COGITO

CONSTRUCTION PHASE
DIGITAL TWIN MODEL

cogito-project.eu

D3.4 – COGITO Data Model and Ontology Definition and Interoperability Design v3

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 958310

D3.4 – COGITO Data Model and Ontology Definition and Interoperability Design v3

Dissemination Level:	Public
Deliverable Type:	Report
Lead Partner:	UPM
Contributing Partners:	Hypertech, UCL, DTU, UEDIN, CERTH, BOC-AG, QUE, NT
Due date:	31-10-2022
Actual submission date:	27-10-2022

Authors

Name	Beneficiary	Email
Socorro Bernardos	UPM	sbernardos@fi.upm.es
María Poveda-Villalón	UPM	mpoveda@fi.upm.es
Raúl García-Castro	UPM	rgarcia@fi.upm.es
Giorgos Giannakis	Hypertech	g.giannakis@hypertech.gr
Christos Tsalis	Hypertech	c.tsalis@hypertech.gr
Theofania Charbi	Hypertech	t.charbi@hypertch.gr
Evangelos Karalis	Hypertech	e.karalis@hypertech.gr
Georgios Lilis	UCL	g.lilis@ucl.ac.uk
Jochen Teizer	DTU	teizerj@byg.dtu.dk
Frédéric Bosché	UEDIN	f.bosche@ed.ac.uk
Vasilios Karkanis	CERTH	vkarkanis@iti.gr
Damiano Falcioni	BOC AG	damiano.falcioni@boc-eu.com
Panagiotis Moraitis	QUE	p.moraitis@que-tech.com
Ján Varga	NT	varga@novitechgroup.sk

Reviewers

Name	Beneficiary	Email
Apostolos Papafragkakis	Hypertech	a.papafragkakis@hypertech.gr
Bohuš Belej	NT	belej@novitechgroup.sk

Version History

Version	Editors	Date	Comment
0.1	UPM	21.09.2022	Update of executive summary, introduction and conclusions
0.2	UPM	05.10.2022	Update of requirements and ontology
0.3	UPM	10.10-2022	Updates of diagrams and ontology
0.4	UPM	18.10-2022	Version for internal review
0.5	Hypertech, NT	25.10.2022	Internal review
0.6	UPM	27.10.2022	Review comments addressed
0.7	UPM	27.10.2022	Final version
1.0	UPM, Hypertech	27.10.2022	Submission to the EC portal

Disclaimer

Executive Summary

The COGITO Digital Twin platform aims to enable interoperability with existing and emerging standards and data models covering different domains. Semantic interoperability in the COGITO platform is based on ontologies that will be exploited throughout the COGITO infrastructure and used to semantically represent the information exchanged.

This document details the methodology and technological infrastructure used to develop the COGITO ontology network, as well as the third version of the ontologies that make up such an ontology network.

The followed methodology is called Linked Open Terms [1] and includes four activities: 1) ontology requirements specification, 2) ontology implementation, 3) ontology publication, and 3) ontology maintenance.

The COGITO ontology network consists of seven modules corresponding to the facility (the construction itself), the construction process, the construction resources, the IoT devices used to locate resources, the quality achieved, the safety information related to the facility, and the entities managed by the digital twin platform (digital twins and information resources). These modules have already been conceptualised, implemented, and published and will be modified and complemented with new modules as the project progresses.

During the maintenance phase of the ontology development process, which will start after this deliverable is submitted, we expect future change requests in the ontologies due to defects found or to new requirements that may appear during integration. The COGITO ontology portal will serve as a living repository for the different ontologies and will contain the latest version of the developed ontologies and all related artifacts.

Note: Since the methodology and infrastructure followed are the same as those described in in the previous versions of the COGITO Data Model and Ontology Definition and Interoperability Design (See D3.2 [2] and D3.3 [3]), this deliverable differs from them mainly in the sections that describe new developments in the ontology, namely Sections 4 and 5. In addition, in D3.2 [2] a coverage analysis was carried out to identify to what extent some of the terms that appeared in the requirements also appeared in the ontologies described in the previous D3.1 deliverable [4]. The conclusions highlighted the fact that reusing ontologies requires analysing multiple factors beyond coverage, and some of them are not easily quantifiable or even cannot be measured objectively. That is why we have not performed the coverage analysis for the new ontology requirements and removed that section.

Table of Contents

List of Figures

List of Tables

List of Acronyms and Abbreviations

Term	Description
BBO	BPMN Based Ontology
BEO	Building Element Ontology
BOT	Building Topology Ontology
BPMN	Business Process Modelling Notation
BPO	Building Product Ontology
DICO	Digital Construction
ETSI	European Telecommunications Standards Institute
IoT	Internet of Things
LOT	Linked Open Terms
ORSD	Ontology Requirements Specification Document
OWL	Web Ontology Language
PMS	Process Modelling and Simulation
SAREF	Smart Applications REFerence ontology
SAREF4BLDG	SAREF extension for the building domain
SSN	Semantic Sensor Network
SOSA	Sensor, Observation, Sample and Actuator
WODM	Work Order Definition and Monitoring tool
WoT	Web of Things

1 Introduction

1.1 Scope and Objectives of the Deliverable

The COGITO Digital Twin platform aims to enable interoperability with existing and emerging standards and data models covering different domains. Semantic interoperability in the COGITO platform is based on ontologies that will be exploited throughout the COGITO infrastructure and used to semantically represent the information exchanged.

In computer science, ontologies are defined as "formal, explicit specifications of a shared conceptualization". The COGITO ontologies have been developed using the W3C Web Ontology Language standard (OWL)[1], reusing existing resources and standards whenever possible. This development is based on the technical specification of the COGITO architecture and is based on a set of requirements extracted from its different components and use cases.

This deliverable describes the environment that supports the development of the COGITO ontologies. To do so, on the one hand, it describes the methodology followed to develop the ontologies and the ontology development infrastructure deployed to support such development. On the other hand, it presents the results of the different ontology development sprints. Finally, this document presents an overview of the current version of the COGITO ontologies that are available online on the COGITO ontology portal.[2]

The version of the ontology presented in this document is the version produced on the date of writing of this document. In any case, the COGITO ontology portal shall always contain the latest version of the ontologies and all related artifacts for ontology development.

1.2 Relation to other Tasks and Deliverables

This deliverable is based on the use cases defined in D2.1-Stakeholder Requirements for the COGITO System [5] and on the COGITO architecture defined in D2.4-COGITO System Architecture [6]. Furthermore, D3.1-Survey of Existing Models, Ontologies and Associated Standardization Efforts [4] was also considered to identify existing ontologies to reuse in the COGITO ontologies.

The COGITO ontology network described here will allow for sharing and interoperability among the different components of the COGITO architecture, especially in the communications with the Digital Twin platform. Because of this, this document has been so important when developing each component of the COGITO architecture, that is, the results of work packages 4, 5, 6, and 7 and their integration in work package 8.

1.3 Updates from the Previous Version

This deliverable is the third one devoted to the development of the COGITO ontologies. As in the previous version of the deliverable (v2), the sections related to the ontology development methodology and the development infrastructure are identical (sections 3 and 6, respectively) and the main changes compared to the previous versions of the deliverable are the following.

- The introduction and conclusions of the document have been updated.
- Section 3 has been updated, including the new requirements gathered for the COGITO ontology since the previous version of the deliverable.
- Section 4 has been updated to include the current description of the ontology that includes the new requirements.
- The section related to the coverage analysis in D3.2-COGITO Data Model and Ontology Definition and Interoperability Design v1 [2] has been removed from the next versions of the deliverable. The conclusions that were extracted from such analysis highlighted the fact that reusing ontologies requires analysing multiple factors beyond coverage, and some of them are not easily quantifiable or even cannot be measured objectively. This, and the high manual effort required to perform the

[1] https://www.w3.org/TR/owl2-primer/
[2] https://cogito.iot.linkeddata.es/

coverage analysis, are the reasons of not performing the coverage analysis for the new ontology requirements that have been gathered.

1.4 Structure of the Deliverable

- **Section 2** provides an overview of the methodology used to develop the COGITO ontologies;
- **Section 3** presents the requirements that should be satisfied by the COGITO ontologies;
- **Section 4** provides a description of the COGITO ontologies;
- **Section 5** presents the deployed ontology development infrastructure; and
- **Section 6** provides some conclusions and insights into future work.

2 Ontology Development Methodology

This section presents the ontology development methodology used for the development of the COGITO ontology network. This methodology includes four activities: 1) ontology requirements specification, 2) ontology implementation, 3) ontology publication, and 3) ontology maintenance. This development methodology is called Linked Open Terms (LOT) [1] and is based on the NeOn methodology [7]. It has been used and refined in previous ontology development processes from the European projects VICINITY [8], BIMERR [9] and DELTA [10], and adapted to the particularities of COGITO (see Sections 3 and 4).

Figure 1 shows an overview of the activities that are performed and the artefacts that result from them: the ontology requirements specification document (ORSD), the ontology implementation, the ontology available online, issues, bugs, etc. The following subsections detail each activity (and product). In the figures included in this section, these artefacts are represented in green squares and assigned a number; the different activities in the methodology enumerate in a green circle the numbers of all the input artefacts that are used in the activity.

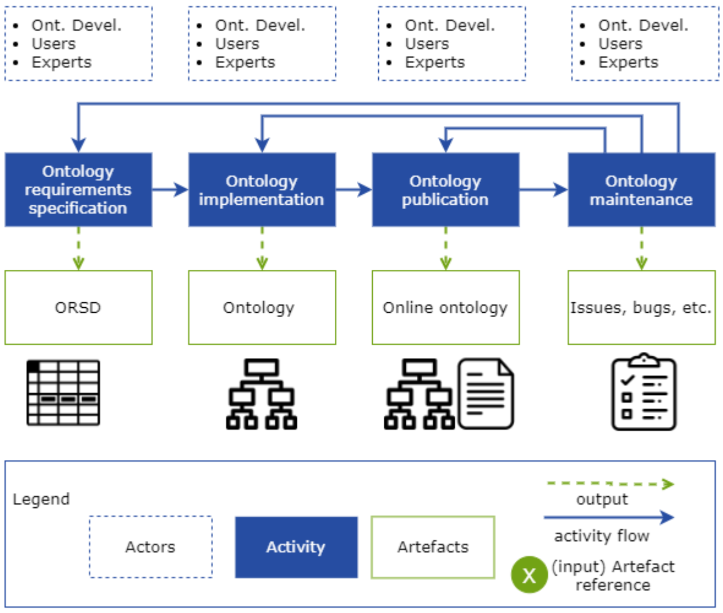
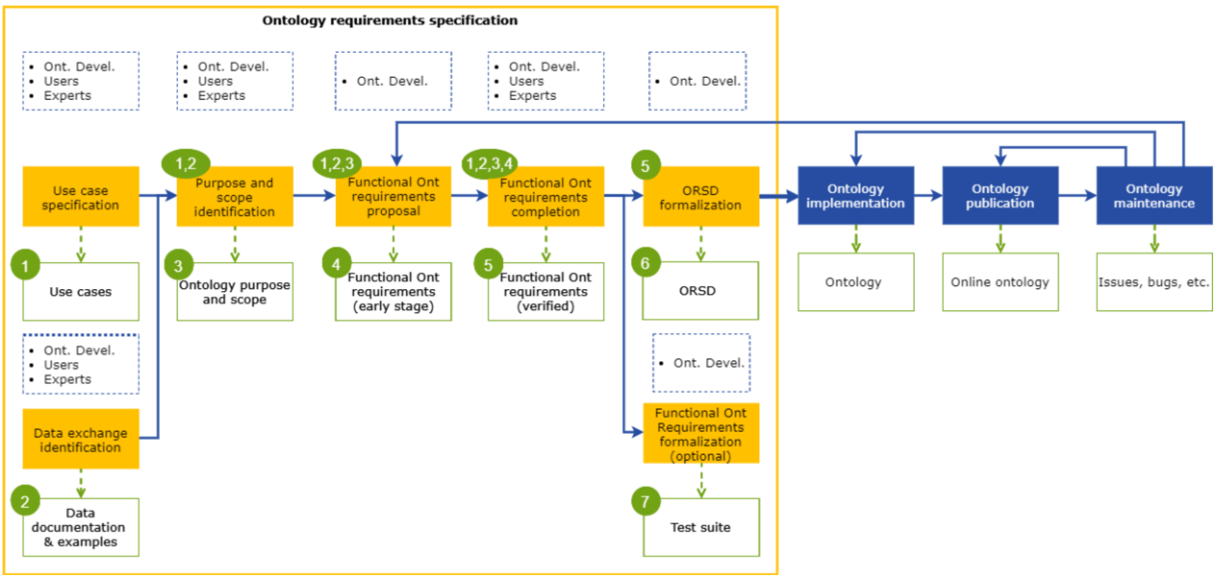


Figure 1 – Ontology development methodology followed in COGITO

2.1 Ontological Requirements Specification

The aim of the requirements specification activity is to identify and define the requirements the ontology to be created needs to fulfil. During this first activity, it is necessary to involve experts in the domain to generate the appropriate industry perspective and knowledge. Figure 2 shows the workflow for the ontology requirements specification activity.


Ontology requirements specification
Ont. Devel.
Users
Experts
Use case specification
Purpose and scope identification
Functional Ont requirements proposal
Functional Ont requirements completion
ORSD formalization
Ontology implementation
Ontology publication
Ontology maintenance
1,2
1,2,3
1,2,3,4
1 Use cases
3 Ontology purpose and scope
4 Functional Ont requirements (early stage)
5 Functional Ont requirements (verified)
6 ORSD
Ontology
Online ontology
Issues, bugs, etc.
Data exchange identification
2 Data documentation & examples
Functional Ont Requirements formalization (optional)
7 Test suite
Figure 2 – Workflow for ontology requirements specification

2.1.1 Identification of Purpose and Scope

The goal of this step is to define the purpose and scope of the given ontology or module. To this end, the ontology development team works in collaboration with users and domain experts to define the purpose and scope of each ontology or module to be developed.

2.1.2 Data Exchange Identification and Use Case Specification

During this step, the ontology development team needs to gather the necessary documentation about the domain to be modelled. This documentation might correspond to the following.

- Standards
- Datasets
- API specifications
- Use cases

This documentation needs to be collected by both the ontology development team and domain experts.

2.1.3 Ontological Requirement Proposal

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

The format used for this requirement proposal follows a tabular approach and includes the following fields:

- Requirement identifier, which must be unique for each requirement.
- Partner who proposed the requirement.
- Component of the COGITO architecture from which the requirement was extracted.
- Sprint in which the requirement is planned to be implemented.
- Competency question or assertion.
- Status of the requirement, which can be: (1) Proposed, (2) Accepted, (3) Rejected, (4) Pending, or (5) Deprecated.
- In case the requirement is deprecated, the identifier of the updated requirement is used.
- Comments on the requirement.
- Provenance of the requirement, e.g., use case or standard.
- Priority of the requirement, which can be: (1) High, (2) Medium, or (3) Low.

The ontological requirements are completed iteratively, as the following ontology development process is incremental.

2.1.4 Completion and Validation of Ontology Requirements

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

2.1.5 Prioritization of Ontological Requirements

Prioritization of requirements allows development teams to schedule the development of the ontology in sprints. In the COGITO project, this prioritization will be performed if there is a need to prioritise functional requirements, mainly for those sprints with a high number of requirements in which the implementation of some of the requirements has to be postponed for a future sprint.

If this prioritisation is needed, to carry it out, the ontology development team works with the domain experts to identify which requirements need to be fulfilled first.

2.2 Ontology Implementation

During the implementation activity, the ontology is built using a formal language based on the requirements identified in the previous activity.

Taking the set of requirements collected in the previous activity as input, the ontology implementation activity is carried out through several sprints. To this end, the ontology developers schedule the ontology development according to the requirements that were identified, and the ontology development team builds the ontology iteratively by implementing only a certain number of requirements in each iteration. The output of each iteration is a new version of the ontology implementation. Figure 3 shows the steps that are followed in this ontology implementation activity.

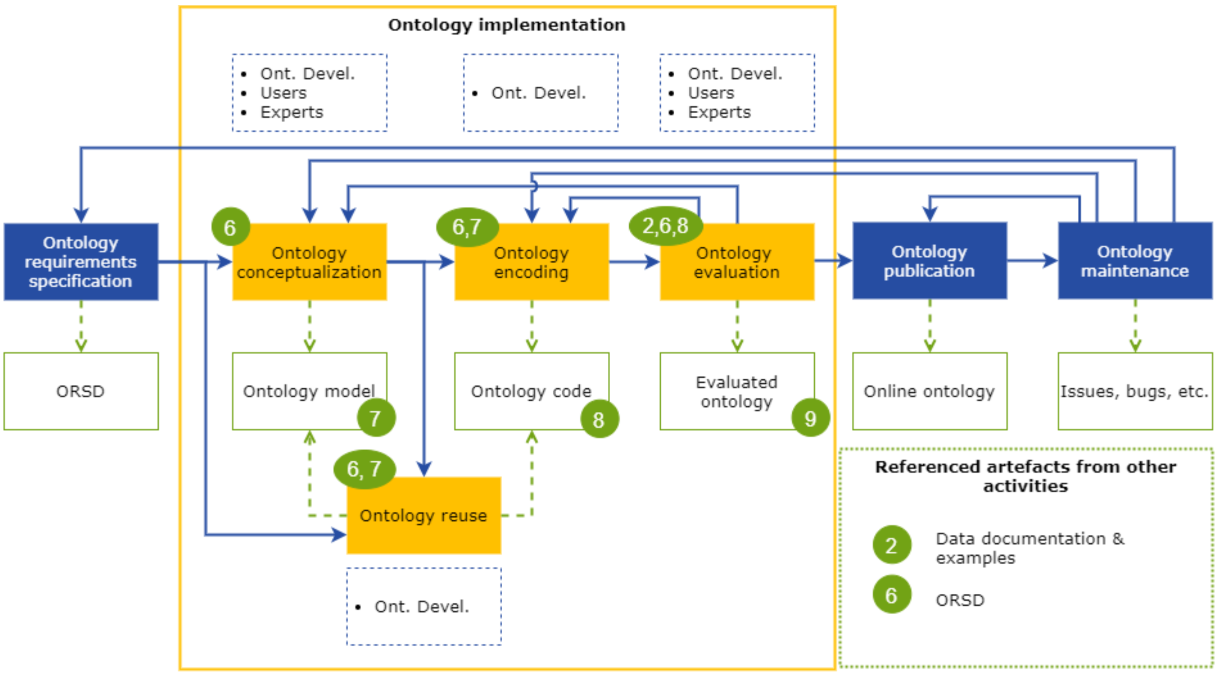


Figure 3 – Workflow for the ontology implementation activity

2.2.1 Ontology Conceptualization

The aim of this activity is to build an ontology model from the ontological requirements identified in the requirements specification process that represents the domain of the ontology. Therefore, during the conceptualization of the ontology, the domain knowledge obtained from the previous activity is organized and structured into a model by the ontology development team.

2.2.2 Ontology Encoding

The purpose of the encoding activity is to implement the ontology in an implementation language such as OWL. The ontology code resulting from this activity includes, in addition to the ontology classes, properties, and axioms. Furthermore, following the FAIR principles [11], the ontology code also includes ontology metadata, such as the creator, title, publisher, license, and version of the ontology, in addition to the metadata for each of the ontology.

To manage the ontology versions developed in the COGITO project, the following version convention was adopted. Following this convention, each release will follow the pattern v.major.minor.fix, where each field follows the rules:

- **major**: The field is updated when the ontology covers the entire domain that it intends to model. That is, it is a complete product and covers the final goal of the development.
- **minor**: The field is updated when:
  - All the requirements of a subdomain are covered.
  - Documentation is added to the ontology.
- **fix**: The field is updated when:
  - Typos or bugs are corrected in the ontology.
  - Classes, relationships, axioms, individuals, or annotations are added, deleted, or modified, but the domain is not covered.

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

2.2.3 Ontology Evaluation

Once the ontology is encoded, it should be evaluated before its online publication. The development of the ontology must guarantee the following aspects:

- The ontology is consistent.
- The ontology does not have syntactic, modelling, or semantic errors.
- The ontology fulfils the requirements scheduled for the ontology, to ensure that the ontology is completed regarding the domain experts needs.

2.3 Ontology Publication

During the ontology publication activity, the ontology development team provides an online ontology that is accessible both as a human-readable document and as a machine-readable file from its URI. Figure 4 shows an overview of the steps performed during the ontology publication activity.

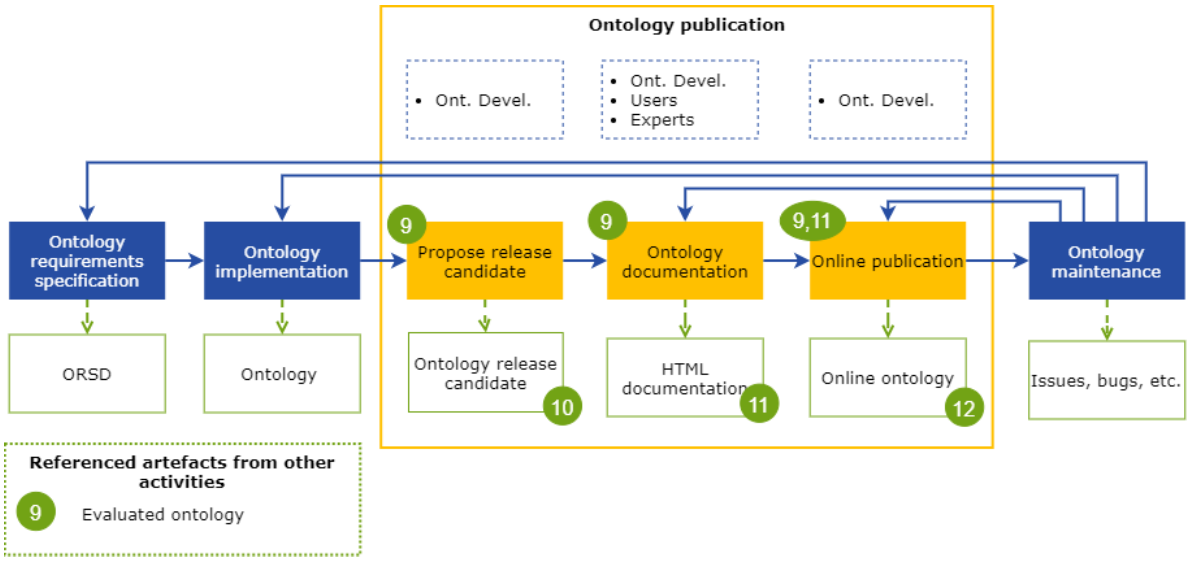


Figure 4 – Workflow for the ontology publication activity

2.3.1 Propose Release Candidate

Once the ontology developers have implemented and evaluated the ontology, the next task is to decide whether the current version is going to be published on the Web (or shared among other project partners, for example, software developers making use of the ontology) or whether it is needed to document such version of the ontology for any other reason (for example, a set of requirements is fulfilled and triggers a new release). In this case, the version at hand becomes a release candidate. In case the ontology is not selected as a release, the ontology development team generates a pre-release version of the ontology. Both the release and the pre-release versions of the ontologies are evaluated and ready to be used.

2.3.2 Ontology Documentation

Taking the ontology generated in the previous activities as input, the ontology development team generates the ontology documentation. This documentation includes the following:

- An HTML description of the ontology that describes the classes, object properties, and data properties of the ontology, the license URI, and title being used. Domain experts must collaborate with the ontology development team to describe classes and properties. This description also includes metadata such as creator, publisher, date of creation, last modification, or version number. It also includes links to different formats for serialization of the ontology, such as TTL, JSON-LD, or RDF/XML.
- Diagrams that store the graphical representation of the ontology, including taxonomy and class diagrams.

2.3.3 Online Publication

During this activity, the ontology, which should already be validated and documented, is published on the Web. This ontology is accessible through its URI as a machine-readable and human-readable file using content negotiation.

2.4 Ontology Maintenance

During this activity, the ontology is updated, and new requirements can be proposed to be added to the ontology. Furthermore, during this activity, the ontology development team, together with domain experts and users, can identify and correct errors in the ontology. Figure 5 shows the steps to follow in the maintenance activity of the ontology.

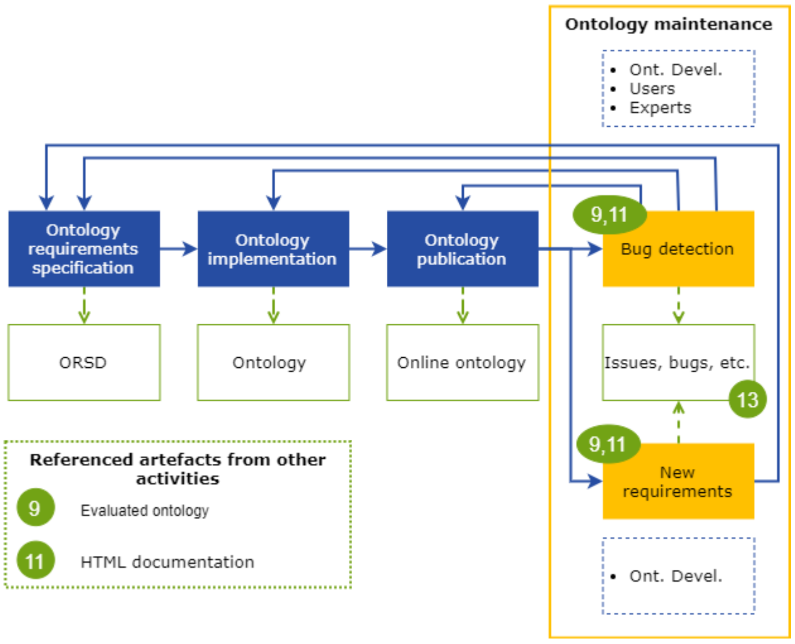


Figure 5 – Workflow for the ontology maintenance activity

2.4.1 Bug Detection

Once the ontology developers have published the ontology, any user, developer, or domain expert can detect and inform about bugs. This notification should be done using an issue tracker, allowing all the information related to the bug, as well as the actor that identifies it, to be stored.

2.4.2 New Requirements Proposal

During this activity, new requirements can be proposed for the ontology. Ontology developers, domain experts, or users can propose modifications to improve the published ontology version. This proposal should be done through an issue tracker so that all the information related to it is stored.

3 Ontology Requirements Specification

Since the goal of the ontology is to facilitate the sharing and interoperability of data between COGITO components through the Digital Twin Platform (see Table 1), we decided to analyse each component of the architecture (defined in D2.4 [6]) in order to gather information about the data it needs as input from other components and the data it produces as output to other components. We also considered the information provided to and by the digital twin platform described in the different use cases.

Several meetings with the partners involved in the components and each use case have helped us specify requirements that are aligned with the needs of those components. These requirements have been organised into seven different domains related to the facility, the construction process, the resources used in that process, the Internet of Things information processed by the platform, the quality of the resulting facility, the safety information related to it, and the entities (digital twins and information resources) managed by the digital twin platform. Table 2 presents the list of requirements for the domains we have identified.

Table 1 – General information on the specification of ontology requirements

Purpose	To facilitate data sharing and interoperability among the COGITO components through the Digital Twin Platform
Scope	Limited to the data shared among the COGITO components through the Digital Twin Platform
Implementation language	Web Ontology Language (OWL)
Intended end-users	COGITO components and application developers COGITO end users and stakeholders
Non-functional ontology requirements	Annotated in English Linked to standards when possible Open license Available online
Functional ontology requirements	Detailed information on the COGITO ontology portal

Table 2 – List of requirements per domain

Identifier (domain+id)	Competency Question / Fact
FACI-1	A project is related to a site
FACI-2	Spatial zones (especially zones of type site) can have facilities
FACI-3	Facilities include buildings, railways, bridges, and roads
FACI-4	Spatial zones (especially facilities and facility parts) can have facility parts
FACI-5	Spatial zones (especially buildings) can have storeys
FACI-6	Spatial zones can have spaces
FACI-7	Spatial zones can contain elements and other spatial zones
FACI-8	Elements include walls, slabs, columns, etc. (building elements)
FACI-9	An element can have other elements
FACI-10	Spatial zones include sites, facilities, facility parts, storeys and spaces
FACI-11	Spatial zones include tracked zones (used by tags) and construction zones (which can be walkable or fall zones)
FACI-12	A spatial zone has an identifier, a name, and a description
FACI-13	An element has an identifier, a name and the material used to build it

FACI-14	A project has an identifier and a name
PROC-1	A project is related to one or more processes
PROC-2	A process can reflect the as-planned original process, the as-planned enriched process and the as-is process
PROC-3	A project includes data about its identifier, name, author, creation date, planned start and end dates, actual start and end dates, time step and time window
PROC-4	A process has a cost, which is measured in a certain currency
PROC-5	A task has a cost, which is measured in a certain currency
PROC-6	A task belongs to a certain process
PROC-7	A task can add, remove, control, or repair elements
PROC-8	A task includes result, progress, and priority information
PROC-9	A task can have a date of creation, of planned and actual start- and planned and actual end-datetime
PROC-10	A task can have sub-tasks
PROC-11	A task can have predecessor tasks
PROC-12	A task requires certain resource types
PROC-13	A task needs a certain quantity of a resource type
PROC-14	There can be geometric quality tasks, visual quality tasks, safety tasks and normal construction tasks
PROC-15	A work order is related to a process and includes certain tasks
PROC-16	A task can have resources assigned
PROC-17	A work order has a human worker as the main provider
RESO-1	Human workers and equipment are resources
RESO-2	A resource belongs to a resource type
RESO-3	Types of resources include human roles and equipment types
RESO-4	A type of resource has an identifier, a name, the maximum quantity and a cost per hour
RESO-5	A resource has an identifier and a status
RESO-6	Human workers can have an email, and pieces of equipment can have a name
RESO-7	Resources can have a tracking tag assigned
RESO-8	A resource can have a tracking tag
IOT-1	A tracking tag signals its location (and hence that of the resource to which it is assigned)
IOT-2	A tracking tag signals its location along with the corresponding timestamp
IOT-3	There are three kinds of location a tracking tag can signal: location locationKF and locationMA
QUAL-1	A project can have information resources (especially point cloud and image jobs)
QUAL-2	An element can have information resources (especially as-built point clouds and meshes)
QUAL-3	An element can have results about its quality
QUAL-4	Quality results include defects and geometric quality information
QUAL-5	A quality result can be detected, confirmed or rejected (status)
QUAL-6	A defect is reflected by an image job
QUAL-7	A defect has a predicted label, a confidence level, and a severity indicator

QUAL-8	An image is taken at a time, using a device with a certain position and orientation and is associated to an element
QUAL-9	An image can be processed before and after being analysed
QUAL-10	Geometric quality information is obtained from a point cloud
QUAL-11	A piece of geometric quality information is related to a rule and includes a result (passed or not passed), a tolerance reference, a scalar result, a unit, a scheduled timestamp, a performed timestamp and an identifier
QUAL-12	A rule includes an identifier, an origin document, a label, a description, and keywords related to element, material, and relationship types.
SAFE-1	A construction zone can be related to safety information
SAFE-2	Safety information includes an identifier, a name, a severity indicator, and mitigation data
PLAT-1	Projects and, elements, are digital twins managed by the platform
PLAT-2	Images, point clouds, and meshes are information resources managed by the platform
PLAT-3	A platform entity (digital twin or information resource) can have forms referring to a knowledge graph, a value in the graph, and a file storage
PLAT-4	A platform entity (digital twin or information resource) can have links referring to an IFC file, a schedule file, a point cloud file, a mesh file and a description of another platform entity

4 Overview of the COGITO Ontology Network

The ontology network developed for the COGITO project consists of two parts. One part contains six ontology modules, where each module corresponds to one specific domain, namely facility[3], process, resource, IoT, quality, and safety; and the other part is a module for the entities that the digital twin platform manages (digital twins and information resources), Figure 6 provides a graphical overview of the COGITO ontology network showing the main concepts defined in each module. To understand this figure and the ones provided in the next subsections, in which each module is explained, it is important to consider the following paragraphs.

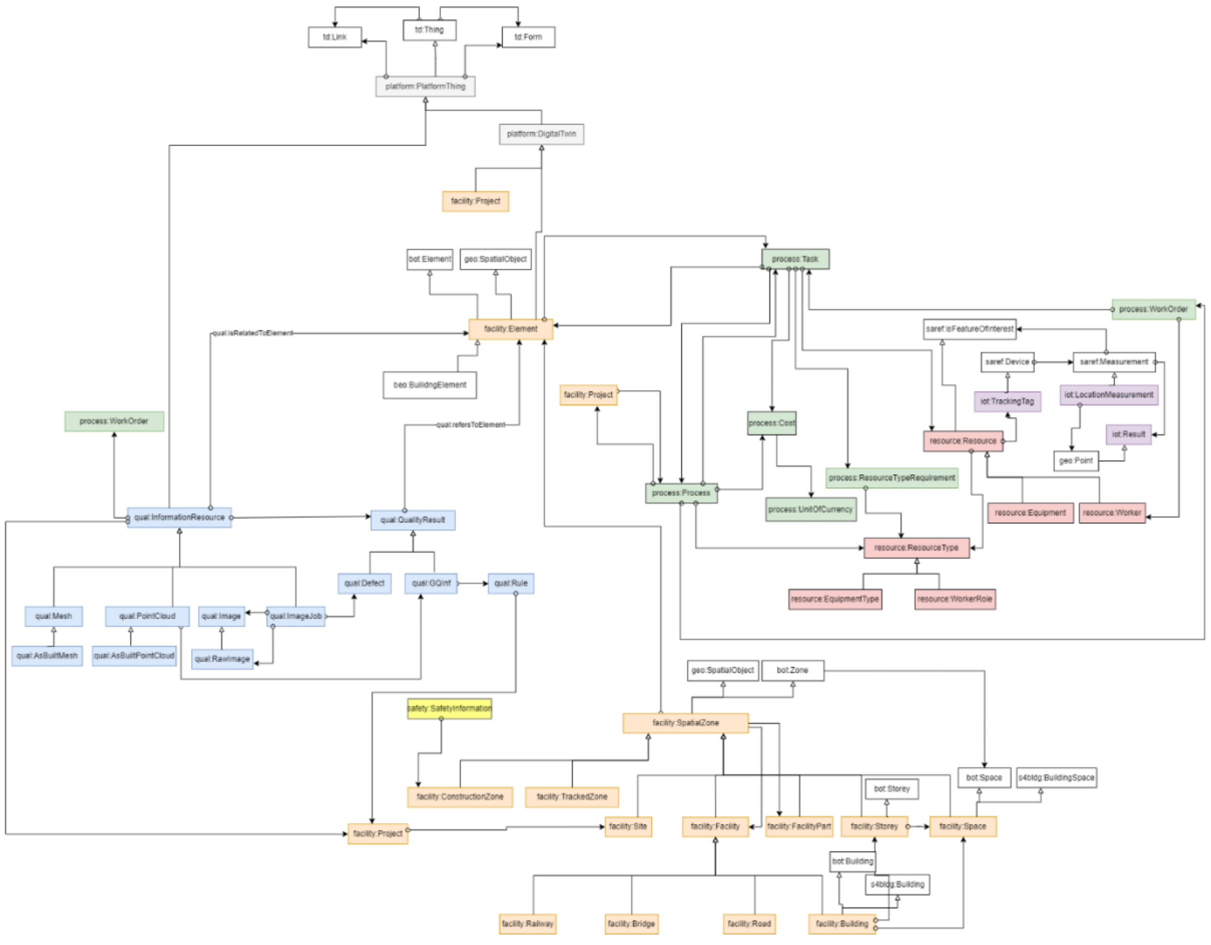


Figure 6 – Overview of the COGITO ontology network

Coloured boxes are used to represent modules currently implemented, while white boxes represent reused terms from existing ontologies. As can be seen, some of the ontologies reviewed in previous steps of the project were reused according to the results of the process described in Section 4, such as BOT and SAREF. Since the COGITO ontology only needs some IFC concepts in order to achieve interoperability, and ifcOWL is a very complex ontology, we decided to first define the main concepts of the COGITO ontology and then analyse their alignment to the IfcOWL ones (see Annex A). It should be noted that the ifcOWL version we have used is the official one, which corresponds to IFC 4.0.2.1. Some concepts relevant to COGITO, such as facility, are inspired by the IFC 4.3.0.0 ones, but have not been implemented yet in ifcOWL.

Additional cross-domain ontologies, such as geo and dcterms, were also reused once requirements and conceptualizations were analysed in detail. The prefixes, and corresponding ontologies, created and reused in the COGITO ontology are listed in Table 3 and Table 4, respectively.

[3] Note that this module was called "construction" in D3.2

Table 3 – List of the ontologies created in the COGITO project

Prefix	Namespace
facility	https://cogito.iot.linkeddata.es/def/facility#
iot	https://cogito.iot.linkeddata.es/def/iot#
platform	https://cogito.iot.linkeddata.es/def/platform#
process	https://cogito.iot.linkeddata.es/def/process#
qual	https://cogito.iot.linkeddata.es/def/quality#
resource	https://cogito.iot.linkeddata.es/def/resource#
safety	https://cogito.iot.linkeddata.es/def/safety#

Table 4 – List of the ontologies reused in the COGITO project

Prefix	Namespace
beo	https://pi.pauwel.be/voc/buildingelement#
bot	https://w3id.org/bot#
dcterms	http://purl.org/dc/terms/
geo	http://www.opengis.net/ont/geosparql#
ifc	https://standards.buildingsmart.org/IFC/DEV/IFC4/ADD2_TC1/OWL#
rdfs	http://www.w3.org/TR/rdf-schema/
s4bldg	https://saref.etsi.org/saref4bldg/
td	https://www.w3.org/2019/wot/td#

The main hierarchies between concepts are also included and ad hoc relations between different modules and within modules are also present. Arrows are used to represent these properties between classes and to represent some RDF, RDFS, and OWL constructs. More precisely:

- Plain arrows with white triangles represent the subclass relationship between two classes. The origin of the arrows is the class to be declared as a subclass of the class at the destination of the arrow.
- Plain arrows between two classes indicate that the object property has declared as domain the class in the origin, and as range the class in the destination of the arrow. The identifier of the object property is indicated in the arrow.
- Dashed labelled arrows between two classes indicate that the object property can be instantiated between the classes in the origin and the destination of the arrow. The identifier of the object property is indicated in the arrow.
- Dashed arrows with the identifiers between stereotype signs (i.e., "<< >>") refer to OWL constructs that are applied to some ontology elements, that is, they can be applied to classes or properties depending on the OWL construct being used.
- Dashed arrows without identifiers are used to represent the rdf:type relation, indicating that the element in the origin is an instance of the class in the destination of the arrow.

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

For more details about the graphical notation used in this diagram, you can check the Chowlk Visual Notation[4].

The complete and up-to-date documentation of each ontology module is provided online, as explained in Section 3. In the rest of this section, only the main concepts and modelling decisions are detailed.

[4] https://chowlk.linkeddata.es/chowlk_spec

4.1 Facility Module

The current conceptual model defined for the Facility module is shown in Figure 7.

The classes and properties used to describe the topological concepts of a building in this module are based on the Building Topology Ontology (BOT), while other construction products, such as railways, bridges, and roads, are described by new classes and properties.

- facility:Project is defined as a large or major undertaking, especially one involving considerable money, personnel, and equipment. A facility:Project is related to a facility:Site.
- facility:SpatialZone is defined as a subclass of bot:Zone and, as such, a part of the physical or a virtual world that is inherently both located in this world and has a 3D spatial extent. This class is the root of a hierarchy that includes facility:ConstructionZone, used to represent zones that are relevant for the Health and Safety module, and facilityTrackedZone, used to represent zones that are relevant for the IoT preprocessing module. It is also the root of another hierarchy that includes facility:Site, facility:Facility, facility:FacilityPart, facility:Storey, and facility:Space.
- facility:Site is defined as a subclass of bot:Site and, as such, a part of the physical world or a virtual world that is inherently both located in this world and having a 3D spatial extent. It can contain (facility:hasFacility) one or more facility:Facility.
- facility:Space is defined as a subclass of bot:Space and, as such, a part of the physical world or a virtual world whose 3D spatial extent is bounded actually or theoretically and provides for certain functions within the zone it is contained in.
- facility:Facility is defined as something designed and built to serve a specific function that provides a convenience or a service. This new class includes facility:Railway, facility:Bridge, facility:Road, and facility:Building.
- facility:FacilityPart is defined as something that is usually contained (facility:hasFacilityPart) in a facility:Facility. A facility:FacilityPart can contain other facility parts (facility:hasFacilityPart).
- facility:Storey is defined as a subclass of bot:Storey and, as such, is contained (bot:hasStorey) in a facility:Building and is intended to contain (bot:hasSpace) one or more facility:Space that are horizontally connected.
- facility:Element is defined as a subclass or bot:Element and, as such, a constituent of a construction entity with a characteristic technical function, form, or position. A facility:Element can contain subelements (bot:hasSubElement). A facility:SpatialZone can contain (bot:hasElement) some facility:Elements. A beo:BuildingElement is a subclass of facility:Element. Its facility:material property is considered when processing visual quality.

The BOT subclasses have been created because they have specific properties such as rdfs:label and dcterms:identifier. They are also subclasses of geo:SpatialObject in order to reuse its location properties.

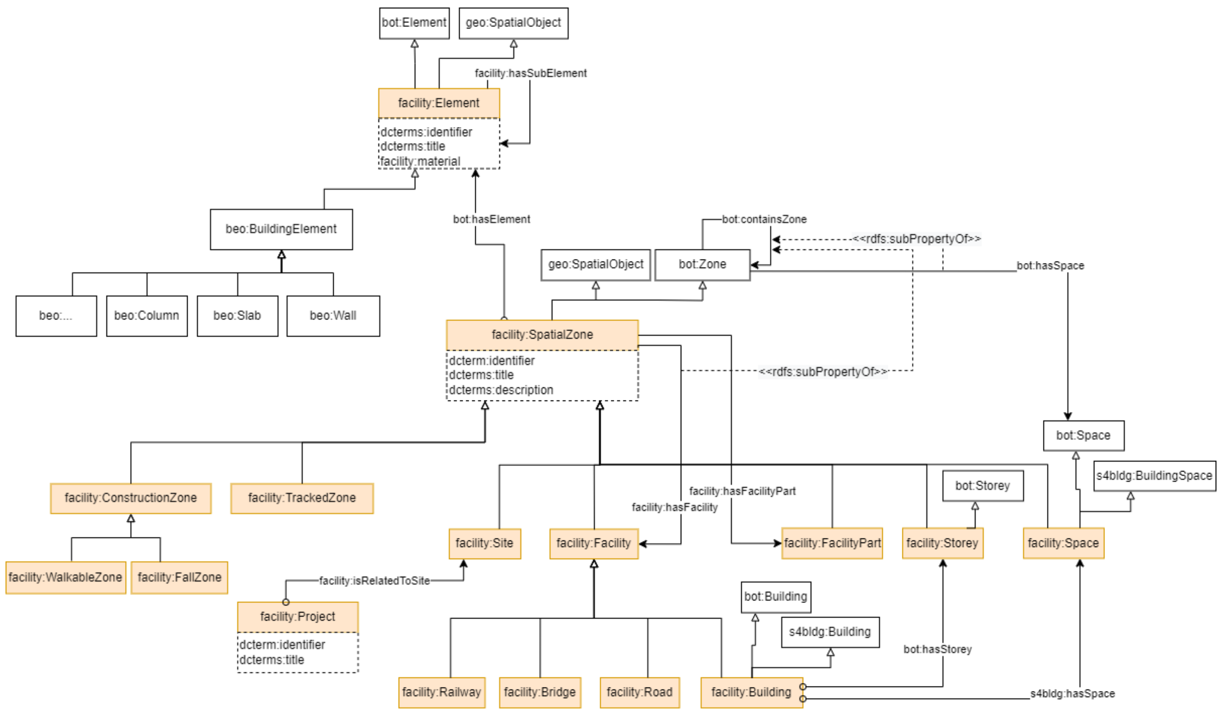


Figure 7 – General overview of the COGITO facility ontology

4.2 Process Module

The current conceptual model defined for the Process module is depicted in Figure 8. The main classes and properties of this module are the following:

- process:Process is defined as a series of actions aimed at accomplishing some result (in this case, related to a facility:Project). A process:Process can correspond to a process:AsPlannedOriginalProcess, a process:AsPlannedEnrichedProcess, or a process:AsIsProcess.
- process:Task is defined as a piece of work, which is performed in a process:Process (process:belongsTo); and can perform different actions (process:addsElement, process:removesElement, process:repairsElement, or processcontrolsElement) to a facility:Element depending on the type of task (process:GeometricQualityTask, process:VisualQualityTask, process:SafetyTask and process:ConstructionTask). A process:Task can have information about its planned and actual duration (process:plannedStartDate, process:plannedEndDate process:actualStartDate, and process:actualEndDate). We can include the process:status and the process:progress of a process:Task. A task can require (process:hasResourceTypeRequirement) several process:ResourceTypeRequirement including the process:quantityNeeded for a resource:ResourceType.
- process:Cost is defined as the price paid to acquire, produce, accomplish, or maintain anything (in this case, process:Process and process:Task); this price is measured (process:measuredIn) in a currency (process:UnitOfCurrency).
- process:WorkOrder is defined as a command or instruction authorizing specific work, repairs, etc. to be done. It has a main provider (resource:hasMainProvider), which is a resource:HumanWorker, and belongs to (process:hasWorkOrder) a process:Process. It is important to know if a process:WorkOrder is completed (process:status).

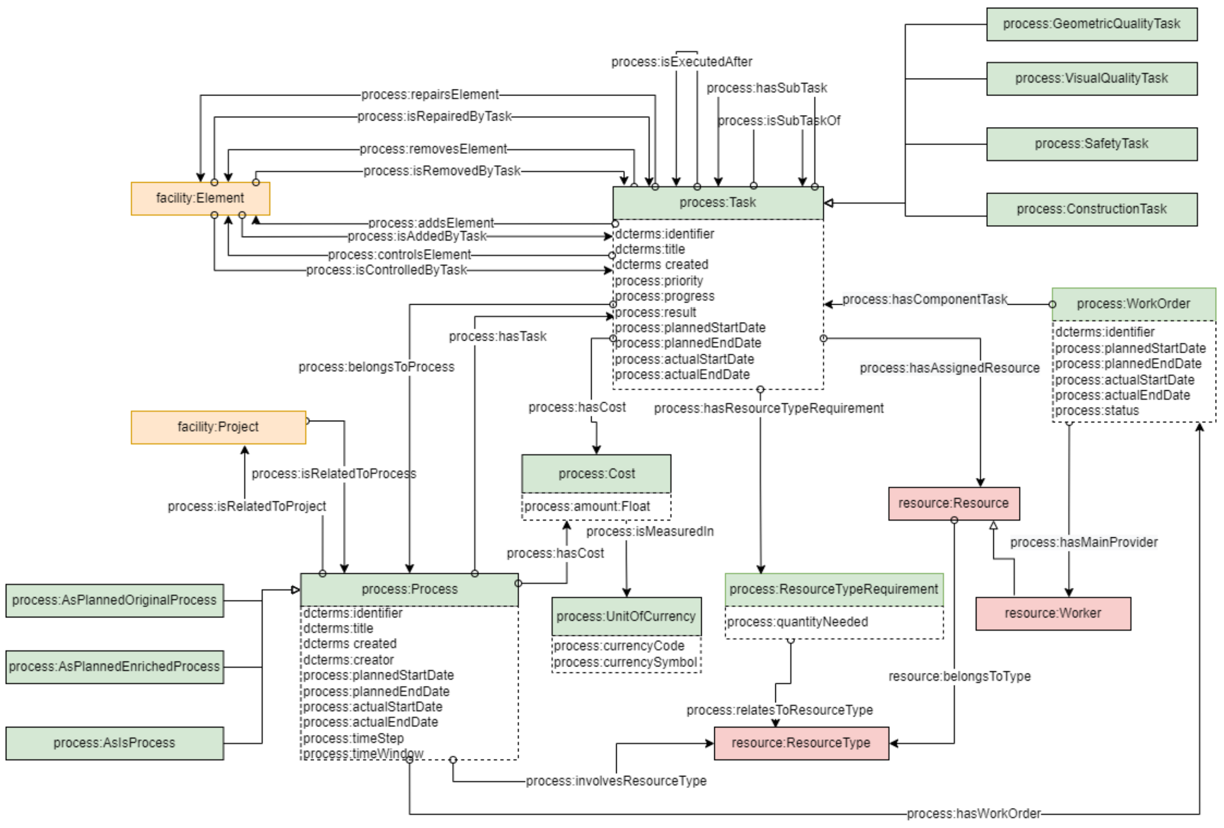


Figure 8 – General overview of the COGITO process ontology

4.3 Resource Module

The current conceptual model defined for the Resource module is shown in Figure 9. The main classes and properties of this module are the following:

- resource:Resource is defined as a source of supply, support, or aid, especially one that can be readily drawn upon when needed. resource:HumanWorker, resource:Equipment and resource:trackingTrack are subclasses of resource:Resource. It is also a subclass of geo:SpatialObject in order to reuse its location properties. A resource:Resource belongs to a resource:ResourceType and can be assigned (rersource:hasTrackingTag) an iot:TrackingTag.
- resource:ResourceType is defined as the kind of resource assigned to a process:Task or involved in a process:Process, indicating their maximum quantity (resource:maxUnit) and cost (resource:costPerHour). resource:HumanRole, resource:EquipmentType are subclasses of resource:ResourceType.

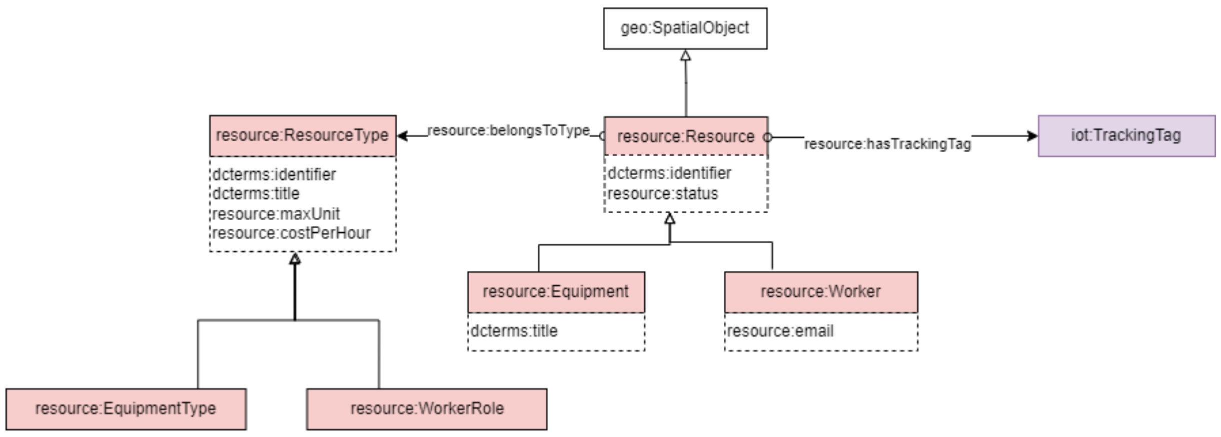


Figure 9 – General overview of the COGITO resource ontology

4.4 IoT Module

The current conceptual model defined for the Internet of Things (IoT) module is shown in Figure 10. The main classes and properties of this module are the following:

- An iot:TrackingTag is defined as a subclass of saref:Device and, as such, a tangible object designed to accomplish a particular task; in this case, signalling its location (iot:LocationMeasurement), which refers (iot:hasLocation, iot:hasLocationKF, and iot:hasLocationMA) to a geo:Point.

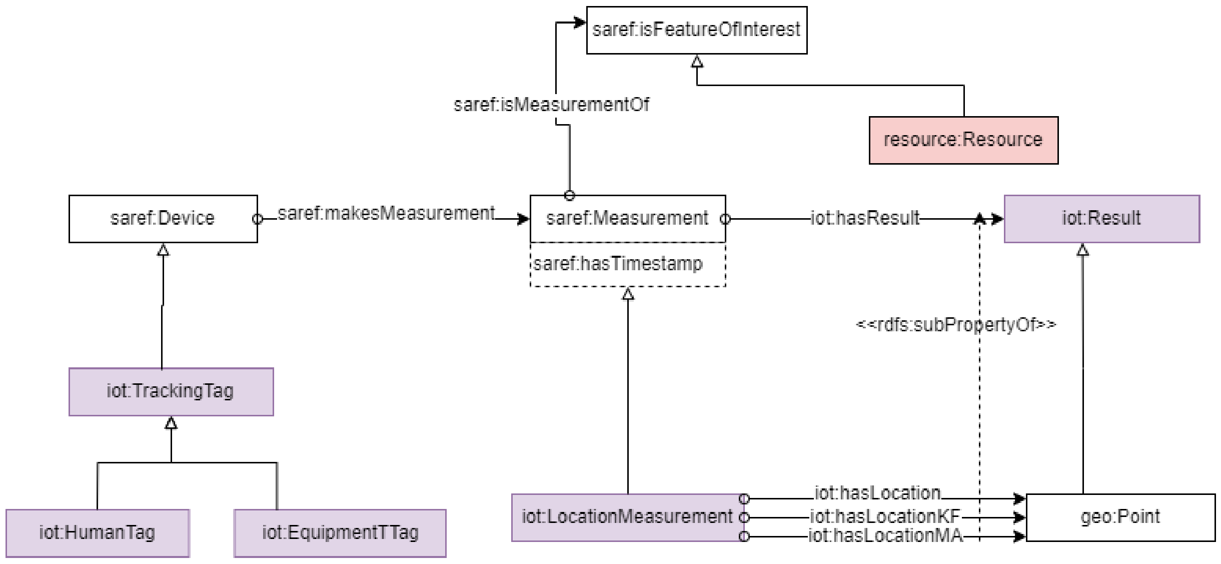


Figure 10 – General overview of the COGITO iot ontology

4.5 Quality Module

The current conceptual model defined for the Quality module is shown in Figure 11. The main classes and properties of this module are the following:

- qual:Defect is defined as a shortcoming, fault, or imperfection regarding a particular facility:Element. This qual:Defect is associated to a qual:ImageJob, which can be related to three qual:Image (qual:hasRawImage, qual:hasProcessedImage, qual:hasResultImage). qual:Image includes data regarding its capture device, position, orientation, and performed time.
- qual:GeometricQualityInformation is defined as data that inform a particular problem regarding a particular qual:Rule on a facility:Element. qual:GeometricQualityInformation includes data on its result, tolerance reference, scalar result, unit, and scheduled and performed time. This information comes from analysing a qual:PointCloud taken at a time. Data about a qual:Rule include an

identifier, an origin document, a label, a description and keywords related to element, material, and relationship types.

- As a generalisation, we can classify qual:Defect and qual:GeometricQualityInformation as qual:QualityResult, and qual:Mesh, qual:PointCloud, qual:Image, and qual:ImageJob as qual:InformationResource.

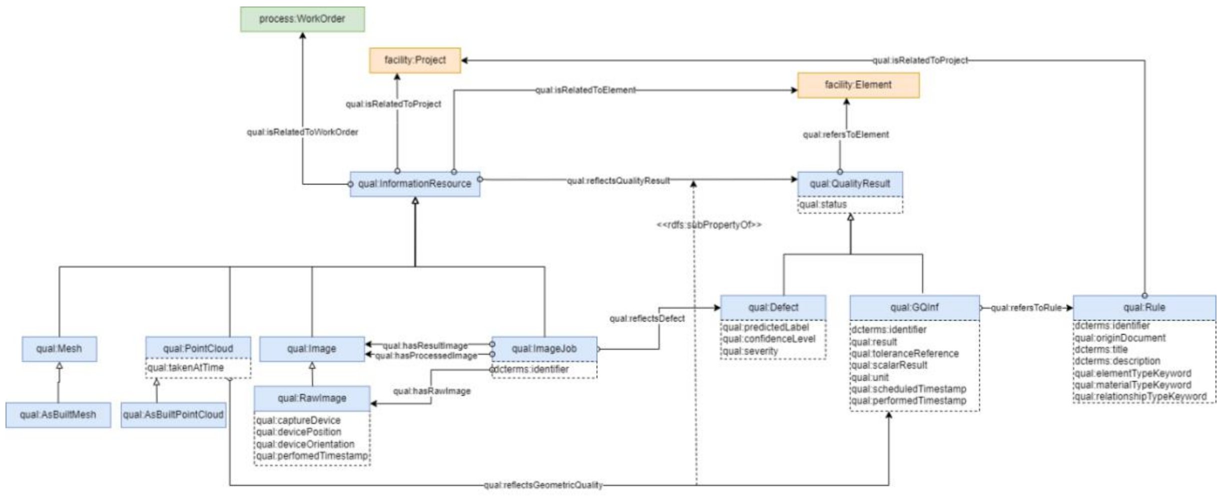


Figure 11 – General overview of the COGITO quality ontology

4.6 Safety Module

The current conceptual model defined for the Safety module is shown in Figure 12. The main classes and properties of this module are the following:

- safety:SafetyInformation is defined as data to prevent injuries by considering the characteristics of a facility:ConstructionZone.

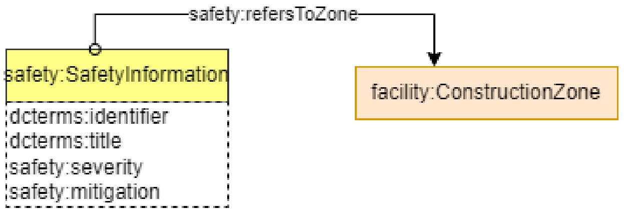


Figure 12 – General overview of the COGITO safety ontology

4.7 Platform Module

The current conceptual model defined for the Platform module is depicted in Figure 13. The main classes and properties of this module are the following:

- A platform:PlatformThing is a subclass of a td:Thing and, as such, an abstraction of a physical or a virtual entity whose metadata and interfaces are described by a WoT Thing Description. In this case, an entity of the digital twin platform.
- A platform:DigitalTwin is a subclass of a platform:PlatformThing including projects (facility:Project) and elements (facility:Element).
- A qual:InformationResource, is a subclass of a platform:PlatformThing including all the kinds of information resources used to derive the quality (see section 4.5) and that need to be stored.
- Subproperties of td:links and td:forms that specify the relations between a platform:PlatformThing and a td:Link and a td:Form have been created.

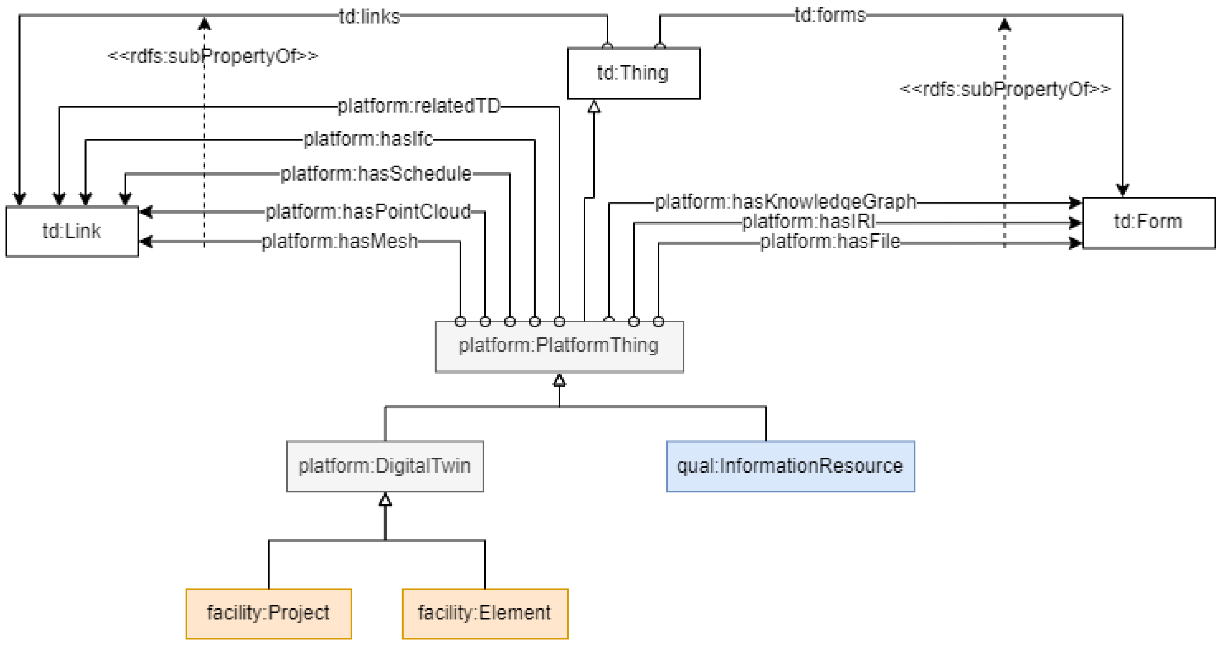


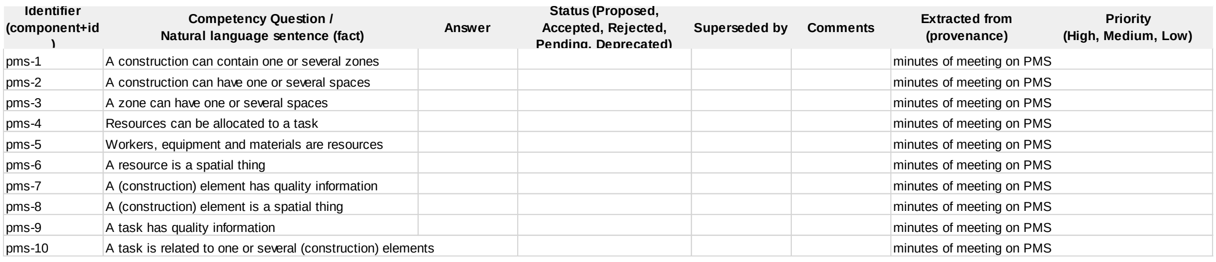
Figure 13 – General overview of the COGITO platform ontology

5 Ontology Development Infrastructure

This section describes the ontology development infrastructure used to support the activities in the ontology development process described above.

5.1 Infrastructure to Support the Requirements Specification Activity

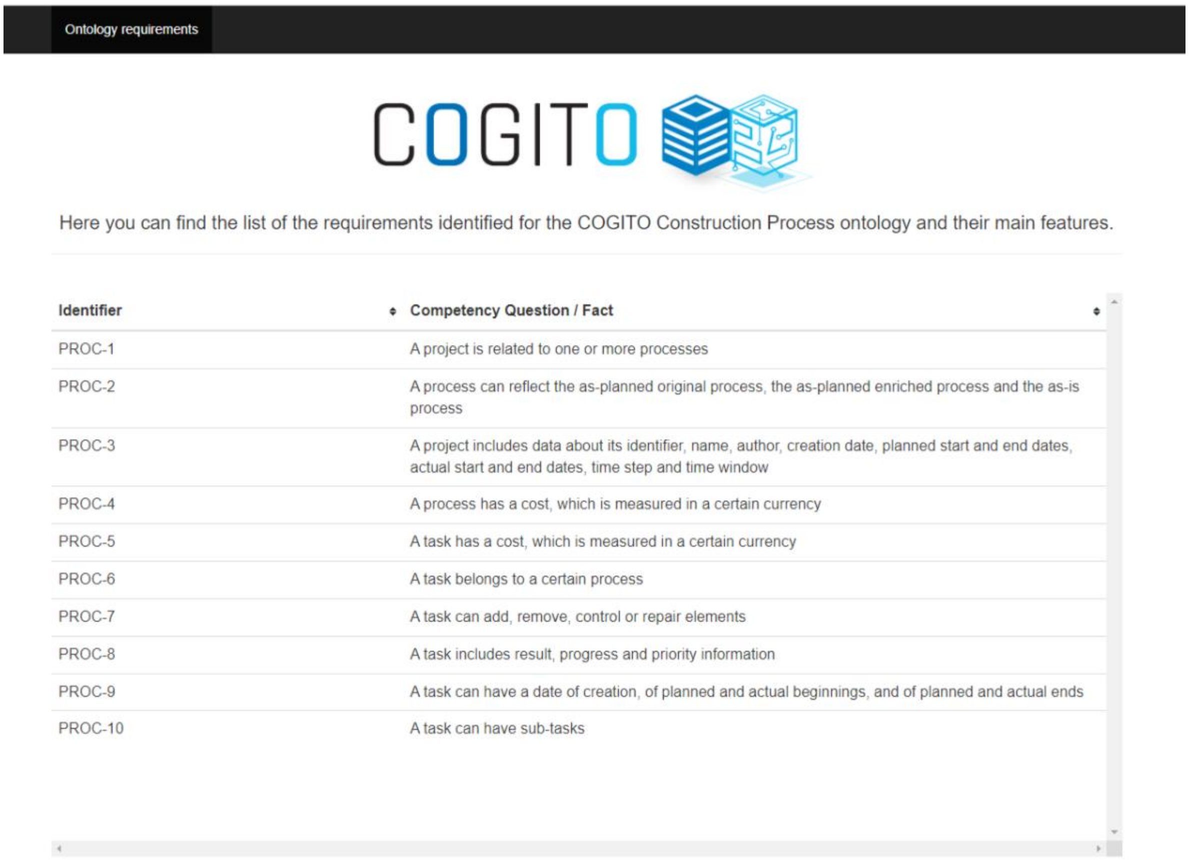
First, the ontology development team used spreadsheets to store requirements per component. An excerpt of these requirements is shown in Figure 14.



Identifier (component+id)	Competency Question / Natural language sentence (fact)	Answer	Status (Proposed, Accepted, Rejected, Pending, Deprecated)	Superseded by	Comments	Extracted from (provenance)	Priority (High, Medium, Low)
pms-1	A construction can contain one or several zones					minutes of meeting on PMS	
pms-2	A construction can have one or several spaces					minutes of meeting on PMS	
pms-3	A zone can have one or several spaces					minutes of meeting on PMS	
pms-4	Resources can be allocated to a task					minutes of meeting on PMS	
pms-5	Workers, equipment and materials are resources					minutes of meeting on PMS	
pms-6	A resource is a spatial thing					minutes of meeting on PMS	
pms-7	A (construction) element has quality information					minutes of meeting on PMS	
pms-8	A (construction) element is a spatial thing					minutes of meeting on PMS	
pms-9	A task has quality information					minutes of meeting on PMS	
pms-10	A task is related to one or several (construction) elements					minutes of meeting on PMS	

Figure 14 – COGITO ontology requirements for PMS

Then these requirements per component were reorganised per domain (see Section 3) and converted into an HTML file and uploaded to the COGITO ontology portal[5] with the most relevant information for users to facilitate visualization. Figure 15 shows an excerpt of the HTML documentation of the COGITO requirements.



Here you can find the list of the requirements identified for the COGITO Construction Process ontology and their main features.

Identifier	Competency Question / Fact
PROC-1	A project is related to one or more processes
PROC-2	A process can reflect the as-planned original process, the as-planned enriched process and the as-is process
PROC-3	A project includes data about its identifier, name, author, creation date, planned start and end dates, actual start and end dates, time step and time window
PROC-4	A process has a cost, which is measured in a certain currency
PROC-5	A task has a cost, which is measured in a certain currency
PROC-6	A task belongs to a certain process
PROC-7	A task can add, remove, control or repair elements
PROC-8	A task includes result, progress and priority information
PROC-9	A task can have a date of creation, of planned and actual beginnings, and of planned and actual ends
PROC-10	A task can have sub-tasks

Figure 15 – HTML requirements for the construction process

[5] https://cogito.iot.linkeddata.es/

5.2 Infrastructure to Support the Ontology Implementation Activity

To support the implementation activity, the ontology development team uses several tools to edit, store, and evaluate the ontology.

- For ontology edition, the ontology development team uses Protégé, [6] which allows the creation, visualisation, and manipulation of ontologies.
- For ontology storage, the ontology development team uses GitHub[7]. A GitHub repository is created for each ontology in the COGITO ontology network. Each repository includes:
  - A folder with the implementation of the ontology.
  - A folder with the ontology modelling diagrams.
  - A folder with the documentation of the ontology.
  - A folder with the requirements and tests of the ontology.

The development team uses the OnToology[8] tool to generate documentation and evaluate the ontology. OnToology, which integrates the tools Widoco[9] [12] and OOPS![10] [13], automatically generates a folder in the GitHub repository that includes the resources: diagrams, documentation, and evaluation report.

5.3 Infrastructure to Support the Ontology Publication Activity

To support the publication activity, the ontology development team creates an ontology portal online to make the ontology and all the associated information (repository, requirements, tests, releases, etc.) to users. This ontology portal has different sections:

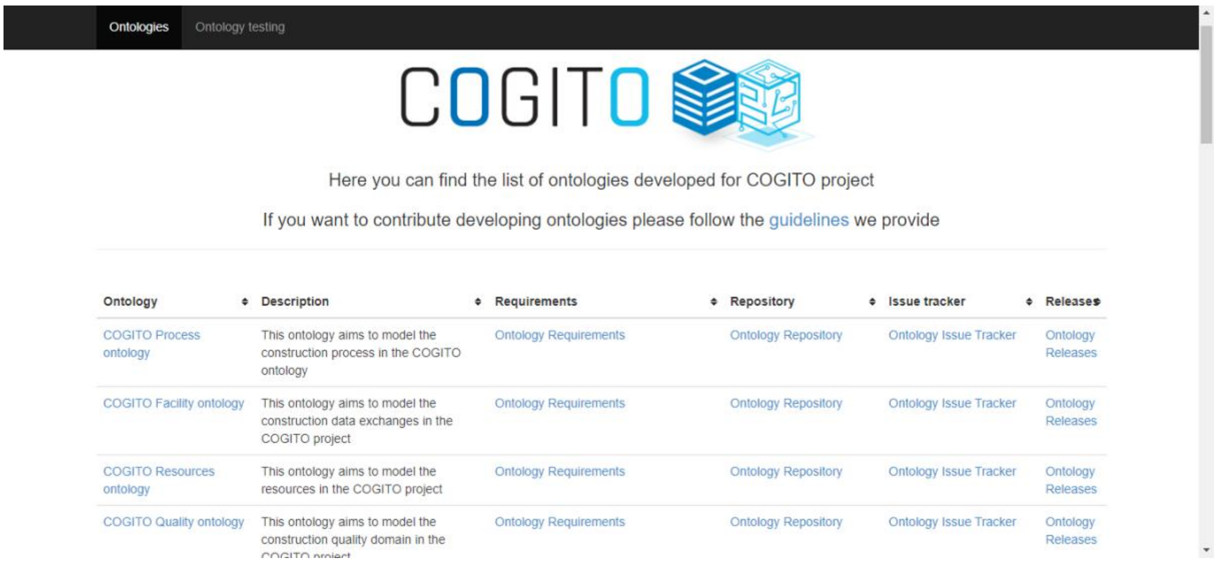
- Ontologies
- How we work

5.3.1.1 Ontologies

The Ontologies section, which is the main section of the portal, shows the main information about the ontologies created in the COGITO ontology network. Figure 16 shows an overview of the information exposed in this section of the portal. The section follows a tabular approach that includes the following:

- Link to the ontology documentation published on the Web
- Ontology Description
- Link to each GitHub repository
- Links to each GitHub issue tracker
- HTML description of the requirements identified by the domain experts
- Link to each ontology release

[6] https://protege.stanford.e
[7] https://github.com/orgs/oeg-upm/teams/cogito
[8] https://ontoology.linkeddata.es/
[9] https://github.com/dgarijo/Widoco
[10] http://oops.linkeddata.es/



Here you can find the list of ontologies developed for COGITO project

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

Ontology	Description	Requirements	Repository	Issue tracker	Releases
COGITO Process ontology	This ontology aims to model the construction process in the COGITO ontology	Ontology Requirements	Ontology Repository	Ontology Issue Tracker	Ontology Releases
COGITO Facility ontology	This ontology aims to model the construction data exchanges in the COGITO project	Ontology Requirements	Ontology Repository	Ontology Issue Tracker	Ontology Releases
COGITO Resources ontology	This ontology aims to model the resources in the COGITO project	Ontology Requirements	Ontology Repository	Ontology Issue Tracker	Ontology Releases
COGITO Quality ontology	This ontology aims to model the construction quality domain in the COGITO project	Ontology Requirements	Ontology Repository	Ontology Issue Tracker	Ontology Releases

Figure 16 – Ontology section in the COGITO ontology portal

5.3.1.2 How We Work

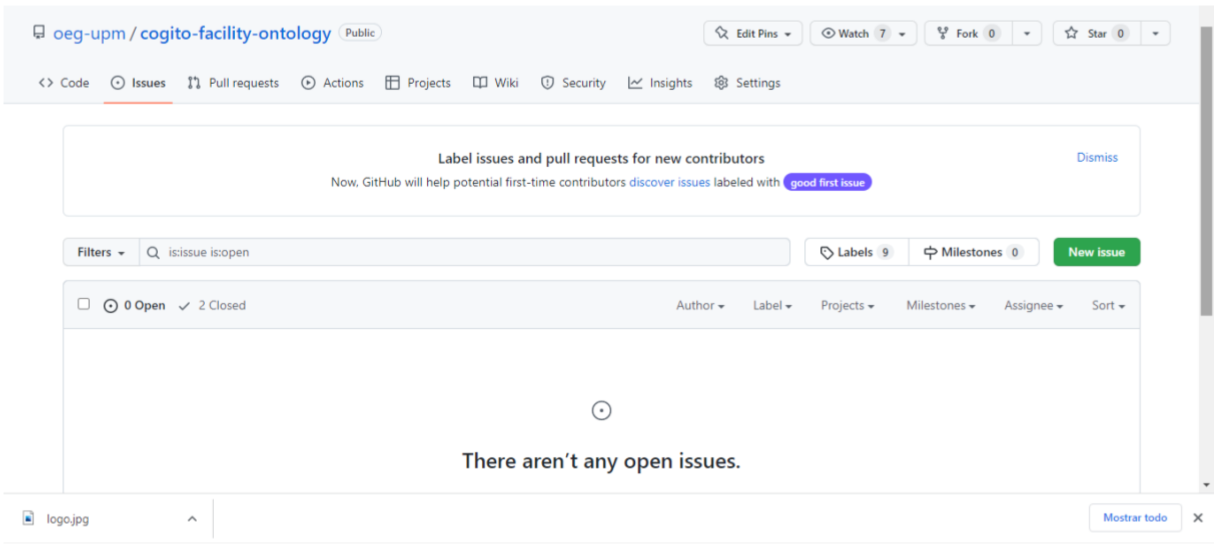
Regarding the section "how we work", it provides a brief overview of the proposed process for developing ontologies and some guidelines, which should be useful to anyone who wants to contribute to the ontologies. This section includes information about the following.

- How should the repository be structured.
- Tools recommended to be used during the ontology development process.
- Ontology versioning.
- Management of issues.

5.4 Infrastructure to Support the Ontology Maintenance Activity

To provide support for maintenance activity, ontology developers use the GitHub issue tracker, which manages and maintains the list of issues identified by domain experts and ontology developers. The GitHub issue tracker provides the status of the issue, assignee, and description, and allows one to add comments to the issue to discuss about it. Each issue tracker is associated with a GitHub repository and consequently with an ontology in the COGITO ontology network.

To manage changes in the ontology, all new proposals and improvements must be agreed upon by all members of the ontology development team. If domain experts, users, or ontology developers want to add, delete, or modify concepts in the ontology, they must create a new issue in the GitHub issue tracker associated with the ontology to be modified, which will be used to discuss the approval or rejection of the proposal. Figure 17**Error! Reference source not found.** shows the GitHub issue tracker for one of the COGITO ontologies.


oeg-upm / cogito-facility-ontology Public

Label issues and pull requests for new contributors

Now, GitHub will help potential first-time contributors discover issues labeled with good first issue

is:issue is:open

0 Open ✓ 2 Closed

There aren't any open issues.

Figure 17 – Example of the GitHub issue tracker

6 Conclusions

This document details the methodology and technological infrastructure used to develop the COGITO ontology network, as well as the third version of the ontologies that make up such an ontology network. The development of these ontologies has been aligned with the gathering of requirements and the specification of the COGITO architecture.

The current version of the COGITO ontology network (see Figure 6) consists of seven modules, corresponding to the facility (the construction itself), the construction process, the construction resources, the IoT devices used to locate the resources, the quality achieved, the safety information related to the facility, and the entities managed by the digital twin platform (see Table 3). These modules have already been conceptualised, implemented, and published in a third version of the COGITO ontologies.

In D3.2 [2] a coverage analysis was carried out identifying to what extent some of the terms appearing in the requirements also appeared in the ontologies described in the previous D3.1 deliverable [4]. The conclusions highlighted the fact that reusing ontologies requires analysing multiple factors beyond coverage, and some of them are not easily quantifiable or even cannot be measured objectively. Therefore, no new coverage analysis was executed for the new ontology requirements.

The COGITO ontology network allows for the sharing and interoperability among the different components of the COGITO architecture, especially in the communications with the Digital Twin platform. That is why this document has been so important when developing each component of the COGITO architecture, that is, the results of work packages 4, 5, 6, and 7, and their integration in work package 8.

During the maintenance phase of the ontology development process, which will start after this deliverable is submitted, we expect future change requests in the ontologies due to defects found or to new requirements that may appear during integration. The COGITO ontology portal will serve as a living repository for the different ontologies and will contain the latest version of the developed ontologies and all related artifacts.

References

[1] M. Poveda-Villalón, A. Fernández-Izquierdo and R. García-Castro, “Linked Open Terms (LOT) Methodology,” January 2019. [Online]. Available: http://doi.org/10.5281/zenodo.2539305.

[2] “D3.2. COGITO Data Model and Ontology Definition and Interoperability Design,” 2021.

[3] “D3.3. COGITO Data Model and Ontology Definition and Interoperability Design v2,” 2022.

[4] “D3.1- Survey of Existing Models, Ontologies and Associated Standardization Efforts,” 2021.

[5] “D2.1 - Stakeholder Requirements for the COGITO System,” 2021.

[6] “D2.4- COGITO System Architecture,” 2021.

[7] M. C. Suárez-Figueroa, A. Gómez-Pérez and M. Fernández-López, “The NeOn Methodology for Ontology Engineering,” in *Ontology engineering in a networked world*, Springer, Berlin, Heidelberg, 2011.

[8] R. García-Castro, A. Fernández-Izquierdo, C. Heinz, P. Kostelnik, Poveda-Villalón-María and F. Serena, “D2.2. Detailed Specification of the Semantic Model,” 2017.

[9] S. Chávez, F. Bosché, N. Bountouni, S. Fenz, R. García-Castro, G. Giannakis, S. González-Gerpe, S. Kollmer, S. Kousouris, D. Ntalaperas, T. Thanos, F. Lampathaki, M. Poveda-Villalón, E. Valero, D. Vergeti and F. Tavakolizadeh, “D4.2 BIMERR Ontology & Data Model,” 2020.

[10] A. Fernández-Izquierdo, A. Cimmino, R. García-Castro, M. Poveda-Villalón, S. Terzi and C. Patsonakis, “T1.3 DELTA Ontology and Data Modelling Framework,” 2019.

[11] D. Garijo and M. Poveda-Villalón, “Best Practices for Implementing FAIR Vocabularies and Ontologies on the Web,” in *Applications and Practices in Ontology Design, Extraction, and Reasoning*, IOS Press, 2020.

[12] D. Garijo, “WIDOCO: a wizard for documenting ontologies,” in *International Semantic Web Conference*, 2017.

[13] M. a. S.-F. M. C. a. G.-P. A. Poveda-Villalón, “Validating ontologies with oops!,” 2012.

[14] ISO 16739, “Industry Foundation Classes (IFC) for data sharing in the construction and facility management industries (ISO 16739:2013),” CEN, 2016.

[15] ISO 13790, “Energy performance of buildings — Calculation of energy use for space heating and cooling,” 2008.

[16] M. C. Suárez-Figueroa, A. Gómez-Pérez and B. Villazón-Terrazas, “How to write and use the ontology requirements specification document,” in *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, 2009.

[17] A. Fernández-Izquierdo and R. García-Castro, “Conformance testing of ontologies through ontology requirements,” *Engineering Applications of Artificial Intelligence,* vol. 97, 2021.

[18] M. H. Rasmussen, P. Pauwels, C. Hviid and J. Karlshøj, “Proposing a Central AEC Ontology That Allows for Domain Specific Extensions,” in *Joint Conference on Computing in Construction*, 2017.

[19] A. Annane, N. Aussenac-Gilles and M. Kamel, “BBO: BPMN 2.0 Based Ontology for Business Process Representation,” in *20th European Conference on Knowledge Management (ECKM'19)*, Lisbon, 2019.

[20] “D7.2-Digital Twin Platform Design & Specification v2”.

Annex A

Table 5 – Relations between the main classes of the COGITO ontology and ifcOWL

COGITO	IFC	comment
facility:Building An independent unit of the built environment with a characteristic spatial structure, intended to serve at least one function or user activity [ISO-12006]. A bot:Building is a part of the physical world or a virtual world that is inherently both located in this world and having a 3D spatial extent, is contained in a building site, and can contain one or more storeys that are vertically connected.	**IfcBuilding** A building represents a structure that provides shelter for its occupants or contents and stands in one place. The building is also used to provide a basic element within the spatial structure hierarchy for the components of a building project (together with site, storey, and space).	Sub-class
facility:ConstructionZone Is defined as a subclass of facility:SpatialZone used to represent zones used by the Safety module.		
facility:Element Is defined as a subclass or bot:Element and, as such, a constituent of a construction entity with a characteristic technical function, form, or position. A facility:Element can contain subelements (bot:hasSubElement).	**IfcElement** An element is a generalization of all components that make up an AEC product.Elements are physical existent objects, although they might be void elements, such as holes. Elements either remain permanently in the AEC product or only temporarily, as formwork does.	Sub-class
facility:Facility Is defined as something designed and built to serve a specific function providing a convenience or a service.		
facility:FacilityPart Is defined as something that is contained (facility:hasFacilityPart) in a facility:Facility. A facility:FacilityPart can contain subfacility parts (facility:hasSubFacilityPart).		
facility:Project Is defined as a large or major undertaking, especially one that involves considerable money, personnel, and equipment.	**IfcProject** IfcProject indicates the undertaking of some design, engineering, construction, or maintenance activities leading towards a product. The project establishes the context for information to be exchanged or shared, and it may represent a construction project but does not have to.	Different concepts
facility:Site A part of the physical world or a virtual world that is inherently both located in this world and having a 3D spatial extent.	**IfcSite** A site is a defined area of land, possibly covered with water, on which the construction of the project is to be completed. A site may be used to erect,	Sub-class

It contains (bot:containsZone) one or more facility:Facility.	retrofit, or turn down building(s), or for other construction related developments.	
facility:Space Is defined as a subclass of bot:Space and, as such, a part of the physical world or a virtual world whose 3D spatial extent is bounded actually or theoretically and provides for certain functions within the zone it is contained in.	**IfcSpace** A space represents an area or volume that is actually or theoretically bounded. Spaces are areas or volumes that provide certain functions within a building. A space is associated with a building storey (or in the case of exterior spaces with a site). A space may span several connected spaces. Therefore, a space group provides for a collection of spaces included in a storey.	Sub-class
facility:SpatialZone Is defined as a subclass of bot:Zone and, as such, a part of the physical or a virtual world that is inherently both located in this world and has a 3D spatial extent. This class is the root of a hierarchy that includes facility:**ConstructionZone**, used to represent zones used by the Health and Safety module, and facility**TrackedZone**, used to represent zones used by the IoT preprocessing module.	**IfcSpatialZone** A spatial zone is a nonhierarchical and potentially overlapping decomposition of the project under some functional consideration. A spatial zone might be used to represent a thermal zone, a construction zone, a lighting zone, or a usable area zone. A spatial zone might have an independent placement and a shape representation. **ifcZone** A zone is a group of spaces, partial spaces, or other zones. Zone structures may not be hierarchical (in contrary to the spatial structure of a project - see IfcSpatialStructureElement), i.e. one individual IfcSpace may be associated with zero, one, or several IfcZone's.	Related concetps
facility:Storey Is defined as a subclass of bot:Storey and, as such, is contained (bot:hasStorey) in one facility:Building and is intended to contain (bot:hasSpace) one or more facility:Space that are horizontally connected.	**IfcBuildingStorey** The building storey has an elevation and typically represents a (nearly) horizontal aggregation of spaces that are vertically bound. A storey is (if specified) associated to a building.	Sub-class
process:Cost Is defined as the price paid to acquire, produce, accomplish, or maintain anything (in this case, process:Process and process:Task); and this price is measured (process:measuredIn) in a currency (process:UnitOfCurrency).	**IfcCostSchedule** The IfcCostSchedule may have assignments of its own using the IfcRelAssignsToControl relationship, where RelatingControl refers to the IfcCostSchedule and RelatedObjects contains one or more objects. **IfcCostItem**: Indicates costs published within this cost schedule, typically a single root cost item forming a hierarchy of nested cost items.	Different concepts
process:Process Is defined as a series of actions aimed at accomplishing some result (in this case, related to a facility:Project).	**IfcProcess** IfcProcess is defined as one individual activity or event that is ordered in time, that has sequence relationships with other	Different concepts

	processes, which transforms input in output, and may connect to other processes through input output relationships. An IfcProcess can be an activity (or task), or an event. It usually takes place in building construction with the intent of designing, costing, acquiring, constructing, or maintaining products or other similar tasks or procedures.	
process:Task Is defined as a piece of work which is performed in a process:Process (process:belongsTo); and can perform different actions (process:addsElement, process:removesElement, process:repairsElement or processcontrolsElement) to a facility:Element depending on the type of task (process:GeometricQualityTask, process:VisualQualityTask, process:SafetyTask and process:ConstructionTask). A process:Task can have information about its planned and actual duration (process:plannedStartDate, process:plannedEndDate process:actualStartDate, and process:actualEndDate). We can include the process:status and the process:progress of a process:Task. A task can require (process:hasResourceTypeRequirement) several process:ResourceTypeRequirement including the process:quantityNeeded for a resource:ResourceType.	**IfcTask** An IfcTask is an identifiable unit of work to be carried out in a construction project. A task is typically used to describe an activity for the construction or installation of products, but is not limited to these types. For example, it might be used to describe design processes, move operations, and other design, construction, and operation related activities as well. The quantities of resources consumed by the task are dealt with by defining the IfcElementQuantity for the resource and not in the instance of IfcTask.	Related concepts
process:WorkOrder Is defined as a command or instruction authorizing specific work, repairs, etc., to be done. It has a main provider (resource:hasMainProvider), which is a resource:HumanWorker, and belongs to (process:hasWorkOrder) a process:Process. There can be some qual:Image and qual:PointCloud related to a process:WorkOrder. It is important to know if a process:WorkOrder is completed (process:status).	**IfcWorkPlan** A work plan contains a set of work schedules for different purposes (including construction and facility management). Contained work schedules are defined through the IfcRelAggregates relationship. Through inheritance from IfcWorkControl, it is also possible to define references to activities (for example, IfcTask) and resources used in the work plan. A work plan has information such as start date, finish date, total free float, etc. IfcWorkPlan can also refer to the construction project represented by the single IfcProject instance (please also check the definition of IfcWorkControl).	Related concepts
qual:Defect		

Is defined as a shortcoming, fault, or imperfection regarding a particular facility:Element.		
qual:GeometricQualityInformation Is defined as data informing a particular problem regarding a particular qual:Rule on a facility:Element.		
resource:Equipment Is a subclass of resource:Resource used to assist in the performance of an activity (process:Task).	**IfcConstructionEquipmentResource** IfcConstructionEquipmentResource is the usage of construction equipment to assist in the performance of construction. Construction equipment resources are consumed entirely or partially or occupied in the performance of construction.	Sub-class
resource:EquipmentType Is defined as the different kinds of construction equipment.	**ifcConstructionEquipmentResourceType** The resource type IfcConstructionEquipmentType defines commonly shared information for occurrences of construction equipment resources. The set of shared information may include: common productivities, common cost rates, or common properties within shared property sets. It is used to define a construction equipment resource specification (the specific information about the resource that is common to all occurrences of that resource). Resource types may be exchanged without being already assigned to occurrences.	Sub-class
resource:WorkerRole Is defined as the different kinds of roles a human worker can play in a construction project.	**IfcLaborResourceType** The resource type IfcLaborResourceType defines commonly shared information for occurrences of labour resources. The set of shared information may include: common productivities, common cost rates, or common properties within shared property sets. It is used to define a labour resource specification (the specific resource information that is common to all occurrences of that resource). Resource types may be exchanged without being already assigned to occurrences.	Sub-class
resource:Worker Is defined as a person who works on a construction project.	**IfcLaborResource** An IfcLaborResource is used in construction with particular skills or crafts required to perform certain types of construction or management related work. The purpose of an IfcLaborResource is to identify a skill set that may be required or used.	Sub-class
resource:Resource	**IfcConstructionResource**	Sub-class

Is defined as a source of supply, support, or aid, especially one that can be readily drawn upon when needed. resource:HumanWorker, resource:Equipment and resource:trackingTrack are subclasses of resource:Resource. It is also a subclass of geo:SpatialThing in order to reuse its location properties. A resource:Resource belongs to a resource:ResourceType.	IfcConstructionResource is an abstract generalization of the different resources used in construction projects, mainly labour, material, equipment, and product resources, plus subcontracted resources and aggregations such as a crew resource. **IfcResource** contains the information needed to represent the costs, schedule, and other impacts from the use of something in a process. It is not intended to use IfcResource to model the general properties of the things themselves, while an optional linkage from IfcResource to the things to be used can be specified (specifically, the relationship from the subtypes of IfcResource to IfcProduct through the IfcRelAssignsToResource relationship).	
resource:ResourceType Is defined as the kind of resources assigned to a process:Task or involved in a process:Process, indicating their maximum quantity (resource: maxUnit) and cost (resource:costPerHour). resource:HumanRole, resource:EquipmentType and resource:trackingTrackGroup are subclasses of resource:ResourceType.	**IfcConstructionResourceType** IfcConstructionResourceType is an abstract generalization of the different resource types used in construction projects, mainly labour, material, equipment, and product resource types, plus subcontracted resource types and aggregations such as a crew resource type.	Related concepts
	TypeResource IfcTypeResource defines a specific (or type) definition of a resource. It is used to define a resource specification (the specific resource that is common to all occurrences that are defined for that resource) and could act as a resource template. An IfcTypeResource may have a list of property sets attached. Values of these properties are common to all occurrences of that resource type. The type of occurrence relationship is realized using the objectified relationship IfcRelDefinesByType.	Sub-class
iot:TrackingTag Is defined as a subclass of saref:Device and, as such, a tangible object designed to accomplish a particular task; in this case, signalling its location.		
safe:SafetyInformation Is defined as data to prevent injuries by considering the characteristics of a facilityConstructionZone.		

COGITO

CONSTRUCTION PHASE
DIGITAL TWIN MODEL

cogito-project.eu

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 958310