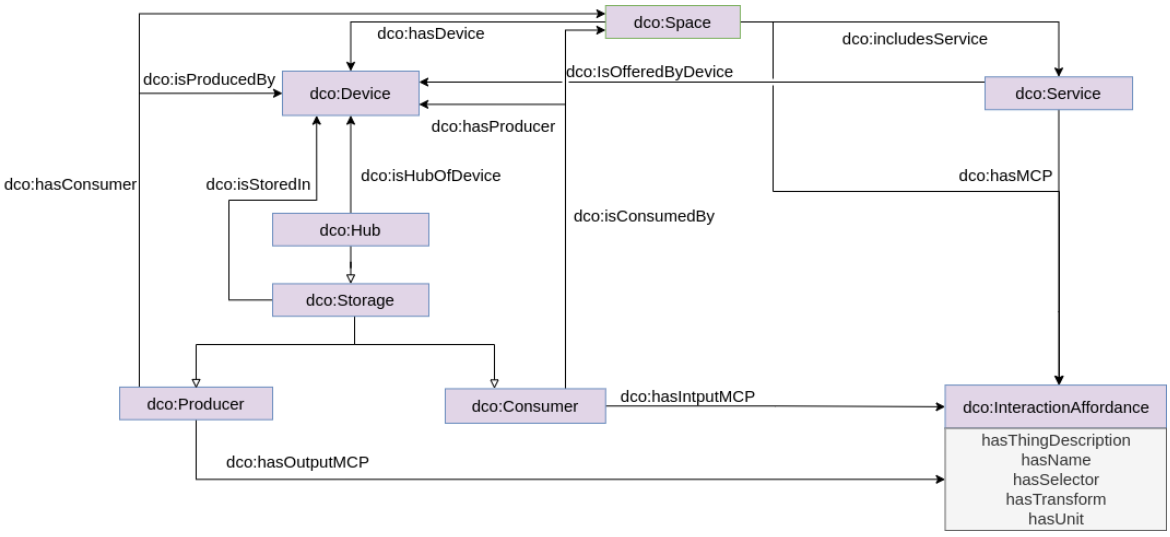


FIGURE 2: BUILDING DESCRIPTION DESIGN OVERVIEW

The Structure of a Building and Description of Building-Specific Metadata

The building description contains the representation of building-related metadata and building topology. We also define a set of metadata related to the characteristics of a building. A dco:Device can be located at a dco:Space using the property "dco:hasDevice". According to the Building Topology Ontology, a "dco:Space" is part of the physical world or a virtual world whose 3D spatial extent is bounded actually or theoretically. A "dco:Space" can be contained in other space using the property "dco:hasSpace". The object property "dco:hasSpace" has 4 sub properties: "dco:hasRoom", "dco:hasStorey", "dco:hasApartment", and "dco:hasBuilding". These properties can be used to link different spaces. We define four types of spaces: dco:Building, dco:Storey, dco:Apartment, and dco:Room. A building must contain at least one storey, which can contain one or more apartments. An apartment can contain one or more rooms. A room can be categorized into several types (e.g., kitchen, living room). A building can also be categorized into several types such as dco:House, dco:ResidentalBuilding, and dco:SingleFamilyHouseBuilding.

We associate a "dco:Space" to a "dco:Service" using the property "dco:includesService", which mentions the services offered by devices in a space. For example, an apartment can include the space heating service.

In the following Figure 3 is the ontological presentation of the building structure.

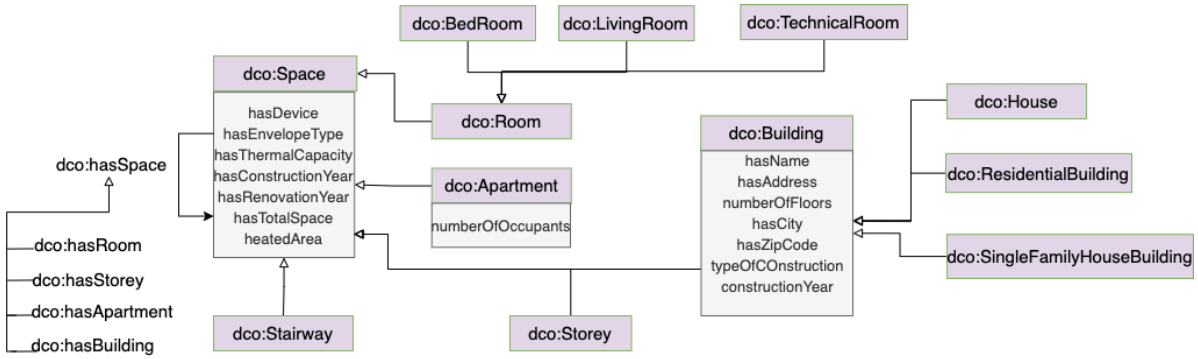


FIGURE 3: BUILDING STRUCTURE DESIGN

Building Services

A "dco:Service" 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 is offered by at least one Device that wants (a certain set of) its function(s) to be discoverable, registerable, and remotely controllable. A Service must specify the Device that is offering the Service. A "dco:Service" is associated to at least one "dco:Device" using the property "dco:isOfferedByDevice", which means that a service uses the one or more to devices to accomplish its function.

Devices Metadata

A dco:Device is a tangible object designed to accomplish a particular task. For example, a Device can be a washing machine, a heater, or a sensor. A Device is associated to a set of metadata properties such as

the serial number (dco:hasSerialNumber) or its maximum off time (dco:maxOffTime). We mention in the Thing Description the link to its API using the property "dco:hasThingDescription".

A dco:Device can be associated to a dco:ThingDescription, which can be associated to three types of interaction affordances: dco:PropertyAffordance, dco:eventAffordance, and dco:actionAffordance.

In the following Figure 4 is an ontological representation of the devices metadata and its integration with a thing description.

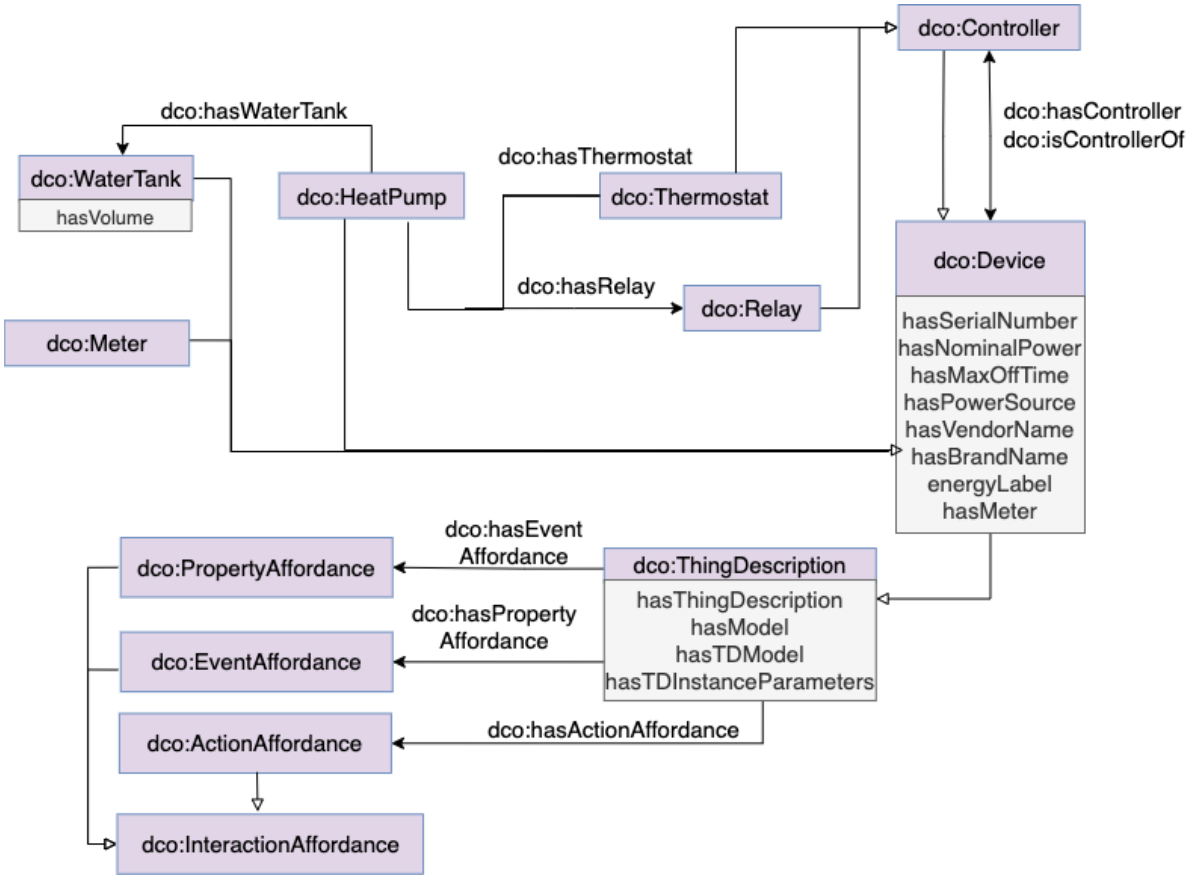


FIGURE 4: DEVICES METADATA DESIGN AND INTEGRATION WITH THING DESCRIPTION

Energy/Electricity/Heat Flux:

The ontological design of the energy/heating/electricity flux has four main abstraction classes:

- dco:Hub: a dco:Hub is associated to a device using the object property dco:isHubOfDevice: which acts at the same time as a dco:Producer, dco:Consumer and dco:Storage. Therefore, the dco:Hub can consume, produce and store heat/electricity/power. The dco:Hub class inherits properties from dco:Producer, dco:Storage and dco:Consumer.
- dco:Consumer is an abstract class to describe the flux of consumption of energy/heat/power. This class has the following properties:
    - dco:isConsumedBy, the range can be a dco:Device and/or dco:Space.
    - dco:hasProducer, the range can be a dco:Device and/or dco:Space

- dco:hasInputMeasurement: is an object property, which has a range of the possible measurements, which can be measured from a dco:Consumer

- - dco:Producer is an abstract class to describe the production of heat/energy/power. This class has the following properties:
  - dco:isProducedBy, the range can be a dco:Device
  - dco:hasConsumer, the range can be a dco:Device and/or dco:Space
  - dco:hasInputMeasurement: is an object property, which has a range of the possible measurements, which can be measured from a dco:Producer
- - dco:Storage inherits all the properties from dco:Consumer and dco:Producer. This class represents the storage of energy/heat/power.

The other classes of the design in Figure 5 below inherit the four main classes. For example, a dco:HeatConsumer is a dco:Consumer with the inherited properties of a dco:Consumer. In the representation of a BD related to a building we can use the leaf classes (i.e., dco:HeatHub, dco:EnergyProducer) to represent the flux of energy/power/heat.

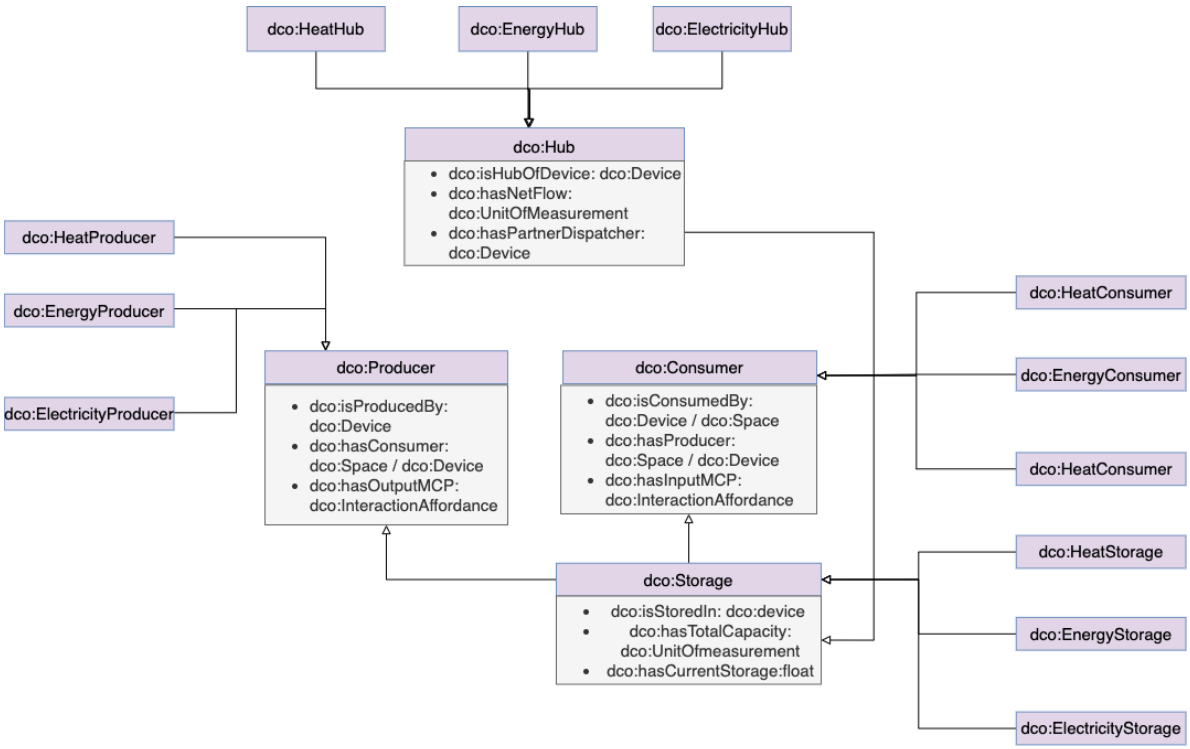


FIGURE 5: ENERGY, ELECTRICITY, AND HEATING FLUX

5.3. Energy Flexibility Ontological Design

In collaboration with partners, AAU-Daisy has developed a universal unified way to represent / model flexibility in electricity consumption and production, and this generalized model is called “FlexOffer” (Pedersen, et al., 2018). The concept of FlexOffer and the underlying technology have been developed in a number of EU H2020 and Danish Innovation Fund projects. A FlexOffer offers a standardized and unified way of describing flexibility from a wide range of flexible devices (resources). Irrespective of the device type, electrical energy amounts and associated flexibilities can be explicitly captured using a FlexOffer.

To ensure the semantic interoperability of the FlexOffer, we provide its ontological design, and we integrate it in the dCO ontology. This will allow a common understanding of the FlexOffer concept from different people and agents and also allows its integration in different systems.

We distinguish mainly between the following four types of FlexOffers.

- **Standard FlexOffer**: it is a FlexOffer defined by only start time constraints and slice (energy) constraints. We consider flexibility over discrete time horizons, divided in units called *time slices:* for example, if time slices are 15 minutes long, one hour will consist of four time slices. A start time constraint determines the earliest and latest time slice at which the load can start. An energy constraint establishes, for each time slice at which the load is operating, the minimum and maximum amount of energy that can be consumed from that load.
- **Total Energy Constraint FlexOffer**: it is a FlexOffer defined by the same constraints of Standard FlexOffers, plus a total energy constraint (TEC). This type of constraint specifies the lower (TEmin) and upper (TEmax) bounds for the total energy that can be consumed over all slices.
- **Dependency FlexOffer**: it is a FlexOffer defined by dependency energy constraints. A dependency energy constraint specifies at each time unit a lower and an upper bound on the amount of energy that can be consumed, depending on the total amount of energy that has been consumed before time unit t.
- **Uncertain FlexOffer**: it is a FlexOffer that takes uncertainty from flexibility approximation into account. There are two main types of uncertainty that are considered: time and amount uncertainty. In order for it to be optimized, a threshold for uncertainty *p* is chosen and, after that, a FlexOffer (Standard, or Dependency) is generated by considering amounts of energy with probability at least *p* to be available according to the Uncertain FlexOffer.

A FlexOffer (dco:FlexOffer) is subclass of (rdfs:subClassOf) SAREF4ENER power sequence (s4ener:PowerSequence). This means that a dco:FlexOffer inherits the object properties and data properties of s4ener:PowerSequence. We reuse terms from SAREF4ENER in the energy flexibility ontological design. We also define new terms based on the FlexOffer specification and requirements in case these terms do not exist in SAREF4ENER.

We represent the FlexOffer concept using the class dco:FlexOffer, which consists of (dco:ConsistsOf) at least one s4ener:Slot. A dco:FlexOffer, consists of (dco:ConsistsOf) of at least one dco:FlexOfferProfileConstraint, which is associated with the following properties:

- hasEnergyConstraintList: (max 1 s4ener:EnergyMin, max 1 s4ener:EnergyMax)
- s4ene:hasPrice: sarefPrice (Contains the list(s) of tariff constraints for one time unit)
- s4ener:MinDuration: The minimum duration of a FlexOffer
- s4ener:MaxDuration: The maximum duration of a FlexOffer
- dependencyEnergyConstraintList: Contains the dependency constraints for energy in a Dependency FlexOffer
- uncertainFunctions: (min 1 Function): Contains the parameters which define the uncertain FlexOffer.
- uncertainThreshold: a real number which represents the probability threshold.

A dco:FlexOffer, consists of (dco:ConsistsOf) of at least one dco:FlexOfferTariffCOnstraint, which is associated with the following properties:

- saref4ener:DefaultDuration: saref:Time
- s4ene:hasPrice: sarefPrice

In the following figure is the detailed ontological design of the energy flexibility. The yellow classes extend the purple classes of S4ENER ontology.

A dco:FlexOffer can have a dco:Schedule. A dco:Schedule can have a dco:scheduleType, which is a dco:defaultSchedule or a dco:FlexOfferSchedule. A dco:Schedule have the following properties:

- dco:scheduleType: which can one of the two classes: dco:DefaultSchedule or dco:FlexOfferSchedule.
- scheduleID: the id of a schedule
- updateID: the update id of a schedule
- dco:startTime: the start time of a schedule

A dco:Schedule consist of (dco:consistsOf) at least one slice (dco:Slice). A dco:Slice is a rdfs:subClassOf the class s4ener:Slot. A dco:Slice inherits all the properties of the s4ener:Slot, and extend it with the dco:haPrice object property. This property defines the energy price of a slice.

In the following Figure 6 is the Energy Flexibility design.

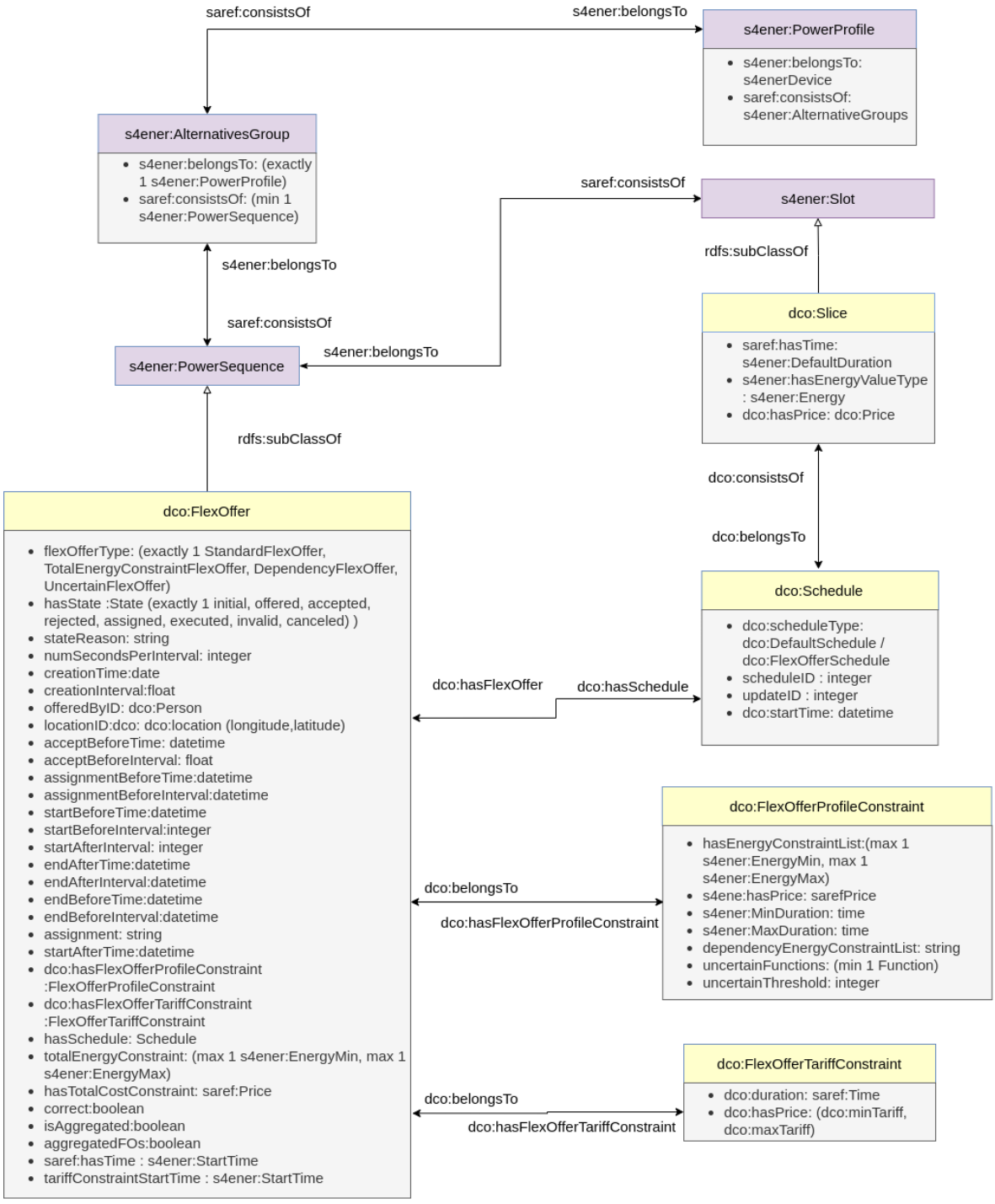


FIGURE 6: ENERGY FLEXIBILITY DESIGN

6. dCO in Action

6.1. TD Annotation

A Thing Description (TD) describes the metadata and interfaces of Things, where a Thing is an abstraction of a physical or virtual entity that provides interactions to and participates in the Web of Things.

- A Thing Description must be annotated using:
- The thing description @Type (e.g., dco:temperatureSensor, dco:space)
- The interaction affordances (e.g., dco:temperaturePropertyAffordance)
- The units of measurements of each interaction affordance (e.g., dco:Celsius)

In the following Figure 7 is an example of an annotated Thing Description.

```
{"@context": [
    "https://www.w3.org/2019/wot/td/v1",
    {"dco": "https://w3id.org/dco"}],
  "title": "the Ambient sensor",
  "id": "5dbff2e6-a788-4cc1-99b8-822cf00f02e2",
  "@type": "dco:AmbientSensor",
  "properties": {
    "temperature01": {
      "@type": "dco:temperaturePropertyAffordance",
      "type": "number",
      "unit": "dco:Celcius",
      "forms": [{"href": "https://ambtsensor.example.eu
        /temperature"}]},
    "humidty": {
      "@type": "dco:HumidtyPropertyAffordance",
      "type": "number",
      "unit": "dco:Percentage",
      "forms": [{"href": "https://ambtsensor.example.eu
        /humidity"}]}},
  "events": {
    "overheating": {
      "@type": "dco:temperatureOverheatingEventAffordance",
      "data": {"type": "string"},
      "forms": [{"href": "https://ambtsensor.example.eu/oh"}]}}
}
```

FIGURE 7: AN ANNOTATED THING DESCRIPTION

6.2. Building Description Example

We consider a building with 2 storeys and a technical room. Each storey has 2 apartments. This building has a space heating service and a set of devices to ensure the service. The Building Description JSON-LD file related to this example could be found here: https://www.dco.domos-project.eu/BDexample.json

This JSON-LD document describes the metadata of a building using the JSON-LD format. JSON-LD (JavaScript Object Notation for Linked Data) is a way to express linked data in JSON format. It uses a set of keywords, such as "@context" and "@type", to indicate relationships between data elements.

The document starts by defining the "@context" which includes a single key "dco" with a value of "https://w3id.org/dco#". This is a namespace prefix that is used throughout the document to indicate that the keys that follow are part of the "dco" namespace.

The document then describes the building using the key "dco:singleFamiliyApartmentsBuilding01", which is of type "dco:SingleFamiliyBuilding". The building has several properties such as a name, envelope type, thermal capacity, construction year, city, zip code, address, renovation year, total space, and a list of storeys. The building also includes a service "dco:spaceHeatingService01" that provides heating to the apartments. The building has two storeys identified by the keys "dco:Storey01" and "dco:Storey02" respectively. The first storey has a space identified by the key "dco:technicalRoom01" which has several devices.

The document also describes the devices that are present in the building, these devices include those located in the technical room of the first storey such as a heat pump, a space heating meter, a heat pump electricity meter, a space heating controller, a water tank, and a water tank temperature sensor. These devices are used to control and monitor the heating service in the building. Additionally, the document also mentions that the building has ambient sensor and heat controller devices located in the living rooms of each of the apartments, those devices will help to control and monitor the temperature in the living rooms of the apartments and make the living spaces more comfortable for the residents.

Overall, this document provides structured data about a building and its properties using JSON-LD format. The use of keys that are part of the "dco" namespace, and the use of "@type" and "@id" keywords make it clear the relationships between the different elements of data.

6.3. FlexOffer Representation Example Using dCO

In this subsection, we describe examples for FlexOffers from a specific use case. We will consider the case of a *Tesla Powerwall* battery. It has a capacity of 14 kWh, maximum charging and discharging power of 5 kW, and round-trip efficiency of 90%. Just for this specific case, for simplicity, we use one hour time units, i.e., the battery can either be charged or discharged by an amount up to 5 kWh at each time unit. The initial state of charge of the battery is 0 kWh, and the battery can only be charged. We will issue FlexOffers for six-time units. Three types of FlexOffers will be issued: Standard, Total energy constraint and Dependency. Each of them is in the format of a JSON message, which contains information about the FlexOffer creator, execution time and energy constraints.

The issued Standard FlexOffer for our use case will appear like in Figure 8. We chose to use an *outer* approximation of flexibility – in other words, the flexibility we are modeling is strictly more than the amount actually available. At each time slice, the bars represent the amount of energy that can be used: here, the minimum is 0 kWh, and the maximum is 5 kWh for each time slice. In the figure, the values e_t represent a possible amount of energy consumption at each time slice. The example for this specific FlexOffer can be found here. In particular, for each time slice, the attribute *energyConstraintList* defines the slice constraints by the attributes *lower* and *upper*, which are 0 kWh and 5 kWh at every time.



FIGURE 8: EXAMPLE OF A STANDARD FLEXOFFER

The Total energy constraint FlexOffer for our use case can be seen in Figure 9. The slice constraints are the same of the Standard FlexOffer represented in Figure 8. In addition to those, there is an additional constraint over the total amount of energy that can be used, written on the top of the figure: in this case, we want the battery to be charged during the considered time horizon by at least 10 kWh, but lower than 14 kWh, which is the total capacity of the battery. The example for this specific FlexOffer can be found here. The slice constraints are represented as showed in the previous example, and the total energy constraint is represented by the *TotalEnergyConstraint* attribute and its *lower* and *upper* subattributes.

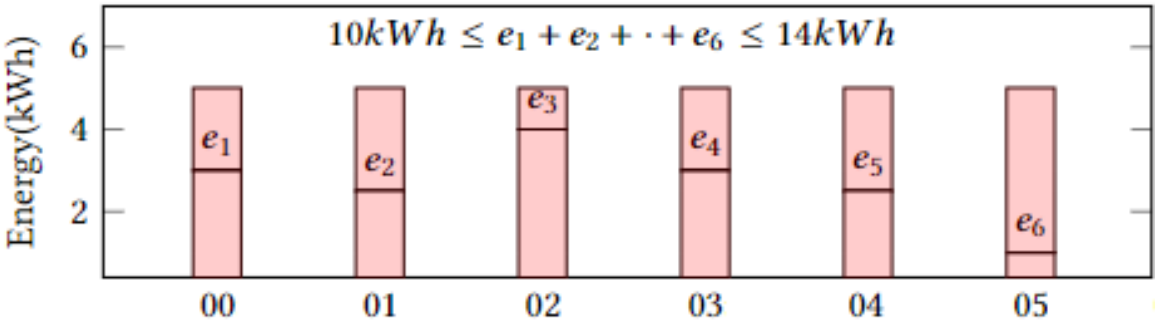


FIGURE 9: EXAMPLE OF TOTAL ENERGY CONSTRAINT FLEXOFFER

We can see the example for the Dependency FlexOffer related to our use case in Figure 10. We only show the first four time slices: from the fifth on, they are exactly as for t = 4. At each time, a polygon in a 2D space is represented: the x axis indicates how much energy has been consumed in total from the device up to the considered time, the y axis indicates how much energy can be consumed at the considered time, and the polygon shows which configurations are possible. For example, in Figure 10, at time t = 3 any amount of energy between 0 kWh and 5 kWh can be consumed, regardless of how much energy has been consumed until then. However, for t = 4 this is no longer true: for example, if the battery has consumed a total of 12 kWh in the first three time slices, the amount of energy that can be consumed now is between 0 kWh and 2 kWh.

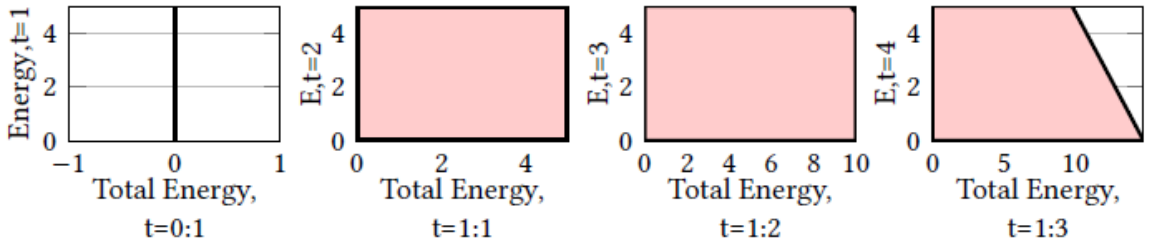


FIGURE 10: EXAMPLE OF DEPENDENCY FLEXOFFER

We will also show a more complex case for Dependency FlexOffers since the constraints from the previous case are effectively the same as total energy constraints. Figure 11 shows a Dependency FlexOffer generated from the same case described at the beginning of this subsection, except the initial state of charge is 7 kWh and the prosumer can decide to switch between charging or discharging the battery at each time. The polygons are defined by sets of equations: for example, at time t = 4, the polygon is the region of space defined by the equations

$$y \geq -5$$

$$y \leq 5$$

$$x \geq -6.64$$

$$x \leq 8.14$$

$$x + y \geq -6.64$$

$$x + y \leq 8.64$$

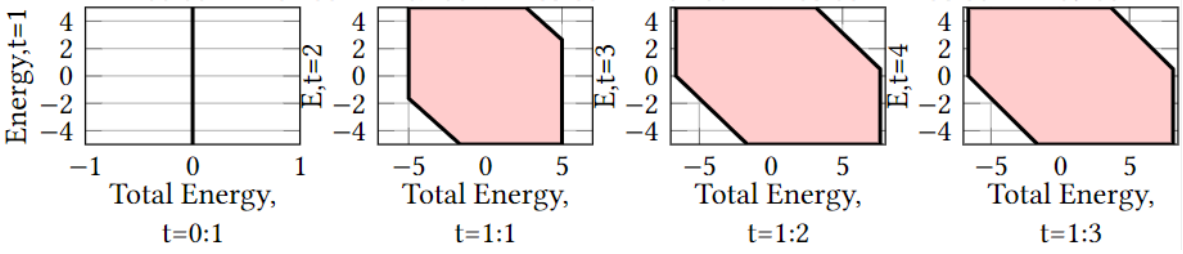


FIGURE 11: EXAMPLE OF DEPENDENCY FLEXOFFER, IN A *SWITCHING* CASE

Here is an example of the representation of a Dependency FlexOffer. The constraints are represented by the attribute *DependencyEnergyConstraintsList*, which contains a matrix, represented as a list of rows. The matrix has as many rows as the number of sides of the polygons, and three columns: if the inequality representing the side is written as ax + by ≤ c, the row will be [a, b, c]. This is called the H-representation of the slice (Siksnys, et al., 2016).

The FlexOffer examples can be found at the following links for Standard, Total energy constraint and Dependency FlexOffers respectively. We have represented using the energy flexibility module of dCO the three above examples of FlexOffers.

The turtle representation of the FlexOffers can be found at the following links for Standard, Total energy constraint and Dependency FlexOffers respectively.

In turtle file of the Standard FlexOffer, we represent a FlexOffer name dco:FO-001, which is an instance of the class dco:FlexOffer. This FlexOffer has the type (dco:flexOfferType) as a Standard FlexOffer (dco:StandardFlexOffer).

This standard FlexOffer is associated with the following properties:

- dco:hasState: State of the FO (initial/offered/ accepted/rejected/assigned/executed/ invalid/ canceled)
- dco:stateReason: Reason for FO rejection (if State == rejected)
- dco:locationID: the location of the FO in the grid system
- dco:acceptBeforeInterval: Interval before which FO must be accepted
- dco:assignmentBeforeInterval: Interval before which FO must be scheduled
- dco:startAfterInterval: Interval after which FO must be started
- dco:startBeforeInterval: Interval before which FO must be started
- dco:endAfterInterval: Interval after which FO execution must end
- dco:endBeforeInterval: Interval before which FO execution must end
- dco:acceptanceBeforeTime: Absolute time before which FO must be accepted
- dco:assignmentBeforeTime: Absolute time before which FO must be scheduled
- dco:numSecondsPerInterval: Duration (in seconds) of a time interval. The default value is 900
- dco:startAfterTime: Absolute time after which FO must be started
- dco:startBeforeTime: Absolute time before which FO must be started
- dco:endAfterTime: Absolute time after which FO execution must end
- dco:endBeforeTime: Absolute time before which FO execution must end
- dco:creationTime: Absolute time at which the FO has been created
- dco:tariffConstraintStartTime: The start time of the tariff constraint

Also, this FlexOffer has (dco:hasSchedule) two Schedules: dco:defaultSchedule01, dco:FlexOfferSchedule01, which represent respectively the default schedule and the FlexOffer schedule. These two schedules are instances of the class dco:Schedule. In the following are the properties associated to a dco:Schedule:

- rdf:type: Can be either a dco:DefaultSchedule or a dco:FlexOfferSchedule
- dco:hasStartTime: the start time of the Schedule
- dco:scheduleID: the Schedule ID of the schedule
- dco:updateID: the update ID of the Schedule
- dco:consistsOf: the list of dco:slice that a schedule has.

In the Standard FlexOffer example, the default schedule has 6 slices (I.e., dco:Slice-1-DS0001, dco:Slice-2-DS0001). A dco:slice is associated to the following:

- s4ener:DefaultDuration: the duration of a dco:Slice.
- s4ener:hasEnergyValueType: the range of this property is either s4ener:Energy or 4ener:Power. In our example it is the energy, in which mention the energy value.
- s4ener:hasPrice: the price of a kwh of a dco:Slice

Moreover, in this example, a dco:FlexOffer is associated (dco:hasFlexOfferProfileConstraint) with 6 dco:FlexOfferProfileConstraint (I.e., dco:FlexOfferProfileConstraint01, dco:FlexOfferProfileConstraint02). Each of Flex Offer profile constraints is associated to the following properties: dco:FlexOfferProfileConstraint, dco:hasEnergyConstraintList, s4ener:hasPrice, s4ener:MinDuration, and s4ener:MaxDuration.

The total energy constraint Flex Offer follows the same representation as the Standard Flex Offer, except an additional property: dco:totalEnergyConstraint, which indicates the maximum total energy of a FlexOffer.

The Dependency Flex Offer extends the class dco:FlexOfferProfileConstraint with the property dco:dependencyEnergyConstraintList, which include a matrix of constraints.

7. Conclusion

This document presents the final version of dCO ontology. We first presented the used methodology for dCO ontology development. We built and described the final version on dCO ontology. This ontology includes the following 6 modules: Thing description module, Devices module, dCO core module, Units of measurement module, Building structure module, power, electricity and heating module and energy flexibility module. We finished the deliverable by presenting the use of dCO in the semantic annotation of a Thing Description, the elaboration of a Building Description and three examples of FlexOffers representation using dCO.

8. References

Charpenay V. and Käbisch S. On modeling the physical world as a collection of things: The w3c thing description ontology. [Conference] // European Semantic Web Conference. - 2020.

Garijo D. and Poveda-Villalón M. Best Practices for Implementing FAIR Vocabularies and Ontologies on the Web [Journal].

Pedersen T.B., Siksnys L. and Neupane B. Modeling and managing energy flexibility using FlexOffers [Journal]. - 2018.

Rasmussen M.H., et al. BOT: the building topology ontology of the W3C linked building data group [Conference] // Semantic Web 12.1. - 2021.

Rijgersberg H., Van Assem M. and Top J. Ontology of units of measure and related concepts [Journal].

Siksnys L. and Pedersen T.B. Dependency-based FlexOffers: scalable management of flexible loads with dependencies [Journal] // ACM e-Energy. - 2016.

Suárez-Figueroa M.C., Gómez-Pérez A. and Fernández-López M. The NeOn methodology for ontology engineering [Journal] // Ontology engineering in a networked world. - 2012.

Wilkinson M.D., et al. The FAIR Guiding Principles for scientific data management and stewardship [Journal] // Scientific data. - 2016.